
AWS
re:Invent

AIM316

How to build personalization models using Amazon SageMaker

Chris Fregly

Developer Advocate, AI and Machine Learning

AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

Why should you personalize your application?

What are popular recommender algorithms?

Why build recommender models on AWS?

Which options to train and tune recommenders with AWS?

Demo: Build & deploy TensorFlow multi-task recommender

How can I scale to a million recommender models?

Summary



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Why should you personalize your application?



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Personalization is everywhere: Don’t be left behind

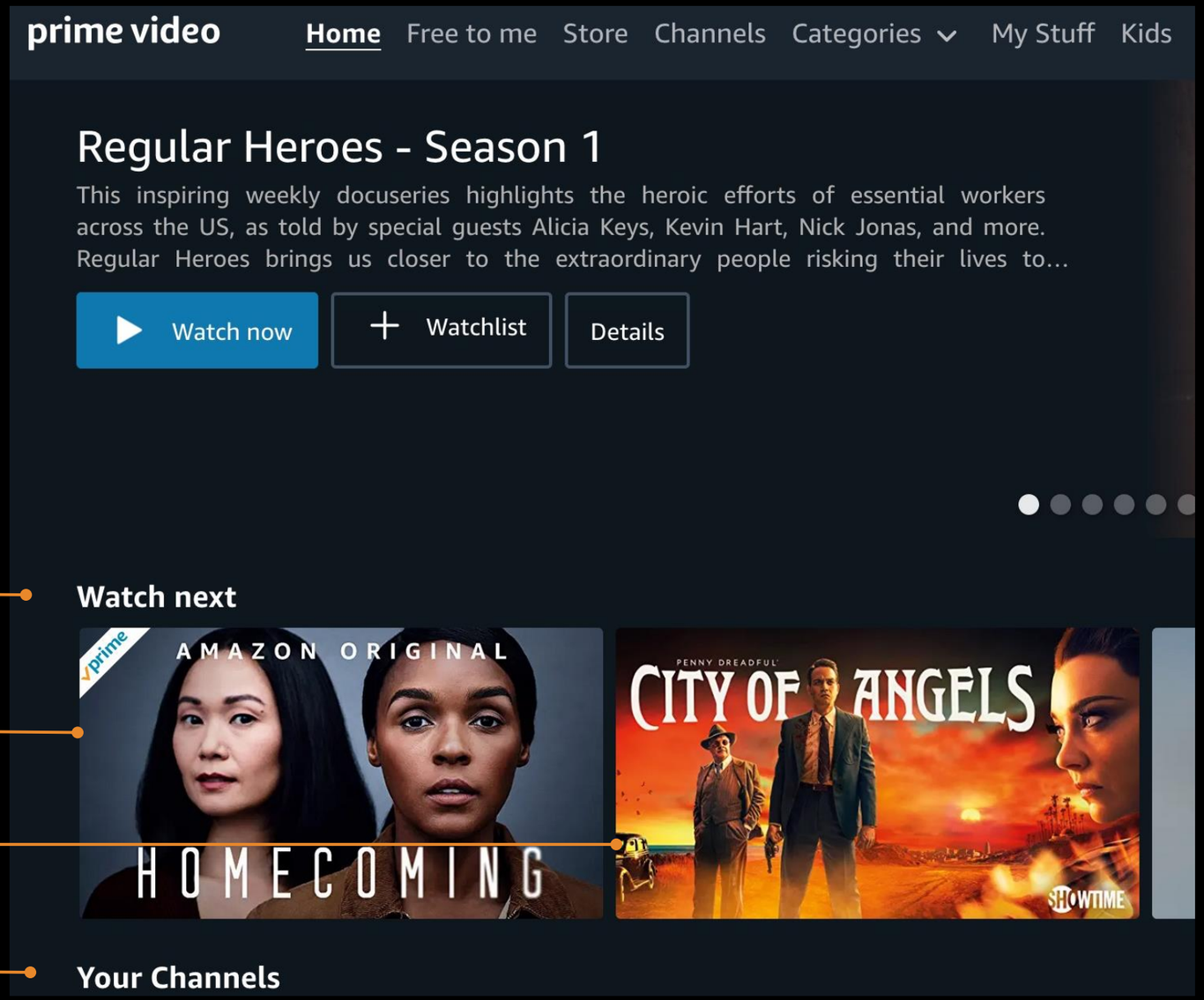
PERSONALIZATION IS NOW INCORPORATED INTO EVERY ASPECT OF THE CUSTOMER EXPERIENCE

Movie recommendations

Images and expressions

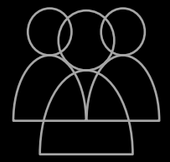
Time-of-day ranking

Customized channels



Personalization offers material business benefits

EVERYBODY'S TASTES, PREFERENCES, AND NEEDS ARE DIFFERENT



Acquire new customers and retain existing customers



Help customers discover products and services they need



Improve customer engagement with your website, application, and content



Increase revenue for your business



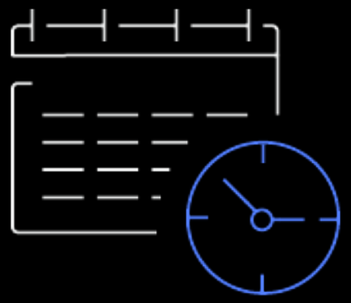
Increase conversions through recommendations – movies, restaurants, music, etc.

Personalized recommendations

IMPROVE YOUR CUSTOMER EXPERIENCE THROUGH BETTER RECOMMENDATIONS



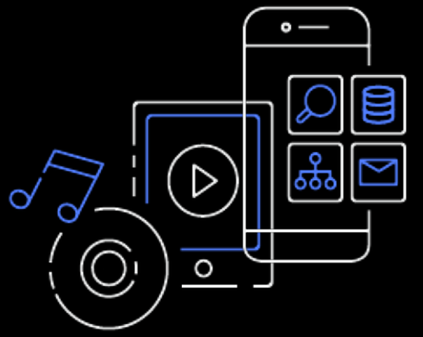
Deliver high-quality recommendations



Adapt to changes in customer intent in real time



Train a recommendation model with a few clicks



Generate recommendations for almost any product or content

Challenges for building effective recommenders

MULTIPLE HARD PROBLEMS NEED TO BE ADDRESSED



Real-time

Personalization must be responsive to the changing user intent



Cold starts

New users should get relevant recommendations; new items should show in recommendations



Popularity trap

Naïve models give recommendations similar to popular items



Scale

Recommendations should scale across millions of users and items



Custom models are required

Personalization models must accurately reflect business context and user behavior

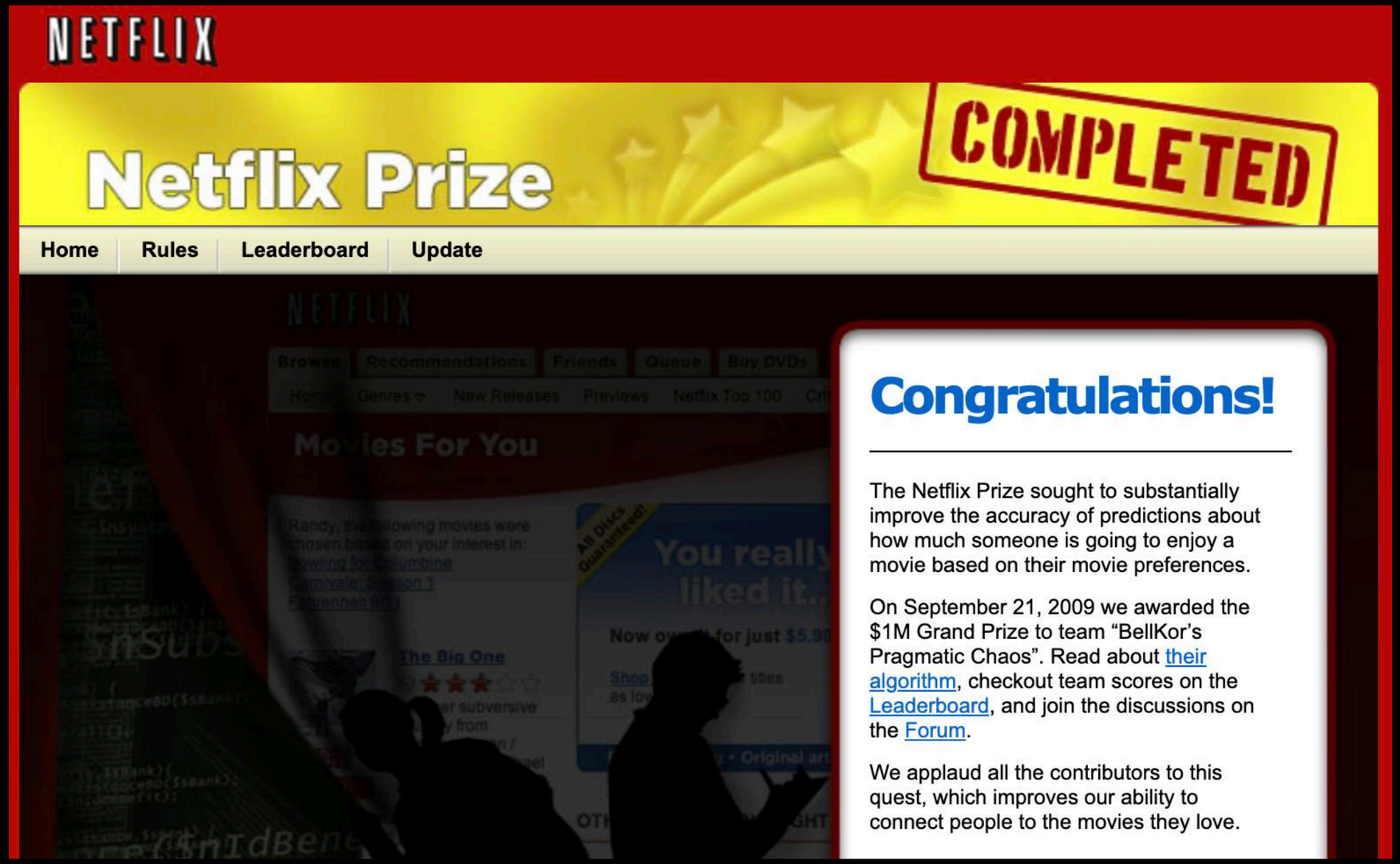
What are popular recommender algorithms?



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Netflix prize (2006–2009)

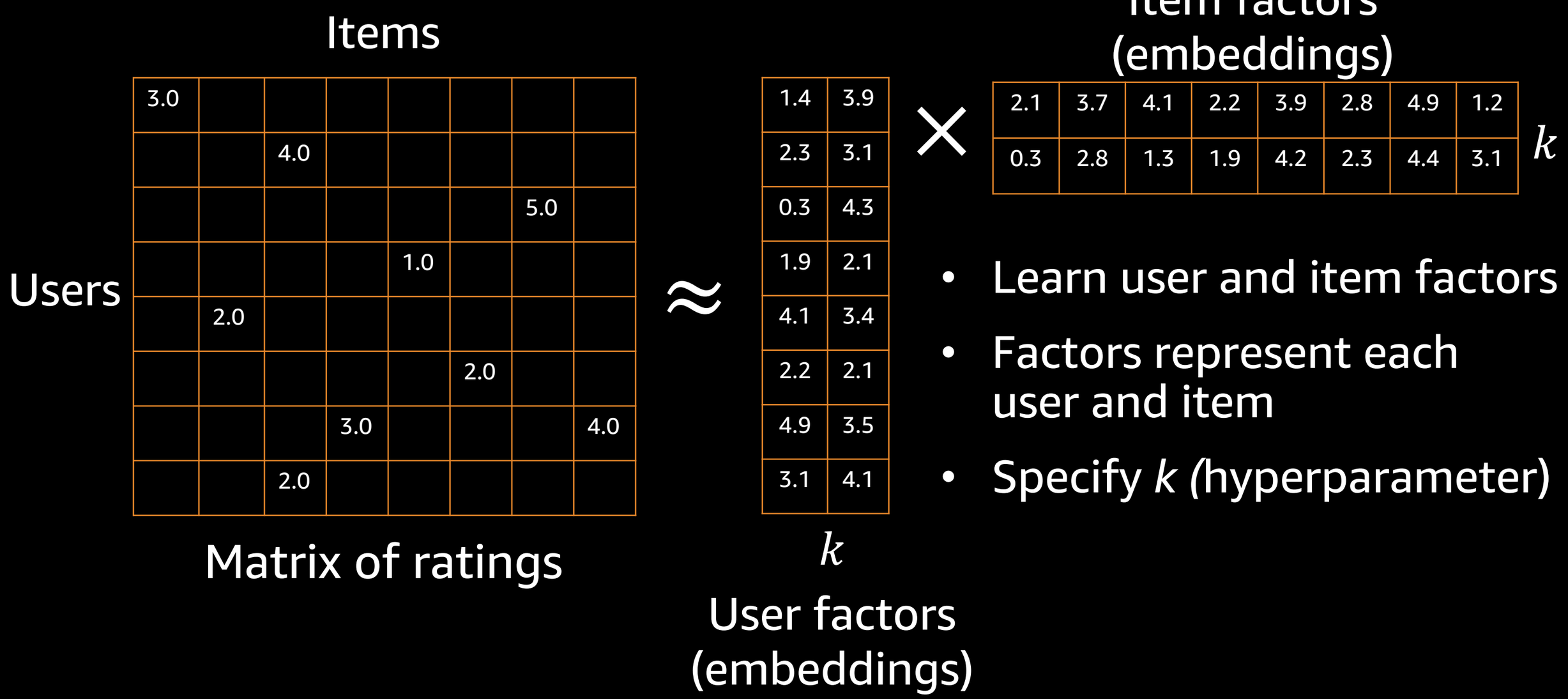
ENSEMBLE INCLUDING COLLABORATIVE FILTERING WITH MATRIX FACTORIZATION WINS THE PRIZE



Source: https://www.netflixprize.com/

Generate recommendations using matrix factorization

FACTOR THE RATING MATRIX INTO SEPARATE USER AND ITEM MATRICES; POPULATE THE MISSING RATINGS



- Learn user and item factors
- Factors represent each user and item
- Specify *k (*hyperparameter)

Generate recommendations using matrix factorization

FACTOR THE RATING MATRIX INTO SEPARATE USER AND ITEM MATRICES; POPULATE THE MISSING RATINGS

Items

Users

3.0	2.3	4.6	1.2	4.1	3.5	4.9	1.2
4.1	3.5	4.0	4.9	1.2	1.9	4.6	2.3
4.8	3.1	2.8	1.9	2.3	2.5	5.0	2.1
4.1	2.9	2.2	1.2	1.0	5.0	3.5	3.9
1.2	2.0	3.5	5.0	2.9	4.6	2.1	4.4
4.6	2.8	3.3	3.9	2.2	2.0	3.5	2.8
2.3	5.0	4.9	3.0	3.5	1.2	4.6	4.0
2.8	3.1	2.0	1.2	3.6	4.9	2.2	4.1

Matrix of ratings

≈

1.4	3.9
2.3	3.1
0.3	4.3
1.9	2.1
4.1	3.4
2.2	2.1
4.9	3.5
3.1	4.1

k

User factors (embeddings)

×

Item factors (embeddings)

2.1	3.7	4.1	2.2	3.9	2.8	4.9	1.2
0.3	2.8	1.3	1.9	4.2	2.3	4.4	3.1

k

- Learn user and item factors
- Factors represent each user and item
- Specify *k* (hyperparameter)

Matrix factorization optimization metrics

THE GOAL OF MATRIX FACTORIZATION IS TO MINIMIZE THE ROOT MEAN SQUARED ERROR (RMSE)

$$\underset{p,q}{minimize} \sum_{(u,i)\epsilon S} (r_{ui} - \langle p_u, q_i \rangle)^2$$

p = User embedding

q = Item embedding

u = User index

i = Item index

S = Set of all ratings

r_{ui} = Rating in user − item matrix at u, i

Multitask recommender models

MODERN ALGORITHM THAT SIMULTANEOUSLY OPTIMIZES MULTIPLE OBJECTIVES DURING MODEL TRAINING

Use multiple types of user interactions to train a better recommender

The following 2 datasets are trained together:

1. Ratings explicit feedback; rating = 0.5 (worst) through 5.0 (best)
2. Watch event implicit feedback; watched = True or False

Variables are shared between the 2 tasks; allows for transfer learning

Note: We will demo this in a bit

Why build recommender models on AWS?



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Amazon has deep recommender experience

OVER 20 YEARS OF EXPERIENCE IN PERSONALIZATION AND RECOMMENDATION TECHNIQUES



Two Decades of Recommender Systems at Amazon.com

Amazon is well-known for personalization and recommendations, which help customers discover items they might otherwise not have found. In this update to our original article, we discuss some of the changes as Amazon has grown.

For two decades now,[1] Amazon.com has been building a store for recommendations, as well as desirable properties such as updating immediately

Source: Smith and Linden (2017), *IEEE Internet Computing*

Amazon item-to-item collaborative filtering

ITEM-TO-ITEM COLLABORATIVE FILTERING MADE POPULAR IN THE EARLY 2000s

Industry Report



Amazon.com Recommendations

Item-to-Item Collaborative Filtering

Greg Linden, Brent Smith, and Jeremy York • *Amazon.com*

Recommendation algorithms are best known for their use on e-commerce Web sites,[1] where they use input about a customer's interests to generate a list of recommended items. Many applications use only the items that customers purchase and explicitly rate to represent their interests, but they can also use other attributes, including items viewed, demographic data, subject interests, and favorite artists.

At Amazon.com, we use recommendation algorithms to personalize the online store for each customer. The store radically changes based on customer interests, showing programming titles to a software engineer and baby toys to a new mother. The click-through and conversion rates — two important measures of Web-based and email advertising effectiveness — vastly exceed those of untargeted content such as banner advertisements and top-seller lists.

There are three common approaches to solving the recommendation problem: traditional collaborative filtering, cluster models, and search-based methods. Here, we compare these methods with our algorithm, which we call *item-to-item collaborative filtering*. Unlike traditional collaborative filtering, our algorithm's online computation scales independently of the number of customers and number of items in the product catalog. Our algorithm produces recommendations in realtime, scales to massive data sets, and generates high-quality recommendations.

Recommendation Algorithms

Most recommendation algorithms start by finding a set of customers whose purchased and rated items overlap the user's purchased and rated items.[2] The algorithm aggregates items from these similar customers, eliminates items the user has

Source: Linden, Smith, and York (2003), *IEEE Internet Computing*

History of recommenders at Amazon

FIRST PERSONALIZED RECOMMENDATION EXPERIENCE AT AMAZON.COM USING COLLABORATIVE FILTERING

Early investment in personalization made shopping easy and fun

Feature launched and patent filed in 1998

Early features such as “Customers who bought this also bought that . . . ”



Which options to train and tune recommender models with AWS?



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS machine learning services for personalization

AMAZON PROVIDES CUSTOMERS WITH CHOICES FOR IMPLEMENTING PERSONALIZATION SOLUTIONS



Amazon Personalize

- Makes it quick and easy for developers and builders to implement AI and improve business outcomes
- No ML experience required



Amazon SageMaker

- Build, train, and deploy your own ML models quickly using a visual IDE
- All tools required for ML in a single service
- Automated machine learning (AutoML) plus prebuilt models, algorithms, and solutions
- Fully managed for scalability, performance, and cost-effectiveness

Options to train and tune model with SageMaker

CUSTOMERS HAVE MANY OPTIONS TO BUILD RECOMMENDERS WITH AMAZON SAGEMAKER

1. SageMaker Studio IDE
   Fully managed notebooks for one-time model training

2. SageMaker built-in algorithms
   Easy-to-use and scalable model training and tuning

3. SageMaker processing jobs
   Serverless Apache Spark processing and training clusters

4. SageMaker training and hyperparameter tuning jobs
   Custom training and hyperparameter tuning with any framework

1

SageMaker Studio
Fully managed notebooks for one-time model training

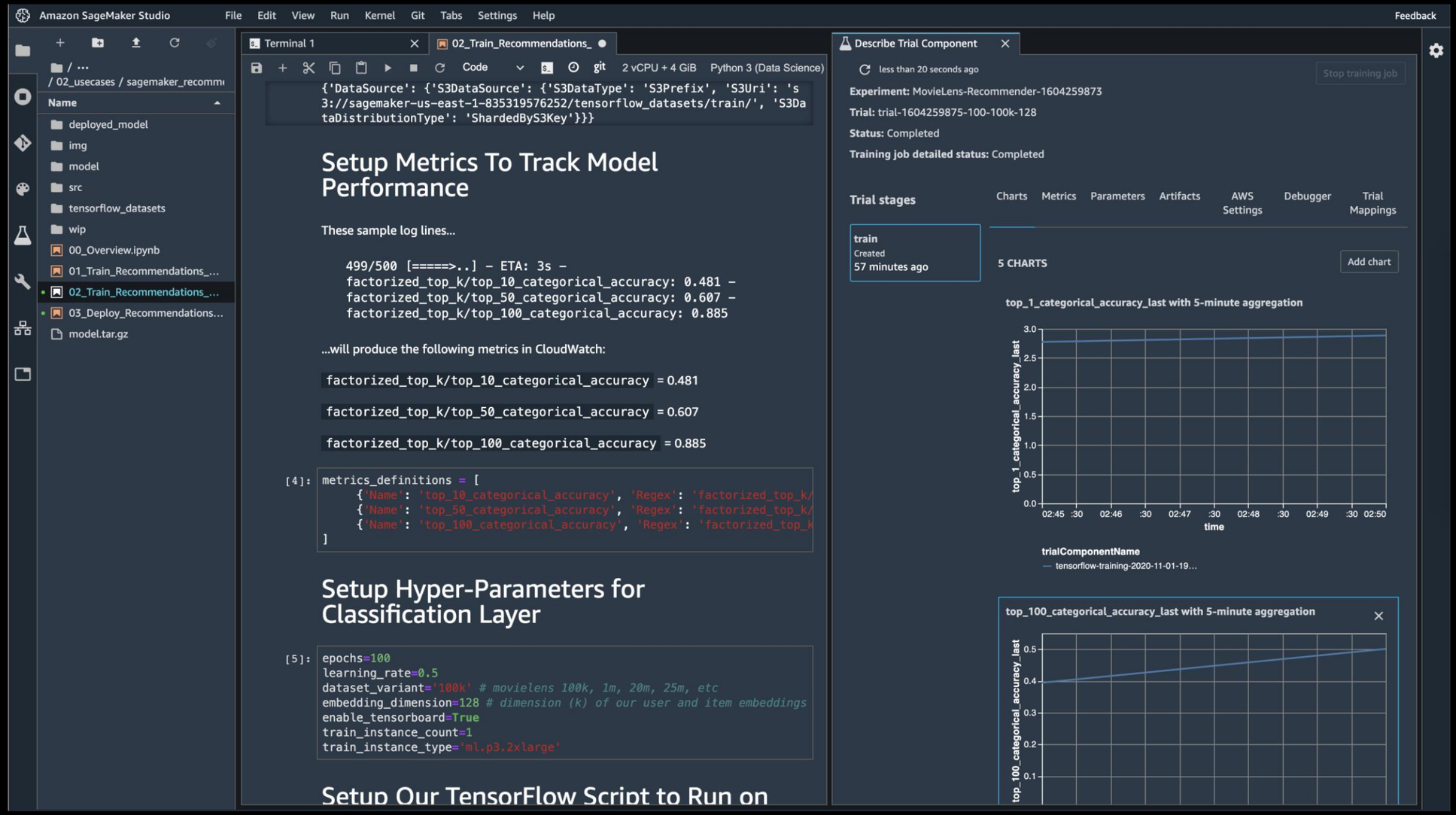


AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

SageMaker Studio IDE

IMMEDIATELY SEE THE IMPACT OF MODEL CHANGES DIRECTLY IN YOUR BROWSER



2

SageMaker built-in algorithms
Easy-to-use and scalable model training



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Built-in algorithms

SAGEMAKER OFFERS OUT-OF-THE-BOX MATRIX FACTORIZATION USING FACTORIZATION MACHINES

Train on many instances

```
In [12]: fm = sagemaker.estimator.Estimator(containers[boto3.Session().region_name],
                                            get_execution_role(),
                                            train_instance_count=10,
                                            train_instance_type='ml.c4.xlarge',
                                            output_path=output_prefix,
                                            sagemaker_session=sagemaker.Session())

         fm.set_hyperparameters(feature_dim=nbFeatures,
                                predictor_type='binary_classifier',
                                mini_batch_size=1000,
                                num_factors=64,
                                _speedometer_period=10,
                                epochs=100)

         fm.fit({'train': train_data, 'test': test_data})
```

Tune your hyperparameters

```
In [12]: fm = sagemaker.estimator.Estimator(containers[boto3.Session().region_name],
                                            get_execution_role(),
                                            train_instance_count=10,
                                            train_instance_type='ml.c4.xlarge',
                                            output_path=output_prefix,
                                            sagemaker_session=sagemaker.Session())

         fm.set_hyperparameters(feature_dim=nbFeatures,
                                predictor_type='binary_classifier',
                                mini_batch_size=1000,
                                num_factors=64,
                                _speedometer_period=10,
                                epochs=100)

         fm.fit({'train': train_data, 'test': test_data})
```

3

SageMaker processing jobs
Serverless Apache Spark processing and training clusters



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Serverless Apache Spark processing and training jobs

CUSTOM APACHE SPARK JOBS INCLUDING ALTERNATING LEAST SQUARES (ALS) MATRIX FACTORIZATION

train_spark_als.ipynb (notebook)

```
[ ]: from sagemaker.spark.processing import PySparkProcessor

processor = PySparkProcessor(base_job_name='spark-als',
                             role=role,
                             instance_count=10,
                             instance_type='ml.r5.2xlarge',
                             max_runtime_in_seconds=1200)
```

```
[ ]: from sagemaker.processing import ProcessingOutput

processor.run(submit_app='train_spark_als.py',
              arguments=['s3_input_data', s3_input_data,
                         's3_output_data', s3_output_data,
              ],
              logs=True,
              wait=False
)
```

train_spark_als.py (Python script)

```
lines = spark.read.text(s3_input_data).rdd
parts = lines.map(lambda row: row.value.split("::"))
ratingsRDD = parts.map(lambda p: Row(userId=int(p[0]),
                                     movieId=int(p[1]),
                                     rating=float(p[2]),
                                     timestamp=int(p[3])))
ratings = spark.createDataFrame(ratingsRDD)
(training, test) = ratings.randomSplit([0.8, 0.2])

# Build the recommendation model using ALS on the training data
als = ALS(maxIter=5,
          regParam=0.01,
          userCol="userId",
          itemCol="movieId",
          ratingCol="rating",
          coldStartStrategy="drop")
model = als.fit(training)

# Evaluate the model by computing the RMSE on the test data
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse",
                                labelCol="rating",
                                predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("\nrmse: " + str(rmse))
```

4

SageMaker training jobs
Custom training and hyperparameter tuning with any framework



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Train a custom recommender with TensorFlow

CUSTOM TENSORFLOW SCRIPT FOR MATRIX FACTORIZATION WITH USER AND MOVIE EMBEDDINGS

train.ipynb (notebook)

```
[ ]: from sagemaker.tensorflow import TensorFlow

estimator = TensorFlow(entry_point='train.py',
                       source_dir='src',
                       role=role,
                       instance_count=train_instance_count,
                       instance_type=train_instance_type,
                       py_version='py37',
                       framework_version='2.3.0',
                       hyperparameters={
                           'epochs': epochs,
                           'learning_rate': learning_rate,
                           'dataset_variant': dataset_variant,
                           'embedding_dimension': embedding_dimension,
                           'enable_tensorboard': enable_tensorboard
                       },
                       metric_definitions=metrics_definitions,
                       debugger_hook_config=False
            )
```

train.py (Python script)

```
# Create a custom Keras model with the user embeddings, movie embeddings
model = MovieLensModel(user_embeddings, movie_embeddings, task)

# Compile the model and prepare for training
model.compile(optimizer=optimizer)

# Train the model
model.fit(ratings.batch(4096), epochs=epochs, callbacks=callbacks)

# Make some sample predictions to test our model
# Note:  This is required to save and server our model with TensorFlow Serving
#        See https://github.com/tensorflow/tensorflow/issues/31057 for more  details.
index = tfrs.layers.factorized_top_k.BruteForce(query_model=model.user_embeddings)
index.index(movies.batch(100).map(model.movie_embeddings), movies)
```

Tune the TensorFlow recommender model

OPTIMIZE HYPERPARAMETERS WITH SAGEMAKER HYPERPARAMETER OPTIMIZATION (HPO)

optimize.ipynb (notebook)

```
[ ]: from sagemaker.tuner import IntegerParameter
     from sagemaker.tuner import ContinuousParameter
     from sagemaker.tuner import CategoricalParameter
     from sagemaker.tuner import HyperparameterTuner

     hyperparameter_ranges = {
         'epochs': CategoricalParameter([1, 2]),
         'learning_rate': ContinuousParameter(0.00001, 0.00005, scaling_type='Linear'),
         'embedding_dimension': CategoricalParameter([32, 64]),
     }
```

```
[ ]: objective_metric_name = 'root_mean_squared_error'

     tuner = HyperparameterTuner(
         estimator=estimator,
         objective_type='Maximize',
         objective_metric_name=objective_metric_name,
         hyperparameter_ranges=hyperparameter_ranges,
         metric_definitions=metrics_definitions,
         max_jobs=2,
         max_parallel_jobs=2,
         strategy='Random',
         early_stopping_type='Off'
     )
```

```
[ ]: tuner.fit(inputs={'train': s3_input_train_data,
                    },
                    include_cls_metadata=False,
                    wait=False)
```

Chrome File Edit View History Bookmarks People Tab Window Help
100% Mon Nov 30 9:44 AM
JupyterLab
d-dsxoghy6ztwy.studio.us-east-1.sagemaker.aws/jupyter/default/lab?
File Edit View Run Kernel Git Tabs Settings Help
Feedback

/ sagemaker_recommendations / src /

Name	Last Modified
test_data	a month ago
inference.py	a month ago
requirements.txt	a month ago
test-local.sh	a month ago
train_multitask.py	7 hours ago
train_spark_als.py	12 minutes ago
train.py	7 hours ago

Training a TensorFlow Multitask Recommender in this SageMaker Notebook (AdHoc)

Multi-task Recommender

In the basic retrieval tutorial we built a retrieval system using movie watches as positive interaction signals.

In many applications, however, there are multiple rich sources of feedback to draw upon. For example, an e-commerce site may record user visits to product pages (abundant, but relatively low signal), image clicks, adding to cart, and, finally, purchases. It may even record post-purchase signals such as reviews and returns.

Integrating all these different forms of feedback is critical to building systems that users love to use, and that do not optimize for any one metric at the expense of overall performance.

In addition, building a joint model for multiple tasks may produce better results than building a number of task-specific models. This is especially true where some data is abundant (for example, clicks), and some data is sparse (purchases, returns, manual reviews). In those scenarios, a joint model may be able to use representations learned from the abundant task to improve its predictions on the sparse task via a phenomenon known as transfer learning. For example, this paper shows that a model predicting explicit user ratings from sparse user surveys can be substantially improved by adding an auxiliary task that uses abundant click log data.

In this tutorial, we are going to build a multi-objective recommender for Movielens, using both implicit (movie watches) and explicit signals (ratings).

Imports

[1]:
```
!pip install -q tensorflow==2.3.0
!pip install -q tensorflow_recommenders==0.2.0
```

[2]:
```
import os
import pprint
import tempfile

from typing import Dict, Text

import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
```

[3]:
```
import tensorflow_recommenders as tfrs
```

Preparing the dataset

We're going to use the Movielens 100K dataset.

[4]:
```
ratings = tfds.load('movielens/100k-ratings',
                    download=False,
```

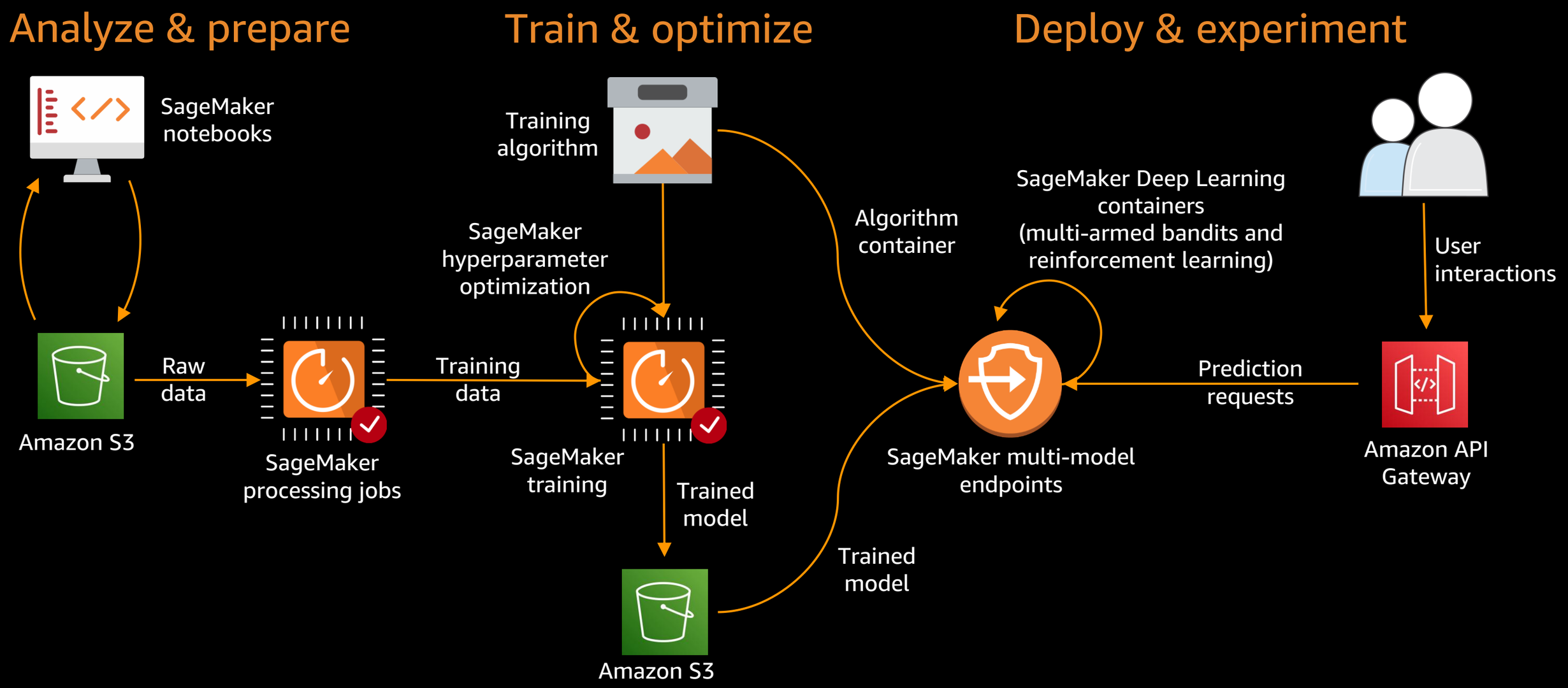
How can I scale to a million recommender models?



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Scale to a million models with end-to-end pipelines

TRAIN AND DEPLOY A RECOMMENDER MODEL SPECIFIC TO EACH OF YOUR MILLION CUSTOMERS



Summary



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Summary

Amazon has 20 years of personalization and recommendation experience

AWS offers many options for a fully personalized user experience

SageMaker supports custom training and tuning with any AI / ML framework

Users choose the best option based on their environment and skillset

Scale to a million recommender models with end-to-end ML pipelines

More information

AWS Blog

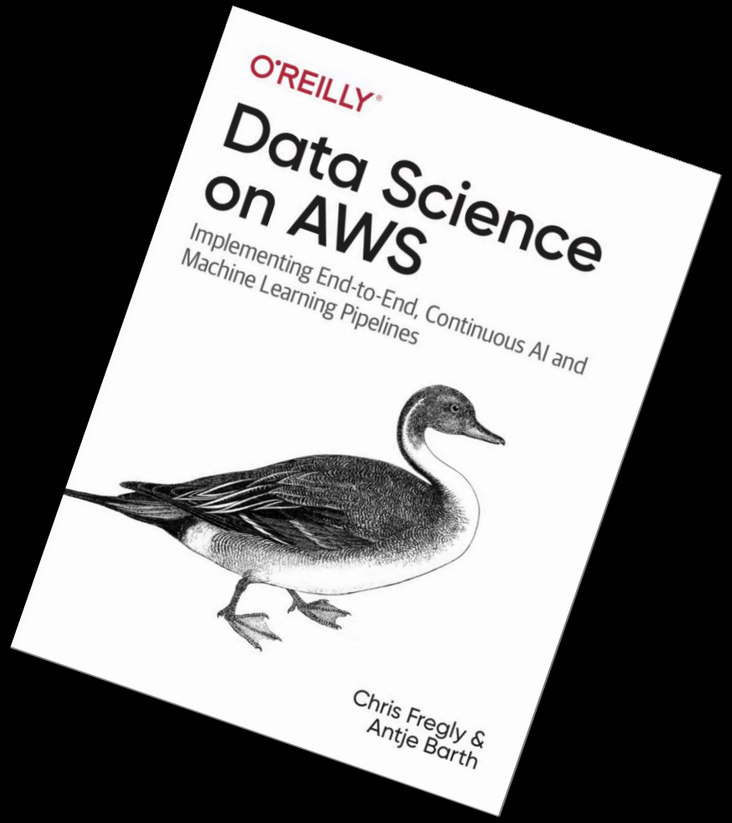
https://aws.amazon.com/blogs/machine-learning/real-time-music-recommendations-for-new-users-with-amazon-sagemaker/

Samples

https://github.com/data-science-on-aws/workshop

Book

https://datascienceonaws.com



Thank you!



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.