

# Summary Report (Include Cloud Service Set Up)

## Building Movie Recommendation System Using Multiple Models

### Introduction

Recommendation systems are one of the most widespread machine learning applications in the industry. They are not only built for movies and music streaming, but also for multiple products and services like e-commerce business, news and so on. Companies like Netflix, Youtube, and Amazon have leveraged recommendation systems to discover user preference and provide them with more relevant goods and services to enhance user experience and generate recurring revenue. No doubt, recommendation system is an indispensable part of their business success.

This assignment went through processes of building necessary elements for recommender engine, implementing an ensemble of machine learning algorithms and making use of AWS Sagemaker endpoint. The final pipeline is capable of offering specific movie suggestions for different users based on a combination result of three different models: ALS, AWS FM and Scikit-Surprise.

### Dataset

The dataset used in this assignment is the famous MovieLens dataset. There are various versions available online( <https://grouplens.org/datasets/movielens/> ), but for ease of use, ml-100k version is chosen to apply to those three models mentioned above. The dataset contains 100k ratings from 943 users on 1682 items. Features included in ml-100k version dataset are rating, genre, tag, movieid, userid, tagid and so on.

### Model 1 : ALS Estimator Using Spark

The movielens dataset is downloaded from an grouplens online URL. After loading and unzip the movielens dataset from grouplens website, two dataframes (with features including userId, movieId, and rating) representing training and test data are created for further use.

According to the Spark ALS documentation, ALS( Alternating Least Squares ) estimates the rating matrix as the product of two matrix with lower rank. During each iteration, one of the matrix is held constant while the other is solved using least squares. Then, the solved one is held constant while solving the other matrix using least squares.

To train an ALS estimator, initially we can simply use default parameters or set certain values, at the same time, specifying user, rating and item column respectively. For cold start strategy, users can choose between “drop” and "NaN". In order for us to derive a performance metric to evaluate the recommendation system, we set the coldStartStrategy parameter as “drop” to drop rows in the prediction dataframe which contains NaN values. Further, cross validation with parameter tuning is applied to enhance ALS model. We set a parameter grid and apply the cross validation method, finally the best parameter with the smallest rmse is

derived. When implementing the ALS model with the best parameter, the RMSE computed on the test data is 0.957.

The prediction result of the first model ALS estimator might not be satisfactory, but this is only from a single ALS model. We can expect further improvement later on when aggregating multiple effective models altogether.

Using als built-in functions `recommendForAllUsers` and `recommendForAllItems`, movies recommended and predicted rating for each user and user recommended for each movie can be easily obtained. The top 5 recommendation movieid are first stored in a list under the feature "recommendations". For further comparison with other methods, the format of the final recommendation dataframe is adjusted, containing three features: `userId`, `movieId` and `predicted_rating`.

## **Model 2 : Scikit-Surprise**

Surprise is a python Scikit used for recommendation system and it stands for Simple Python Recommendation System Engine. Users can use built-in datasets like movielens and custom dataset to build a recommendation system. There are various ready-to-use algorithms in Surprise package including collaborative filtering, matrix decomposition and so on. This package is a quite simple and easy to use. However, Surprise doesn't support content-based information and implicit ratings.

The process of building recommendation system using Surprise is quite straightforward. After importing relevant modules and loading ml-100k dataset, trainset and testset are specified respectively for further use. Next, SVD algorithm, which also deals with Matrix factorization, is trained on the trainset and tested on the testset. The rmse of SVD model on the test set is 0.675, which is much smaller than that of ALS estimator. Next, a function called `get_top_n` is defined to return the top n (default number is five) recommendations for each user. The top 5 movieid recommendations for all users can be obtained using for loop. Again, just like the transformation shown in previous als recommendation result, the recommendations result using Surprise is adjusted to the same format.

## **Model 3 : FM SageMaker**

The final model built for this recommendation system is Factorization Machines(FM), the famous built-in algorithm of Amazon SageMaker, which is widely used for recommendation system (eg, FM models are used by Netflix to recommend movies for users). Factorization Machines is one class of collaborative filtering algorithms. As the name suggests, FM also uses matrix factorization to reduce problem dimensionality and thus, greatly boost computational efficiency on large sparse dataset.

In the movielens dataset (and also in real world practice !), the number of users and items are often large whereas users normally rate a small portion of all movies available. Therefore, the actual number of recommendations is quite small, resulting a large sparse dataset. The basic idea of factorization in FM model is that a sparse rating matrix can be decomposed into a dense user matrix and item matrix with lower dimension. Another benefit of using FM is that

matrix factorization can also help us fill the blank values in the rating matrix, which means we can recommend new items to users.

The pipeline of the recommendation system using sagemaker fm involves loading ml-100k data, data preprocessing stage (one hot-encoding user and movie for sparse matrix, and building binary label), converting data to protobuf and write to S3, and training, hyperparameter tuning, deploying and finally predict using AWS SageMaker endpoint. The detailed code for building the pipeline is covered in the notebook. Here, a few steps to set up the cloud service are shown below.

The first step is to create a notebook instance and specify the instance name and type. For example, for the notebook instance named derecommendation, the instance type is ml.t2.medium and the volume size is 5GB. Next we can open jupyter and create a 'conda\_python 3' notebook to write the code and analysis.

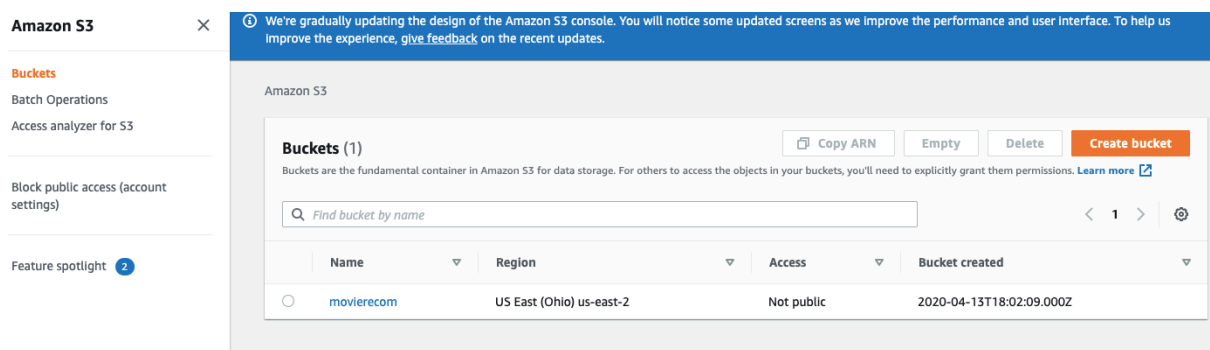
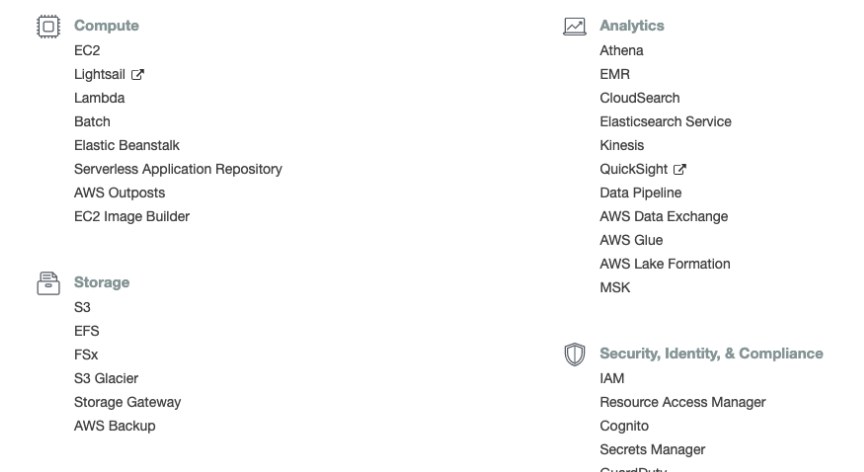
The image shows two screenshots from the Amazon SageMaker console. The top screenshot displays the 'Notebook instances' page, which lists two instances: 'DE-movie-rec' and 'derecommendation'. Both are of type 'ml.t2.medium' and are in the 'InService' status. The bottom screenshot shows the 'Notebook instance settings' for the 'derecommendation' instance, detailing its configuration.

Name	Instance	Creation time	Status	Actions
DE-movie-rec	ml.t2.medium	Apr 20, 2020 20:17 UTC	InService	<a href="#">Open Jupyter</a>   <a href="#">Open JupyterLab</a>
derecommendation	ml.t2.medium	Apr 12, 2020 14:50 UTC	InService	<a href="#">Open Jupyter</a>   <a href="#">Open JupyterLab</a>

Notebook instance settings		
Name	Status	Notebook Instance type
derecommendation	InService	ml.t2.medium
ARN	Creation time	Elastic Inference
arn:aws:sagemaker:us-east-2:210178500767:notebook-instance/derecommendation	Apr 12, 2020 14:50 UTC	-
Lifecycle configuration	Last updated	Volume Size
-	Apr 12, 2020 14:53 UTC	5GB EBS

Another important step is to store the data in protobuf format to S3. To achieve this, first we need to create a bucket in Sagemaker service-Storage-S3 interface. Remember to notedown the bucket name as we need to specify the data storage bucket in the code.



## Create bucket

### General configuration

Bucket name

Bucket name must be unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)

Region

US East (Ohio) us-east-2

### Bucket settings for Block Public Access

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

☒ **Block all public access**

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

```
#specify my personal bucket name
bucket = 'movierecom'
prefix = 'fm'

#write the key and prefix for train, test and output
train_key      = 'train.protobuf'
train_prefix   = '{}/{}'.format(prefix, 'train')

test_key       = 'test.protobuf'
test_prefix    = '{}/{}'.format(prefix, 'test')

output_prefix  = 's3://{}/output'.format(bucket, prefix)
```

Training data S3 path: s3://movierecom/fm/train/train.protobuf  
Test data S3 path: s3://movierecom/fm/test/test.protobuf  
FM model output S3 path: s3://movierecom/fm/output

We can also use SageMaker to tune hyperparameters. After fitting the fm model using an initial set of hyperparameters, it is also quite convenient to do hyperparameter tuning jobs in SageMaker training-hyperparameter tuning jobs interface. We need to create a tuning job and set configurations as shown in the screenshots. Note that the objective metric here is `binary_classification_accuracy` and the goal is to maximize it.

🔍 Notebook

Notebook instances

Lifecycle configurations

Git repositories

🔍 Training

Algorithms

Training jobs

Hyperparameter tuning jobs

🔍 Inference

Compilation jobs

Model packages

Models

Endpoint configurations

Endpoints

Batch transform jobs

Hyperparameter tuning jobs

🔄 Add/Edit tags

Create hyperparameter tuning job

🔍 Search hyperparameter tuning jobs

< 1 > ⚙️

	Name	Status	Training completed/total	Creation time	Duration
<input type="radio"/>	movie-tuning	🟢 Completed	1 / 1	Apr 27, 2020 14:38 UTC	12 minutes
<input type="radio"/>	movie-rec-tuning	🔴 Failed	0 / 1 1 Failed	Apr 27, 2020 14:27 UTC	3 minutes
<input type="radio"/>	movielens-tuning	🔴 Failed	0 / 1 1 Failed	Apr 27, 2020 14:17 UTC	4 minutes

Algorithm options

Use an Amazon SageMaker built-in algorithm, your own algorithm, or a third-party algorithm from AWS Marketplace.

▼ Algorithm source

☒ Amazon SageMaker built-in algorithm [Learn more](#)

☐ Your own algorithm resource

☐ Your own algorithm container in ECR [Learn more](#)

☐ An algorithm subscription from AWS Marketplace

▼ Choose an algorithm

Factorization Machines

Container

The registry path where the training image is stored in Amazon ECR. [Learn more](#)

404615174143.dkr.ecr.us-east-2.amazonaws.com/factorization-machines:1

Input mode

You can provide your training data as a file or pipe.

File

☐ Enable SageMaker metrics time series

### Objective metric

To find the best training job, set an objective metric and tuning type. See the hyperparameter tuning job detail page for a summary of the best training job.

Objective metric

Type

test:binary\_classification\_accuracy

maximize

### Hyperparameter configuration

To find the best training job, choose ranges of hyperparameters that hyperparameter tuning searches. To set a fixed value for a hyperparameter, set the type to Static. We've set default hyperparameter ranges for the algorithm you've chosen. [Learn more](#)

Name	Type	Scaling type	Value / Range
feature_dim	Static	-	2625
mini_batch_size	Integer	Linear	50 - 3000
epochs	Integer	Linear	50 - 500
num_factors	Static	-	64
predictor_type	Static	-	binary_classifier

movie-tuning-001-83011120

Clone

Create model package

Stop

Create model

#### Job settings

Job name	Status	SageMaker metrics time series	IAM role ARN
movie-tuning-001-83011120	Completed <a href="#">View history</a>	Disabled	arn:aws:iam::210178300767:role/service-role/AmazonSageMaker-ExecutionRole-20200412T155021
ARN	Creation time	Training time (seconds)	
arn:aws:sagemaker:us-east-2:210178300767:training-job/movie-tuning-001-83011120	Apr 27, 2020 14:38 UTC	538	
Last modified time	Billable time (seconds)		
Apr 27, 2020 14:49 UTC	538		
	Managed spot training savings		
	0%		
	Tuning job source/parent		
	movie-tuning		

The best parameter which gives us the highest binary classification accuracy is shown below. Therefore it is optimal to set the hyperparameter as follows, deploying and predicting based on this model.

Hyperparameters	
Key	Value
_tuning_objective_metric	test:binary_classification_accuracy
bias_init_method	normal
epochs	403
factors_init_method	normal
feature_dim	2625
linear_init_method	normal
mini_batch_size	359
num_factors	64
predictor_type	binary_classifier

Finishing the prediction using sagemaker endpoint, we can test on a particular record and a set of records. It is easy to get on-demand response for a particular userId and movieId pair, and compare the predicted scores and labels among different pairs. However, we need to derive the top n recommendations for each user, not just a single line response. In the notebook, three functions named ‘GetRecIndex’, ‘GetRecMovieID’, ‘UserFmDf’ to derive the final recommendations for each user (see the code in the notebook). The basic idea is to get all the response for each user and sort the positive response(predicted label=1) by score,

and map the result to movieId in the test dataset to get the corresponding movieid for each response.

## **Model Combination : Scoring Strategy**

Now the movie recommendations generated by the each model and corresponding predicted score/rating are available. We need to figure out a way to weight the result from different models and derive final top 5 movie recommendations for each user based on a combination score of those three models : ALS, Sagemaker FM, Surprise.

The scoring strategy takes both accuracy of each model and predicted score of each pair into consideration. To ensure the predicted scores of each pair in different models are comparable, standardization process is implemented since we only care about the relative distance of each predicted score in a model. Next, a weight metric is constructed using predicted score(after scaling) divided by the rmse of each model. The higher the weight is, the more accurate the recommendation result will be. Then, we add the weight of each pair(userid, movieid) generated by different models. By sorting the sum of weight of different pairs, we can easily get the top 5 recommendations for each user.

## Reference:

- [1]  
<https://spark.apache.org/docs/latest/api/java/org/apache/spark/ml/recommendation/ALS.html>
- [2]  
<https://surprise.readthedocs.io/en/stable/FAQ.html>
- [3]  
<https://aws.amazon.com/blogs/machine-learning/build-a-movie-recommender-with-factorization-machines-on-amazon-sagemaker/>