BU.330.740 Large Scale Computing on the Cloud

**Lab 3. AWS SageMaker – Python Code Instructions**

1. **Download and Unzip Dataset**

Download the MovieLens 100k dataset and unzips it. This dataset is used for building a recommendation system.

!wget http://files.grouplens.org/datasets/movielens/ml-100k.zip

!unzip -o ml-100k.zip

1. **Preprocess the Dataset**

 Change the working directory to the extracted dataset folder.

 Shuffle the ua.base file (training data) to create a randomized version, ua.base.shuffled.

 Display the first 10 rows of the shuffled training data.

%cd ml-100k

!shuf ua.base -o ua.base.shuffled

!head -10 ua.base.shuffled

1. **Import Libraries**

Load necessary libraries for using Amazon SageMaker, handling data preprocessing, and working with sparse matrices.

import sagemaker

import sagemaker.amazon.common as smac

from sagemaker import get\_execution\_role

from sagemaker.deserializers import JSONDeserializer

import boto3, csv, io, json

import numpy as np

from scipy.sparse import lil\_matrix

1. **Define Dataset Parameters**

Specify:

* Number of users (nbUsers) and movies (nbMovies).
* Total features (nbFeatures), which is the sum of users and movies.
* Number of training samples (nbRatingsTrain) and test samples (nbRatingsTest).

nbUsers=943

nbMovies=1682

nbFeatures=nbUsers+nbMovies

nbRatingsTrain=90570

nbRatingsTest=9430

1. **Load Dataset and Create Features**

Define a function to read the dataset file using a sparse matrix (X), where each row represents one user-movie interaction. This consists of 943 one-hot encoded features for the user ID and 1682 one-hot encoded features for the movie ID. Labels (Y) are binary (like or don’t like): 4-star and 5-star ratings are set to 1, while lower ratings are set to 0. Note that the [FM implementation](https://docs.aws.amazon.com/sagemaker/latest/dg/fact-machines.html" \t "_blank) in Amazon SageMaker requires training and test data to be stored in *float32* tensors in [protobuf](https://developers.google.com/protocol-buffers/" \t "_blank) format.

def loadDataset(filename, lines, columns):

# Features are one-hot encoded in a sparse matrix

X = lil\_matrix((lines, columns)).astype('float32')

# Labels are stored in a vector

Y = []

line=0

with open(filename,'r') as f:

samples=csv.reader(f,delimiter='\t')

for userId,movieId,rating,timestamp in samples:

X[line,int(userId)-1] = 1

X[line,int(nbUsers)+int(movieId)-1] = 1

if int(rating) >= 4:

Y.append(1)

else:

Y.append(0)

line=line+1

Y=np.array(Y).astype('float32')

return X,Y

1. **Generate Training and Testing Datasets**

Load the training and test datasets and build the following data structures:

* A training sparse matrix: 90,570 lines and 2,625 columns
* A training label array: 90,570 ratings
* A test sparse matrix: 9,430 lines and 2,625 columns
* A test label array: 9,430 ratings

X\_train, Y\_train = loadDataset('ua.base.shuffled', nbRatingsTrain, nbFeatures)

X\_test, Y\_test = loadDataset('ua.test',nbRatingsTest,nbFeatures)

1. **Validate Dataset Shapes**

Ensure that the dimensions of the training dataset match the expected size.

print(X\_train.shape)

print(Y\_train.shape)

assert X\_train.shape == (nbRatingsTrain, nbFeatures)

assert Y\_train.shape == (nbRatingsTrain, )

zero\_labels = np.count\_nonzero(Y\_train)

print("Training labels: %d zeros, %d ones" % (zero\_labels, nbRatingsTrain-zero\_labels))

print(X\_test.shape)

print(Y\_test.shape)

assert X\_test.shape == (nbRatingsTest, nbFeatures)

assert Y\_test.shape == (nbRatingsTest, )

zero\_labels = np.count\_nonzero(Y\_test)

print("Test labels: %d zeros, %d ones" % (zero\_labels, nbRatingsTest-zero\_labels))

1. **Configure S3 Paths**

Define the S3 bucket and file paths for uploading the dataset and storing output. Change the bucket to your S3 bucket name.

bucket = 'yourbucketname'

prefix = 'sagemaker/fm-movielens'

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)

1. **Upload Dataset to S3**

Convert datasets into Protobuf format and upload them to S3.

def writeDatasetToProtobuf(X, Y, bucket, prefix, key):

buf = io.BytesIO()

smac.write\_spmatrix\_to\_sparse\_tensor(buf, X, Y)

buf.seek(0)

obj = '{}/{}'.format(prefix, key)

boto3.resource('s3').Bucket(bucket).Object(obj).upload\_fileobj(buf)

return 's3://{}/{}'.format(bucket,obj)

train\_data = writeDatasetToProtobuf(X\_train, Y\_train, bucket, train\_prefix, train\_key)

test\_data = writeDatasetToProtobuf(X\_test, Y\_test, bucket, test\_prefix, test\_key)

print(train\_data)

print(test\_data)

print('Output: {}'.format(output\_prefix))

1. **Set SageMaker Containers**

Specify the container images for SageMaker in different regions to run the Factorization Machines model. We will choose the one that matches our region for model training.

containers = {'us-west-2': '174872318107.dkr.ecr.us-west-2.amazonaws.com/factorization-machines:latest',

'us-east-1': '382416733822.dkr.ecr.us-east-1.amazonaws.com/factorization-machines:latest',

'us-east-2': '404615174143.dkr.ecr.us-east-2.amazonaws.com/factorization-machines:latest',

'eu-west-1': '438346466558.dkr.ecr.eu-west-1.amazonaws.com/factorization-machines:latest'}

1. **Train the Model**

Configure the training instance and hyperparameters, then starts training using the uploaded datasets.

We will use 1000 as the mini-batch size, 64 factors, and 100 training epochs.

**This step takes a long time.**

fm = sagemaker.estimator.Estimator(containers[boto3.Session().region\_name],

get\_execution\_role(),

instance\_count=1,

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,

epochs=100)

fm.fit({'train': train\_data, 'test': test\_data})

1. **Deploy the Model**

Deploy the trained model as a HTTP endpoint for real-time inference. Please be patient, as this step takes a long time. **Do not rerun this step or refresh your webpage**. Just wait until it is completed. You can check the SageMaker Dashboard for your endpoint.

fm\_predictor = fm.deploy(instance\_type='ml.c4.xlarge', initial\_instance\_count=1)

1. **Make Predictions**

We’re now ready to invoke the model’s HTTP endpoint using *predict()*. The format for both request and response data are JSON, which requires us to provide a simple serializer to convert our sparse matrix samples to JSON.

We’re now able to classify any movie for any user. We will make predictions on user 1 in the test dataset <https://files.grouplens.org/datasets/movielens/ml-100k/ua.test>.

You can build a new data set, process it the same way as the training and test set, and use *predict()* to get results. You can also experiment with different prediction thresholds.

import json

from sagemaker.serializers import JSONSerializer

from sagemaker.deserializers import JSONDeserializer

class FMSerializer(JSONSerializer):

def serialize(self, data):

js = {"instances": []}

for row in data:

js["instances"].append({"features": row.tolist()})

return json.dumps(js)

fm\_predictor.serializer = FMSerializer()

fm\_predictor.deserializer = JSONDeserializer()

result = fm\_predictor.predict(X\_test[0:10].toarray(), initial\_args={"ContentType": "application/json"})

print(result)

print (Y\_test[0:10])

1. **Clean Up Resources**

Delete the SageMaker endpoint to avoid incurring additional costs. Please check the dashboard to make sure all endpoints are deleted. You can also deploy the model later.

sagemaker.Session().delete\_endpoint(fm\_predictor.endpoint\_name)

**Once everything is completed, please terminate your notebook instance. Ensure that nothing is running in your dashboard.**

A screenshot of a computer

AI-generated content may be incorrect.