Project Milestone 3

Joseph Patten, Md Mahedi Hasan, Md Muhtasim Billah, Scott Walters

1.

1(a). Problem Statement

The project for this class aims at analyzing the co-purchasing behavior of Amazon products. Making use of the available datasets in hand, a recommender system can be built that uses the collaborative filtering algorithm to predict items that a certain user might be interested in. A formal problem statement of the project consisting of the input, output and the ultimate goal of the proposed solution is provided below.

i) Problem input:

The dataset made available by Amazon will be used as the input for building the recommender system. Datasets were collected from an open source platform (https://nijianmo.github.io/amazon/index.html# code) which contains information about Amazon products as well as its users. Though, data are available for several categories, only a selective few will be used for the analysis. Moreover, the dataset contain multiple features for items and users but for collaborative filtering, only three features- the `userID`, `itemID` and `ratings` for the products will be necessary. So, a sample input dataset will mainl ycontain the above three features with hundreds of thousands of observations depending on the category.

ii) Problem output:

Using the input data, several outputs will be expected from the algorithm that can answer the following questions:

- Based on the product ratings previously given by a specific user, what are some similar products that can be recommeded to that user.
- Based on the ratings given by other users, what product can be recommended to a new user who is similar in characteristics to the other users?

iii) Ultimate goal:

The ultimate goal for this project is to build a reliable recommender system that implements collaborative filtering algorithm both in sequential and distributed platforms. Several candidate algorithms will be tested for experimentation to compare their accuracy.

1(b). Sequential Algorithms Description

There are multiple approaches for collaborative filtering using sequential algorithms. Primarily, these algorithms can be divided into two categories: memory based and model based. These two categories can again be divided into multiple subcategories. A brief explanation of how these algorithms work are provided below with codes.

I) **Memory based algorithms:**

Memory based collaborative filtering can be subdivided into item-based and user-based algorithms based on the concept of similarity.

a) Item-based: Item based algorithm stands on the idea of similarity among the items. It assumes that the a certain item can be recommended to a certain user if it is very similar to the items that this certain user previously liked.

b) User-based: User based algorithm stands on the idea of similarity among the users. It assumes that the a certain item can be recommended to a certain user if it is liked by other users who are very similar this certain user.

The similarity of the users or the items can be calculated using different similarity metrics such as the `Euclidean distance`, `MSD` the `cosine distance` (also called `Pearson correlation`). To realize how an algorithm is performing, several evaluation metrics can be used such as `rmse` and `mae`. The user-based algorithm is described below in steps.

- Filter df (denote `userSubset`) to only contain items that a user (given `userInput`) has rated.
- Group `userSubset` by `reviewerID`:
  - Iterate through each `reviewerID` group (each group contains the same items that the user rated from the `userInput`).
  - Calculate a score for "how similar" a reviewer is to the user from `userInput`.
- Now we have a series of similarity scores for each reviewer (when compared with user from `userInput`).
- Using unfiltered df, merge in calculated similarity scores for each reviewer.
- Reweight each reviewers' item ratings using similarity score multiplied by their rating for an item.
- Groupby each item, and sum the weighted rating and similarity score.
- We define the recommendation score (for each item) as being the overall summed weighted rating over the overall summed similarity score.
- Sort items (in descending order) by recommendation score.
- Give user (from `userInput`) top n items.

Statistical concept of `Centered KNN` can also be used for implementing the aforementioned memory based algorithms. For this project, a popular python library called `Surprise` that was specially developed for recommender systems will be used. Using this library, memory based algorithms can be implemented as below.

```
#load necessary libraries
import numpy as np # for linear algebra
import pandas as pd # for data processing
from surprise import Dataset, Reader, KNNWithMeans # for rec sys
#load data
mydata=pd.read_csv("file.csv",names = ['item','user','rating'])
#parse data for feeding into surprise function
reader = Reader(rating_scale=(1, 5))
#parsed dataframe
data = Dataset.load_from_df(mydata, reader)
#options for training colab filtering
sim_options = {
    "name": "cosine", # alternatives: eulidean, msd
    "user_based": True,  # alternative: False
}
#pass the options into centered KNN algorithm
algo = KNNWithMeans(sim_options=sim_options)
```

```
#build training set
trainingSet = data.build_full_trainset()
#fit the training set into the model
algo.fit(trainingSet)
#check the evaluation metric for model accuracy
print(algo.best_score["rmse"])
```

As seen from the above descriptions, that there might be several combinations of parameters that can be looked into to find the best one. This can be done using a grid search as below.

```
#load necessary libraries
from surprise.model_selection import GridSearchCV
#options for finding the best combination of parameters
sim_options2 = {
    "name": ["msd", "cosine", "euclidean"],
    "min_support": [3, 5, 7],
    "user_based": [True, False],
}
param_grid = {"sim_options": sim_options2}
#apply the grid search on the centerd-KNN algorithm
gs = GridSearchCV(KNNWithMeans, param_grid, measures=["rmse", "mae"], cv=5)
#fit the data into the grid search model
gs.fit(data)
#print the best score and the best parameter combo
print(gs.best_score["rmse"])
print(gs.best_params["rmse"])
```

II) **Matrix factorization algorithms:**

Another approach for collaborative filtering uses the concept of matrix factorization. This approach involves a step to reduce or compress the large but sparse user-item matrix and works specially well for large datasets. It can also be seen as breaking down a large matrix into a product of smaller ones. There are several algorithms that fall within this category such as the singular value decomposition or `SVD`, stochastic gradient descent or `SGD` and alternating least squares or `ALS`. These algorithms can be used within the python `Surprise` library as well. A grid search implementation for `SVD` is shown below.

```
from surprise import SVD
#parameter dict including #epochs, learning rate and regularization
param_grid = {
    "n_epochs": [5, 10],
    "lr_all": [0.01, 0.03],
    "reg_all": [0.3, 0.5]
}
#apply the grid search on the SVD algorithm
gs2 = GridSearchCV(SVD, param_grid, measures=["rmse", "mae"], cv=5)
#fit the data into the grid search model
gs2.fit(data)
#print the best score and the best parameter combo
print(gs2.best_score["rmse"])
print(gs2.best_params["rmse"])
```

2.

Collaborative Filtering in Apache Spark

Collaborative filtering can naturally be extended to a distributed platform. For this project, `Apache Spark` will be used to run the algorithm using its python API called `PySpark`. Utilizing the Spark dataframes or RDD (resilient distributed dataset), the loaded data is intuitively distributed across partitions in a cluster. `PySpark` will be loaded in the jupyter notebook and the rest of the procedures will be followed through the notebook. The python pseudocode for this is provided below along with brief description of the procedures.

Spark comes with its own modules for dealing with SQL commands as well as implementations for machine learning algorithms. PySpark's `mllib` includes the `ALS` algorithm for collaboraitve filtering which will be used for making recommendation system on the distributed platform. First, the required modules are loaded from the pyspark library and a spark session is initiated which launches the cluster. The gateway to the spark session is a spark context object which is automatically created for a Spark session.

```
# import necessary libraries
from pyspark import SparkContext
from pyspark.ml.recommendation import ALS
from pyspark.ml.feature import StringIndexer
from pyspark.sql import SparkSession, Row
from pyspark.sql.functions import col
from pyspark.sql.types import StructType, StructField, IntegerType, StringType
appName="CollabFilter_PySpark"
# initialize the spark session
spark = SparkSession.builder.appName(appName).getOrCreate()
# get sparkcontext from the sparksession
sc = spark.sparkContext
```

next, a schema is created for the dataset to be used which is passed into the spark dataframe while loading the data. It's worth mentioning here that both Spark dataframe and RDD objects can be deployed to the cluster and they can be converted into one another with simple function calls. However, we decide to proceed with the dataframe. However, the dataset we use contains the item ratings as strings and thus needs to changed into integer applying the `withColumn()` method. More importantly, for applying `ALS` for collaborative filtering, the `Spark` machine learning library expects the `user` and `item` values to be unique integers but our dataset has them as string type. To convert the `item` and `user` columns into integer format, the `StringIndexer()` function is called. The `ALS` algorithm will be applied on the indexed dataframe denoted.

```
#define schema
schema = StructType([
    StructField("item", StringType(), True),
    StructField("user", StringType(), True),
    StructField("rating", StringType(), True),
    StructField("timestamp", IntegerType(), True)])
#read the file as a dataframe
df = spark.read.csv("Gift_Cards.csv",header=False,schema=schema)
#convert rating colum from string to integer
df = df.withColumn("rating", df["rating"].cast(IntegerType()))
#provide index values for item and user to convert them into integers
stringIndexer = StringIndexer(inputCols=["item","user"], outputCols=["itemIndex","userIndex"])
#fit the model
model = stringIndexer.fit(df)
```

```
#transform the dataframe into an indexed dataframe
df_indexed = model.transform(df)
```

Before building the ALS model, the data is divided into training and testing set. Then the data is passed into the ALS model with a set of parameters as shown in the code below. Then the training data is fitted into the model and predictions can be made on the test dataset.

```
#split the data into training and testing set
(training, test) = df_indexed.randomSplit([0.8, 0.2])
#training the model
#define the model parameters
als = ALS(maxIter=5,
          implicitPrefs=False,
          userCol="userIndex",
          itemCol="itemIndex",
          ratingCol="rating",
          coldStartStrategy="drop")
#train the model
model = als.fit(training)
# predict using the testing datatset
predictions = model.transform(test)
#display the results
predictions.show()
```

Finally, a function can be defined that takes the user ID as input and shows the top 5 recommendations for this user based on the fitted collaborative filtering algorithm. This is shown in the code cell below.

```
def recommendedItems(userIndex,limit):
    test =  model.recommendForAllUsers(3)\
        .filter(col('userIndex')==userIndex)\
        .select(["recommendations.itemIndex","recommendations.rating"])\
        .collect()
    return test
# display top 5 recommended artists for user 9386
recommendedItems(9386,5)
```

The ALS model, implemented in Spark via PySpark can be efficiently used for building a pretty accurate recommender system as shown above. The codes have been deployed via a jupyter notebook in the AWS EC2 instance and could be run successfully. However, no necessity for optimization was found for this specific dataset and algorithm.

3.

Initial Experimental Plan

The experimentation plans for the project are descibed below in segements.

3(a). Dataset description

Since we are focusing purely on collaborative filtering we are able to move from the SQL tables (that we initially used to house our data) to the dataframes. Our process is downloading the datasets from https://nijianmo.github.io/amazon/index.html#code. Some of these datasets will be loaded as Spark RDD or dataframes for implementing the algorithm. This algorithm allows us to use `userID`, `itemID` and `rating` instead of the large set of metadata dealt with in Milestone 2. We are still dealing with a substantial amount of data with ratings on the dataset `Clothing, Shoes and Jewelry` housing `32 million` ratings.

3(b). Algorithms to Test

For the sequential algorithms, a comparative analysis among the memory-based and model-based aproaches will be presented. For memory based, both the user-based and item-based methods will be explored where user-based will be the baseline algorithm. For model-based approaches, SVD will be considered as the baseline algorithm. There are also some relevant parameters that need to be tuned for achieving the best model performance. The distance metrics and the evaluation metrics will also be tested for better performance.

For disributed platform, `Apache Spark` is being used which comes with its built in machine learning library. This library has the built in function for only the `ALS` algorithm and thus, till now, no other algorithms has been tested on the distributed platform. But, there's a plan to explore at least another algorithm in Spark before the milestone 4 for the project is due.

3(c). System Setting

An EC2 (Elastic Computing) instance provided by Amazon Web Service (AWS) has been used for running both the algorithms. The instance used was a linux operating system (Ubuntu 2020) that has 8GB memory and 300GB disk space which belong to the `r2a.xlarge` category of Amazon machine image (AMI). `Apache Spark` was installed in this machine for creating distributed platform using this one single machine which served as the master as well as the worker node. The version used was 2.4.7 with a compatible Hadoop version. A python API for spark called `PySpark` was used within a Jupyeter notebook to run the programs.

3(d). Experimentation Plan

i) Several sequential algorithms will be tested such as `user-based`, `item-based`, `SVD` and `SGD`. These algorithms' performance will be compared to the 'ALS algorithm implemented in the distributed platform. The model based algorithms are expected to perform better than the memory based approaches, specially for large datasets.

ii) For model accuracy, evaluation metrics such as mean absoulte error (`MAE`) and root mean squared error (`RMSE`) will be used where the latter one usually proves to be more reliable. In case of program runtime, no significant difference is expected since the same machine is being deployed as both the master and the worker node.

iii) The Amazon dataset has a different file for each category which has largely varying observations (in terms of number of ratings). And thus, some of the dataset are very small where some are very large. So, the data sizes span from a few MS up to >1GB. The data size will definitely influence the model performance which will also be investigated in this project.