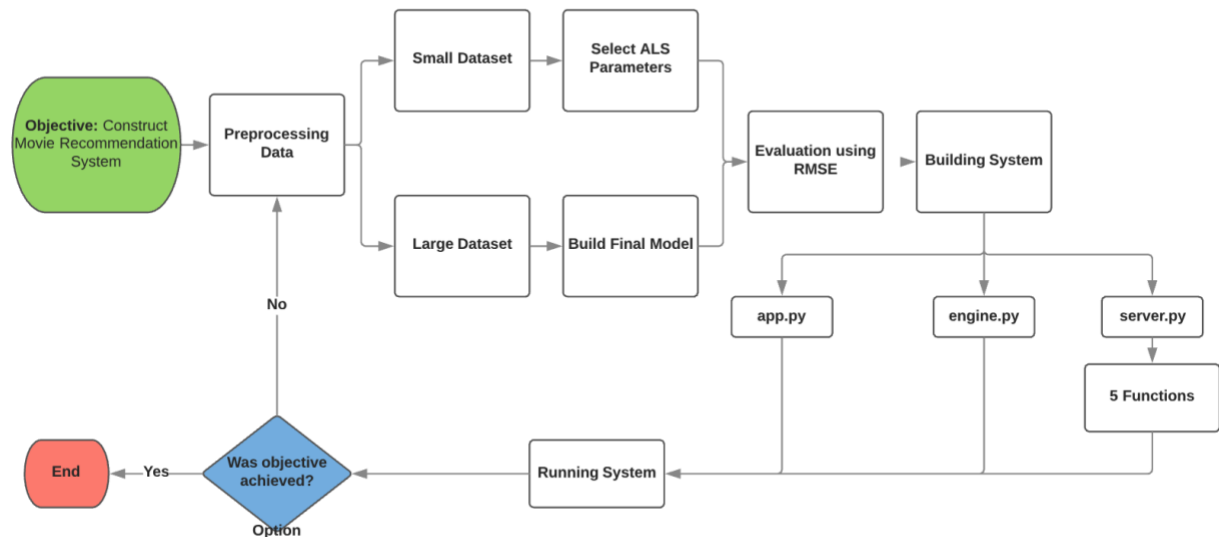


Movie Recommendation System Project

Group Member: Chang Liu, Yuzheng An, Taiming Cao, Sandeepan Datta, He Dong

Flow Chart



1. Problem Setting

During this project, we want to construct a movies recommendation system about the MovieLens dataset to have a clear look over all the ratings and movies in the dataset and do analysis about these movies with our recommendation system. Then we can recommend movies for user which they may be fond of based on their previous interest.

For example, we can take a look at the movies with the most number of ratings. And we could see the average rating for each movie and find the movies with the highest average rating. We could also build model to predict the movie title, movie rating given the movie IDs and user IDs for specific user.

The approach we used to accomplish this goal is to use collaborative filtering with Spark Alternative Least Square implementation. And the overall project is about two parts: backend and frontend. For the back end, firstly, we need to sparse the data from the Internet and then load data into Spark RDDs. Then we should build the recommender

with ALS algorithm. For the front end, we need to use flask to construct a web-service based on our Spark model in the back end.

2. Data Description

We firstly preprocessed our model data before we started to build an online movie recommender. Our preprocessing methods included loading and parsing dataset, and persisting the resulting RDD for later use. Building the recommender model using the complete dataset and persist the dataset for later use. In our project, we have found two datasets from GroupLens Research which has collected and made available rating datasets from the MovieLens web site, which is a non-commercial, personalized movie recommendation website. The data sets were collected over various periods of time, depending on the size of the set.


























The dataset includes small dataset and full dataset. The small dataset includes 100,000 ratings and 2,488 tag applications applied to 8,570 movies by 706 users. The full dataset includes 21,000,000 ratings and 470,000 tag applications applied to 27,000 movies by 230,000 users. The format of files in these datasets is uniform and simple, which makes it easier for us to use python function “split()” to parse the lines after they loaded into RDDs. After parsing the movies and rating files, we generate two RDDs:

- For each line in the ratings dataset, we create a tuple of (UserID, MovieID, Rating). We dropped the timestamp because we do not need it for this recommender.
- For each line in the movies dataset, we create a tuple of (MovieID, Title). We dropped the genres because we do not use them for this recommender.

In order to compute our model results efficiently, we were using the small dataset to test our models and used the large dataset for final recommendations.

3. Algorithms

The major algorithm we used is collaborative filtering to predict the interest of the user by collecting information from the users. The algorithm works this way if X user has the same taste for movie like Y user. Then X and Y are likely to prefer the same movie. The image show

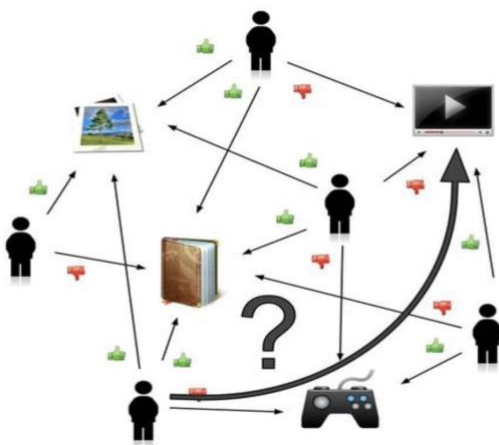
the example of collaborative filtering. The system will predict the recommendation for movie based on the existing rating of other users with the similar taste for the movies.

The implementation in MLlib has the following parameters:

- numBlocks
- rank
- iterations
- lambda
- implicitPrefs
- alpha

Collaborative filtering is using alternating least squares (ALS), For a user's-products-rating dataset, ALS will build an $m \times n$ matrix for user*product, m is number of users, n is number of products. In this dataset, not all users have rated all products, so this matrix

is usually sparse, user i 's rating for product j can be blank. What ALS does is to fill all the blanks for this sparse matrix, so that we can get all users' ratings for all products, the rating that is filled by ALS is the predicted rating.



To get the lowest RMSE score of our model. We split the data set into training data and testing data. And adjust our parameters to get the best parameters for our models. Firstly we should build our model based on the small data set, since it will take much less time. Then to build

the recommender model, we need to use the complete model to train the model. After we get the trained model, we can use it to obtain top movies recommendations for the users. For a new user, we assign the ID to be 0, get some new ratings and then add the new information to the data set. At last we need to train the model for the new data set to get the recommendations.

4. Evaluation of our recommendation:

We are trying to evaluate the recommendation ALS models through the method of comparing their RMSE. We consider the ALS model with the lowest RMSE would be the best model for our predicted outcomes.

In the first part of the process of evaluating, we used small dataset to determine the best ALS parameters from the model with the lowest RMSE score. We firstly got the the best rank with lowest RMSE score on the validation dataset (20%) after applying training dataset (60%) on it, then we test the selected model on testing dataset (60%) and evaluated through the RMSE score of our model on the small dataset. Our best model on the testing dataset has a RMSE of 0.958.

```
model = ALS.train(training_RDD, best_rank, iterations=iterations,
                  lambda_=regularization_parameter)
predictions = model.predictAll(test_for_predict_RDD).map(lambda r: ((r[0], r[1]), r[2]))
rates_and_preds = test_RDD.map(lambda r: ((int(r[0]), int(r[1])), float(r[2]))).join(predictions)
error = math.sqrt(rates_and_preds.map(lambda r: (r[1][0] - r[1][1])**2).mean())
```

```
print ('For testing data the RMSE is %s' % (error))
```

For testing data the RMSE is 0.9580864192129078

In the process of evaluating our recommender models on the complete dataset, we used similar method on the small dataset. We trained the recommender model using training dataset (70%) and tested on our testing dataset (30%), from which we got RMSE at nearly 0.822. We can see that we got a more accurate recommender when we using a larger dataset.

```
test_for_predict_RDD = test_RDD.map(lambda x: (x[0], x[1]))

predictions = complete_model.predictAll(test_for_predict_RDD).map(lambda r: ((r[0], r[1]), r[2]))
rates_and_preds = test_RDD.map(lambda r: ((int(r[0]), int(r[1])), float(r[2]))).join(predictions)
error = math.sqrt(rates_and_preds.map(lambda r: (r[1][0] - r[1][1])**2).mean())
```

```
print 'For testing data the RMSE is %s' % (error)
```

For testing data the RMSE is 0.82183583368

5. Architecture

Our recommendation system include three important components, engine.py, app.py, and server.py. We use server.py to start up our system, use app.py to apply our 5 main applications and use engine.py as our tools box.

5.1 Back end: consists of engine.py file

1. Engine.py a back end py file to define function which are going to be used for the front end.

Functions:

- get_counts_averages: We input the movie IDs and their ratings, and return the average ratings for each movies.
- count_and_average_rating: This function will update the movies ratings from the current data and get average ratings.
- train_model: It will train the ALS model with the current data.
- predict_rating: Input user IDs and movie IDs and make predictions. Then output the movie titles, movie ratings and number of ratings.

- `add_rating`: This function will add additional movie ratings in the format of (user IDs, movie IDs and ratings.)
- `get_ratings_for_movie_ids`: Input a user ID and a movie IDs and predict ratings for them.
- `get_top_rating`: Recommends up to 'movies_count' top unrated movies to each user

5.2 Front end: consists of two Python files:

1. `app.py` - a Flask web application that defines a RESTful-like application around the engine. We initialize Flask while calling a function `create_app`. The Recommendation Engine object is created and then we associate the `@main.route` annotations. A route, that is its URL and may contain parameters between `<>`. There are four of these annotations defined, that correspond with the four Recommendation Engine methods.

Functions:

- `GET /<user_id>/ratings/top/<int:count>` get top recommendations from the engine.
- `GET /<user_id>/ratings/<movieid>` get predicted rating for a individual movie for a particular user
- `GET /ratings/top_ave/<int:count>` get the top average ratings for movies with rating count greater than 25
- `GET/ratings/top/<int:count>` get the most popular movies by specifying the rating count

2. `server.py`- Initializes a *CherryPy* web server after creating a Spark context and Flask web app. It is also easy to run multiple HTTP servers (e.g. on multiple ports) at once. All this makes it a perfect candidate for an easy to deploy production web server for our online recommendation service. We use the `__main__` entry point, we do three things:

Functions:

- Create a spark context as defined in the function `init_spark_context`, passing additional Python modules there.
- Create the Flask app calling the `create_app` we defined in `app.py`.
- Run the server itself.

6. Result/Output

6.1 How to start?

1. Start server in terminal or command line
2. Change directory to the system documents directory

3. Run the system in local mode

`spark-submit --master local[1] --conf spark.driver.memory=1g server.py`

```
INFO:engine:ALS model built!
[13/May/2018:14:55:09] ENGINE Bus STARTING
INFO:cherrypy.error:[13/May/2018:14:55:09] ENGINE Bus STARTING
[13/May/2018:14:55:09] ENGINE Started monitor thread 'Autoreloader'.
INFO:cherrypy.error:[13/May/2018:14:55:09] ENGINE Started monitor thread 'Autoreloader'.
[13/May/2018:14:55:09] ENGINE Serving on http://0.0.0.0:5432
INFO:cherrypy.error:[13/May/2018:14:55:09] ENGINE Serving on http://0.0.0.0:5432
[13/May/2018:14:55:09] ENGINE Bus STARTED
INFO:cherrypy.error:[13/May/2018:14:55:09] ENGINE Bus STARTED
```

6.2 Posting new ratings:

Curl is used to post new ratings from the shell. The below command will give ratings as a list of list: [curl --data-binary @user_ratings.file http://localhost:5432/0/ratings](http://localhost:5432/0/ratings)

```
18/05/12 17:27:33 INFO DAGScheduler: Job 23 finished: first at MatrixFactorizationModel.scala:67, took 0.016089 s
INFO:engine:ALS model built!
127.0.0.1 - - [12/May/2018:17:27:22 -0500] "POST /0/ratings HTTP/1.1" 200 34 "-" "curl/7.59.0"
```

```
260, 5
1, 4
16, 3.6
25, 4
32, 4.5|
335, 2
379, 1
296, 3.5
858, 5
50, 4
```

6.3 Getting recommendations:

1. To get the top 10 user recommendations for the new user(0) based on ratings of other movies already posted by the new user (as defined above), we can type in the web browser the following command: <http://localhost:5432/0/ratings/top/10>

← → ↻ localhost:5432/0/ratings/top/10

Top 10 Movies for User 0

Movie Name	Expected Rating	Popular
"Lives of Others	4.860975367163149	37
Modern Times (1936)	4.746077710527332	32
"Godfather	4.7196226923280635	201
Citizen Kane (1941)	4.678109211755507	85
Lawrence of Arabia (1962)	4.570118397406325	51
"Lord of the Rings: The Fellowship of the Ring	4.568751831915058	200
"Third Man	4.56027843586665	38
"Lord of the Rings: The Return of the King	4.523978741210939	176
Cool Hand Luke (1967)	4.517064956741879	46
"Godfather: Part II	4.496775306019216	135

2. We get the movie name, expected rating and the count of the the no of ratings (Popular column) for the new user with user id '0'; We can also use the same format to recommend movies for an old user. For example, if we wanted to recommend top 3 movies to user 8, we can type in the webbrowser the following command: <http://localhost:5432/8/ratings/top/3>

← → ↻ localhost:5432/8/ratings/top/3

Top 3 Movies for User 8

Movie Name	Expected Rating	Popular
Pulp Fiction (1994)	4.4937990824699865	325
"Shawshank Redemption	4.461830878703093	311
Fight Club (1999)	4.44266691708666	202

3. To get predicted rating for a individual movie for a particular user, we can type in the webbrowser the following command: <http://localhost:5432/8/ratings/3>

← → ↻ localhost:5432/8/ratings/3

Movie 3 predicted rating for User 8

Movie Name	Expected Rating	Popular
Grumpier Old Men (1995)	2.7839041080128033	59

From the output we can see that the movie 'Grumpier Old Men'(movie id 3) has a predicted rating of 2.78 for user 8

4. To get the top average ratings for movies with rating count greater than 25, we can type in the webbrowser the following command: http://localhost:5432/ratings/top_ave/8

← → localhost:5432/ratings/top_ave/8

8 highest rated movies (more than 25 ratings)

Movie Name	Rating
"Godfather	4.490049751243781
"Shawshank Redemption	4.487138263665595
On the Waterfront (1954)	4.448275862068965
All About Eve (1950)	4.434210526315789
Ran (1985)	4.423076923076923
"African Queen	4.42
Roger & Me (1989)	4.392857142857143
"Maltese Falcon	4.387096774193548

The top 8 highest rated movies with more than 25 ratings are given above

5. To get the most popular movies by specifying the rating count, we can type in the webbrowser the following command: <http://localhost:5432/ratings/top/5>

← → localhost:5432/ratings/top/5

5 most popular movies

Movie Name	Total Rating Received
Forrest Gump (1994)	341
Pulp Fiction (1994)	325
"Shawshank Redemption	311
"Silence of the Lambs	304
Star Wars: Episode IV - A New Hope (1977)	292

The above gives the 5 most popular movies by specifying the rating count

7. Citation Source

J. (2017, April 20). Jadianes/spark-movie-lens. Retrieved May 12, 2018, from <https://github.com/jadianes/spark-movie-lens>