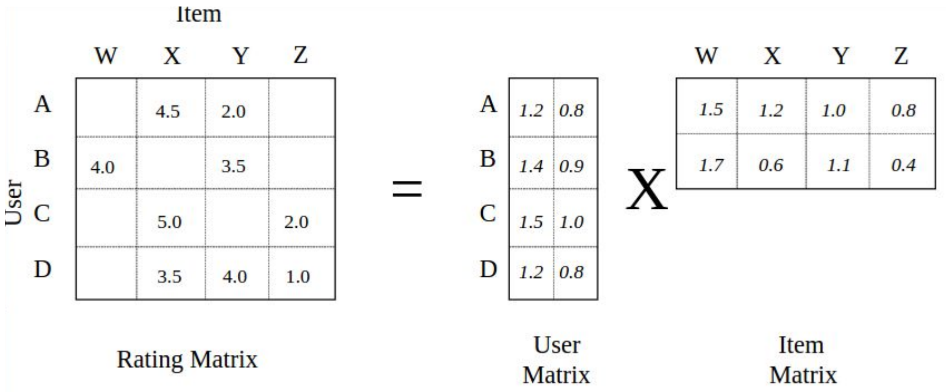
Double Latent Factor Learning based Recommender

Group GL

Explicit Recommender

The input of a recommender systems is a rating matrix of user-item with missing entries.

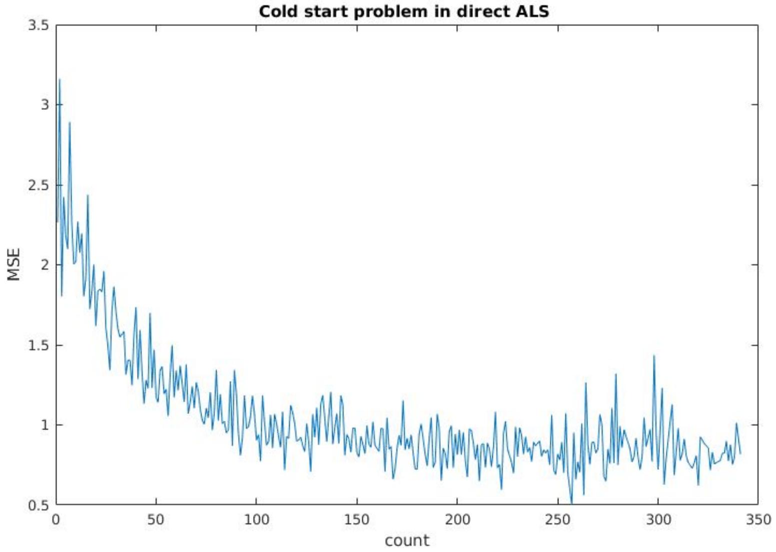
Latent Factor Based Algorithm for Recommendation System **aims to fill in the missing entries** of a user-item association matrix, by decomposing the RatingMatrix to UserMatrix * ItemMatrix.



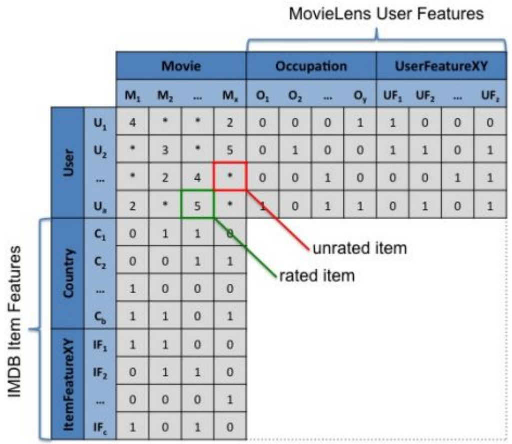
Sparsity Problem

The percentage of missing entries reflects the sparsity of input data. In reality, most of the matrix entries are blank. Sparsity is high.

Cold Start Problem:the lack of data for new users and items that limit the recommendations for that user as well as performance decreases for sparse data.



Related Work



There are mainly two categories of recommending methods:

- Content based
  - products clustering with machine learning algorithms
  - products classification with machine learning algorithms
- Collaborative filtering:
  - Matrix Factorization: SGD (Stochastic Gradient descent), Alternating Least Squares (ALS)
  - Nearest Neighbor: varies similarity measure function, select ratings of product j for Top N similar users to user i to predict the rating for user i to product j.

For sparsity problem

- Hybrid content based and collaborative filtering
  - One paper provided a method that directly concatenate tag-movie to user-movie and use ALS

Our work

Character: Double ALS to solve sparsity problem

We implement our **first direct user-movie ALS** in Spark

We implement our **second indirect user-tag ALS**:

> Preprocess to transform RDD(user,**movie**,rating) into RDD(user,**tag**,rating) using the information of (movie,tags). A rating record of movie i with n tags would be mapped to n records of RDD(user,tag,rating).
>
> Use RDD(user,tag,rating) to train a ALS model to predict the rating of a **user to a tag**.
>
> we sum up user A’s prediction for all the tags of a movie B, and **take the average as the final predicted rating** for user A to movie B.

We switch between the two result of ALS based algorithm using count(movie i):

Used Algorithm ALS

- Matrix Factorization decompose your large user-item matrix into lower dimensional user feature matrix P and item feature matrix Q.

- **Estimate the user rating** (or in general preference) by multiplying those factors according to the following equation:

$$r'_{ui} = p_u^T q_i \ (1)$$

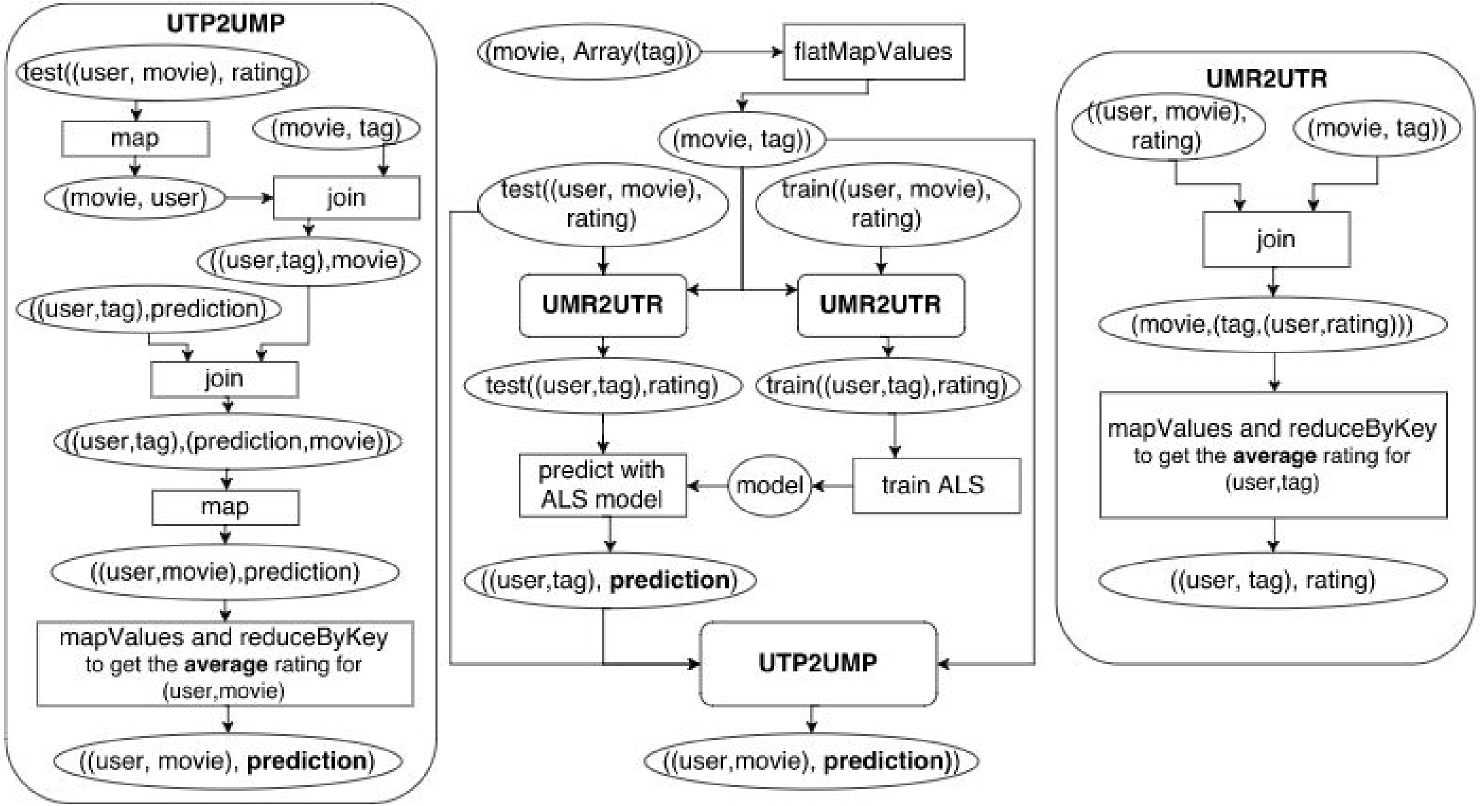
- **In order to learn those factors** (i.e. P, Q) you need to minimize the following **quadratic loss function**:

$$min_{q,p} \sum_{u,i} (r_{ui} - p_u^T q_i)^2 \ (2)$$

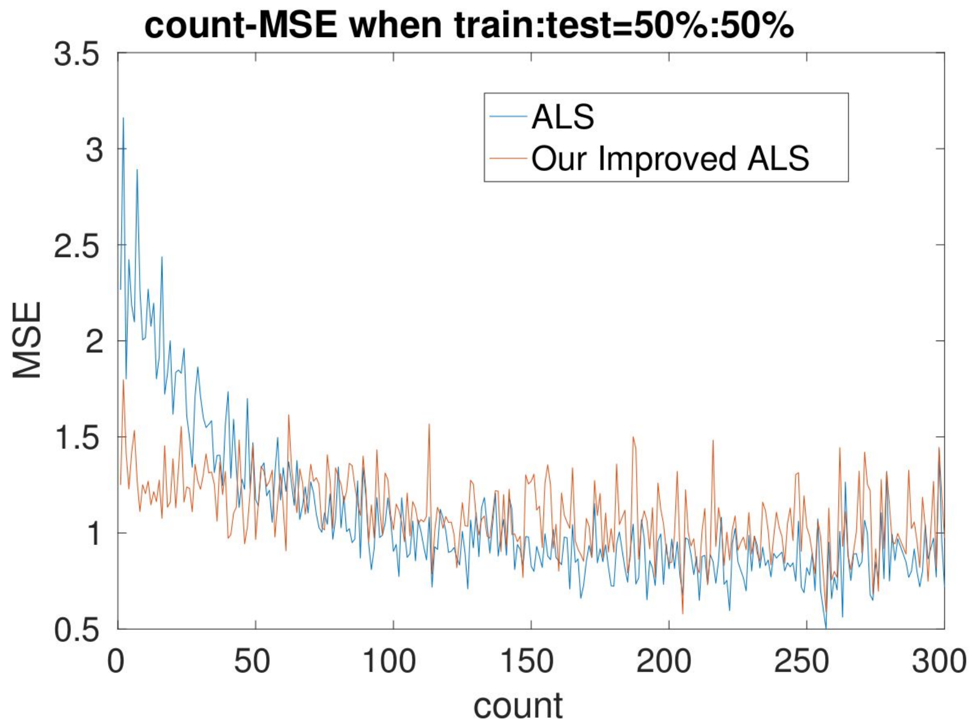
In an **SGD (Stochastic Gradient descent)** approach, after calculating the error for each training (user,item,rating) pair. You update the parameters by a factor in the opposite direction of the gradient.

Alternating Least Squares (ALS) represents a different approach to optimizing the loss function. The **key insight** is that you can turn the non-convex optimization problem in Equation (2) into an "easy" quadratic problem if you fix either pu or qi. ALS fixes each one of those alternatively. When one is fixed, the other one is computed, and vice versa.

Implementation of Indirect ALS - RDD Graph



Result-Local



For count<50, our algorithm has better result. => cold start problem gets improved

Similar result for new user

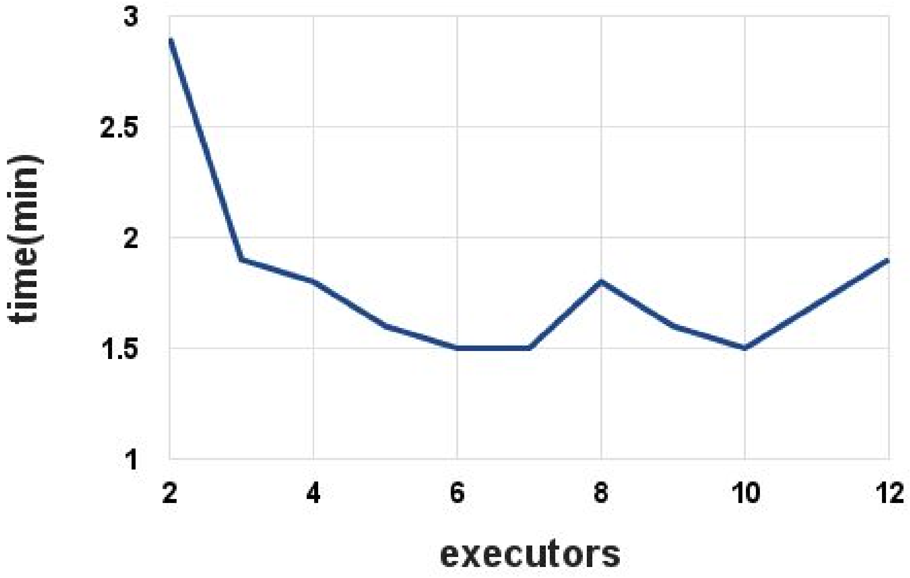
As (train:test) get lower (i.e. training matrix gets more sparse), the cross points for the two gets

```
modelUM Mean Squared Error = 0.918513552758948
modelUT Mean Squared Error = 0.706517505741195
modelUM2 Mean Squared Error = 1.06918692869097
```

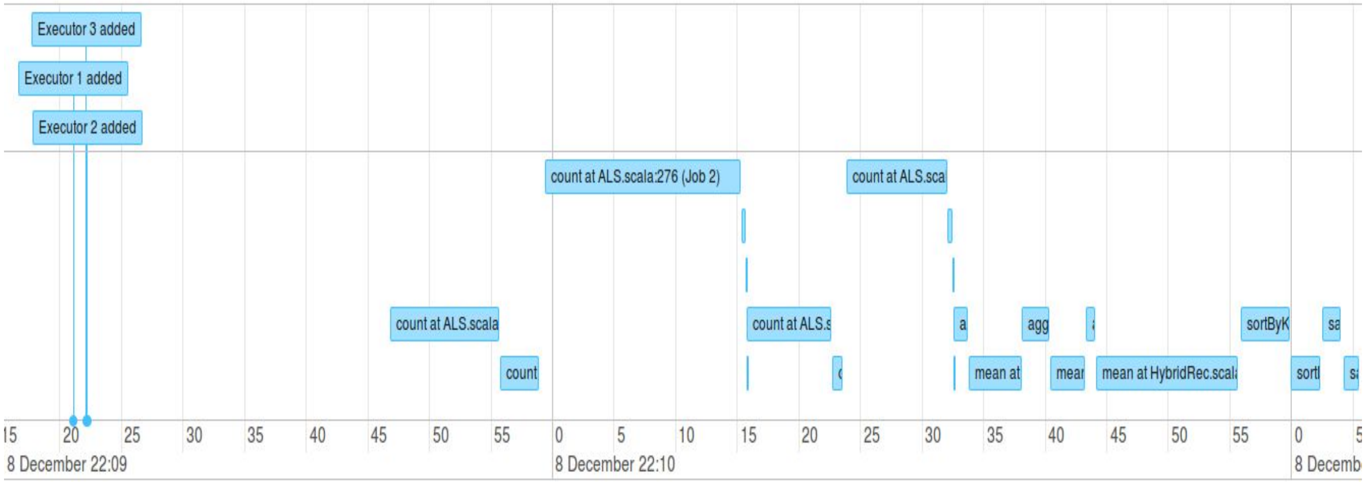
Result-Deploy to AWS

The executor’s maximum is 7, So the most effective performances reaches peaks at 8 instances.

Communication Cost
Algorithm Cost



Future Works



- Scala up the test data size
- Too much ‘count’ function
- Come up with a formula to calculate the switch point, in form of Sparsity(‘count’ of input matrix).
- Include timestamp in ALS model.

Questions?