Movie Recommendation System

14BIT029- Kishan Mistri
Guided by: Prof. Sapan Mankad

❖ Flow of presentation

- Problem statement
- Solutions
- Implementation details
  - Technological overview
  - Problem’s solution code snippets
- Timeline
- Previous milestones

Project objective:

Estimate a ‘**utility function’** that ‘**automatically predicts’** efficiently how user will ‘**like’** an item(i.e Movie in this context)

Based on:

- Past behavior
- Item similarity
- Context
- Many more...

Solutions

Content based Filtering

Filter based on features of Item or User to get relatable recommendations

Collaborative Filtering

Recommend based on similarities between users or items

Probabilistic Approach

This type of models yields a single solution describing the outcome of problem from appropriate inputs given.

Implementation

Django web framework

Positives:

- Swift Deployment
- Admin Panel
- Backend handling support
- Lightning fast
- Convenient Scalable

Negatives:

- Real-time web apps

PostgreSQL

Positives:

- Open Source
- Production type database handling
- GUI based design tool is also available
- Backend hashing
- Impressive database specifications*

Negatives:

- Learning curve

Datasets specification

1: MovieLens:

- ➔ It is a data set that provides 1,00,00,054 user ratings on movies.
- ➔ 95,580 tags applied to 10,681 movies by 71,567 users.
- ➔ Users of MovieLens were selected randomly.
- ➔ All users rated at least 20 movies.
- ➔ Each user represented by a unique id.

2. Netflix:

- ➔ 4,80,189 users
- ➔ 17,770 movies
- ➔ 100 M+ ratings

Code-Snippets - 1

SVD : MovieLens dataset had sufficient amount of rating for each movie.

- GridSearch
- Holdout Cross -Validation
- fit
- RMSE

```
In [28]: print('Grid Search...')
    ...: param_grid = {'n_epochs': [5, 10], 'lr_all': [0.002, 0.005]}
    ...: grid_search = GridSearchCV(SVD, param_grid, measures=['rmse'], cv=3)
    ...: grid_search.fit(data)
Grid Search...

In [29]: algo = grid_search.best_estimator['rmse']

In [30]: trainset = data.build_full_trainset()
    ...: algo.fit(trainset)
Out[30]: <surprise.prediction_algorithms.matrix_factorization.SVD at 0x2ac2e869320>

In [31]: predictions = algo.test(trainset.build_testset())
    ...: print('Biased accuracy on A,', end='   ')
    ...: accuracy.rmse(predictions)
Biased accuracy on A,   RMSE: 0.8355
Out[31]: 0.83547819579582971

In [32]: testset = data.construct_testset(B_raw_ratings)  # testset is now the set B
    ...: predictions = algo.test(testset)
    ...: print('Unbiased accuracy on B,', end=' ')
    ...: accuracy.rmse(predictions)
Unbiased accuracy on B, RMSE: 0.9546
Out[32]: 0.95455967117350726
```

Code-Snippets - 2

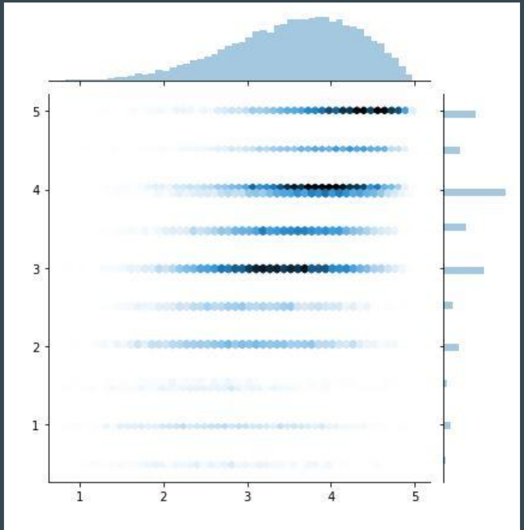
SVDpp :

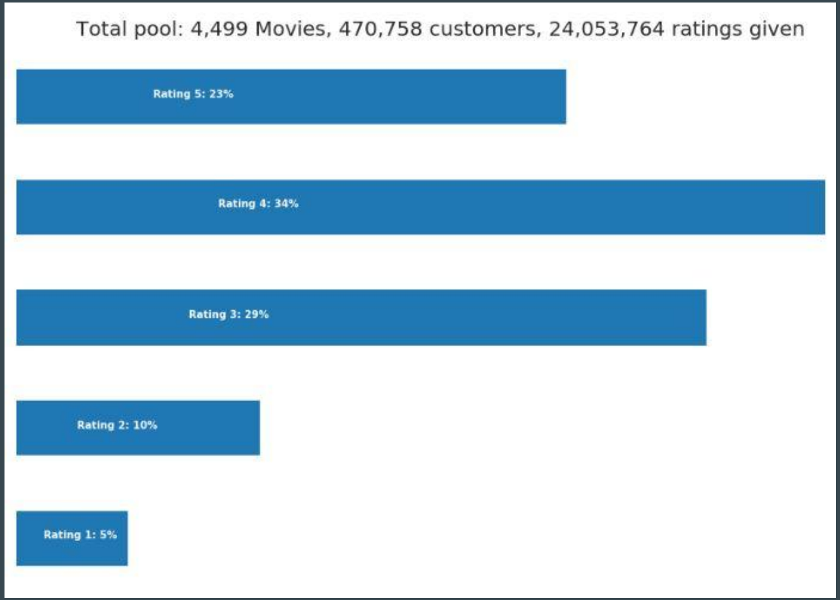
- Difference between SVD & SVD++ (SVDpp)
- Surprise library exploration

```
In [7]: print('Grid Search...')
   ...: param_grid = {'n_epochs': [5, 10], 'lr_all': [0.002, 0.005]}
   ...: grid_search = GridSearchCV(SVDpp, param_grid, measures=['rmse'], cv=3)
Grid Search...

In [8]: grid_search.fit(data)

In [9]: algo = grid_search.best_estimator['rmse']

In [10]: trainset = data.build_full_trainset()
    ...: algo.fit(trainset)
Out[10]: <surprise.prediction_algorithms.matrix_factorization.SVDpp at 0x1c44ca884e0>

In [11]: predictions = algo.test(trainset.build_testset())
    ...: print('Biased accuracy on A,', end='   ')
    ...: accuracy.rmse(predictions)
Biased accuracy on A,   RMSE: 0.8900
Out[11]: 0.8899890939119427

In [12]: testset = data.construct_testset(B_raw_ratings)  # testset is now the set B
    ...: predictions = algo.test(testset)
    ...: print('Unbiased accuracy on B,', end=' ')
    ...: accuracy.rmse(predictions)
Unbiased accuracy on B, RMSE: 0.9356
Out[12]: 0.9355797104150447З
```

Something is not right here

Own Collaborative Model Code- Snippet - 3





- Pearson Correlation Similarity

```
In [14]: def pearson(s1,s2):
    ...:     s1_c = s1-s1.mean()
    ...:     s2_c = s2-s2.mean()
    ...:     return np.sum(s1_c*s2_c)/np.sqrt(np.sum(s1_c**2)*np.sum(s2_c**2))
```

Code- Snippet - 4

```
In [15]: def get_recs(movie_name,m,num):
    ...:     import numpy as np
    ...:     reviews=[]
    ...:     for title in m.columns:
    ...:         if title == movie_name:
    ...:             continue
    ...:         cor=pearson(m[movie_name],m[title])
    ...:         if np.isnan(cor):
    ...:             continue
    ...:         else:
    ...:             reviews.append((title,cor))
    ...:     reviews.sort(key=lambda tup:tup[1],reverse= True)
    ...:     return reviews[:num]
```

```
In [17]: print(pearson(m['Money Train (1995)'],m['Taxi Driver (1976)']))
-0.0566821657822

In [18]: print( get_recs('Taxi Driver (1976)' ,m, 10))
__main__:4: RuntimeWarning: invalid value encountered in double_scalars
[('Show Me Love (Fucking Åmål) (1998)', 0.37986794529603091), ('We Bought a Zoo (2011)', 0.35644896714848273), ('Children of Paradise (Les enfants du paradis) (1945)',
0.34401050606613581), ('Everything Is Illuminated (2005)', 0.33489136001624376), ('Full Metal Jacket (1987)', 0.33305467566773672), ('Control (Kontroll) (2003)',
0.32799160772726799), ('24: Redemption (2008)', 0.32341688427604121), ('Saw V (2008)', 0.29977949808031873), ('Keeping Mum (2005)', 0.29977949808031851), ('Moneyball (2011)',
0.28722215417774705)]

In [19]: %timeit get_recs('Taxi Driver (1976)' ,m, 10)
__main__:4: RuntimeWarning: invalid value encountered in double_scalars
43.9 s ± 7.35 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

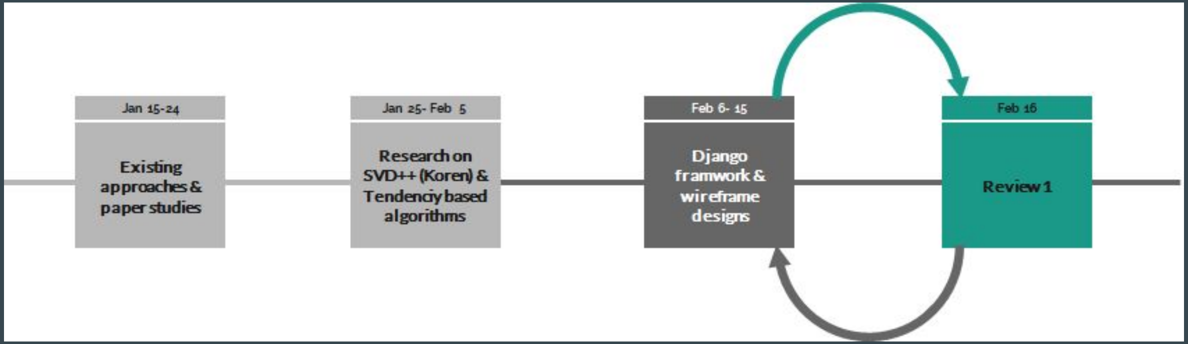
Observation of limitation

```
In [6]: print(get_recs(15 ,mat, 10))
[(49, 0.10965594579868189), (10, 0.07863596934848649), (2, 0.066052768778205151), (27, 0.063176682771133127), (31, 0.060732948051034172), (50, 0.059022527619310934), (1,
0.050947689174661873), (21, 0.048280304916188177), (29, 0.046333659296885404), (37, 0.045864393456715419)]

In [7]: df_m1=pd.read_sql_query('Select * from rating  where movieid <= 100;',conn)

In [8]: mat=df_m1.pivot_table(index=['custid'],columns=['movieid'],values='rating')

In [9]: print(get_recs(15 ,mat, 10))
[(94, 0.12582686399534174), (49, 0.10965594579868189), (62, 0.093562601214470117), (10, 0.07863596934848649), (61, 0.073127773471731536), (2, 0.066052768778205151), (27,
0.063176682771133127), (87, 0.062872052998006897), (31, 0.060732948051034172), (50, 0.059022527619310934)]
```

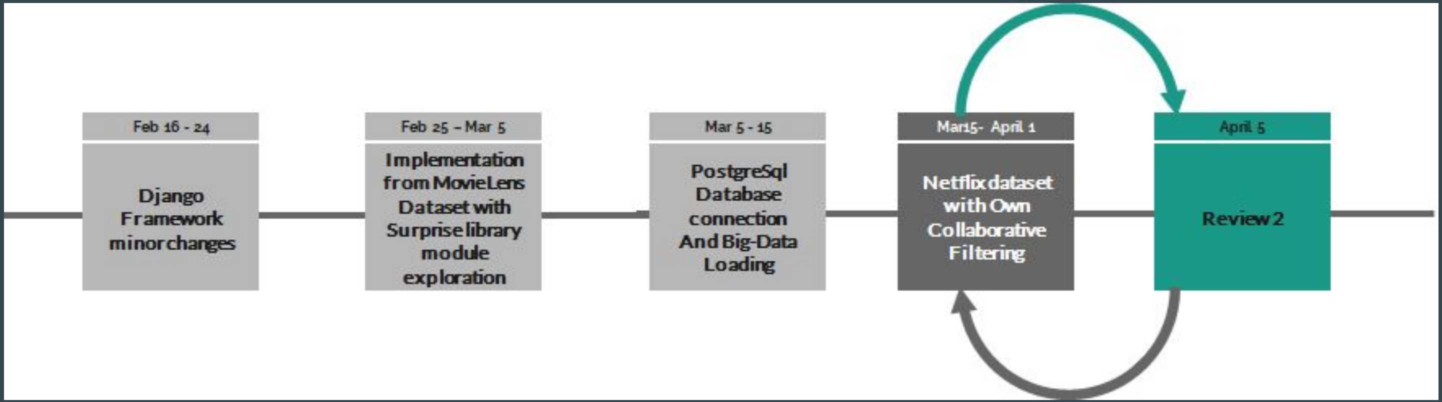
- Computationally inefficient (slow)
- More logical mapping for recommendation (Item-Item CF)
- Precomputation is slow
- But Prediction is lot faster than most of them [O(k)]

Timeline

Review : 1

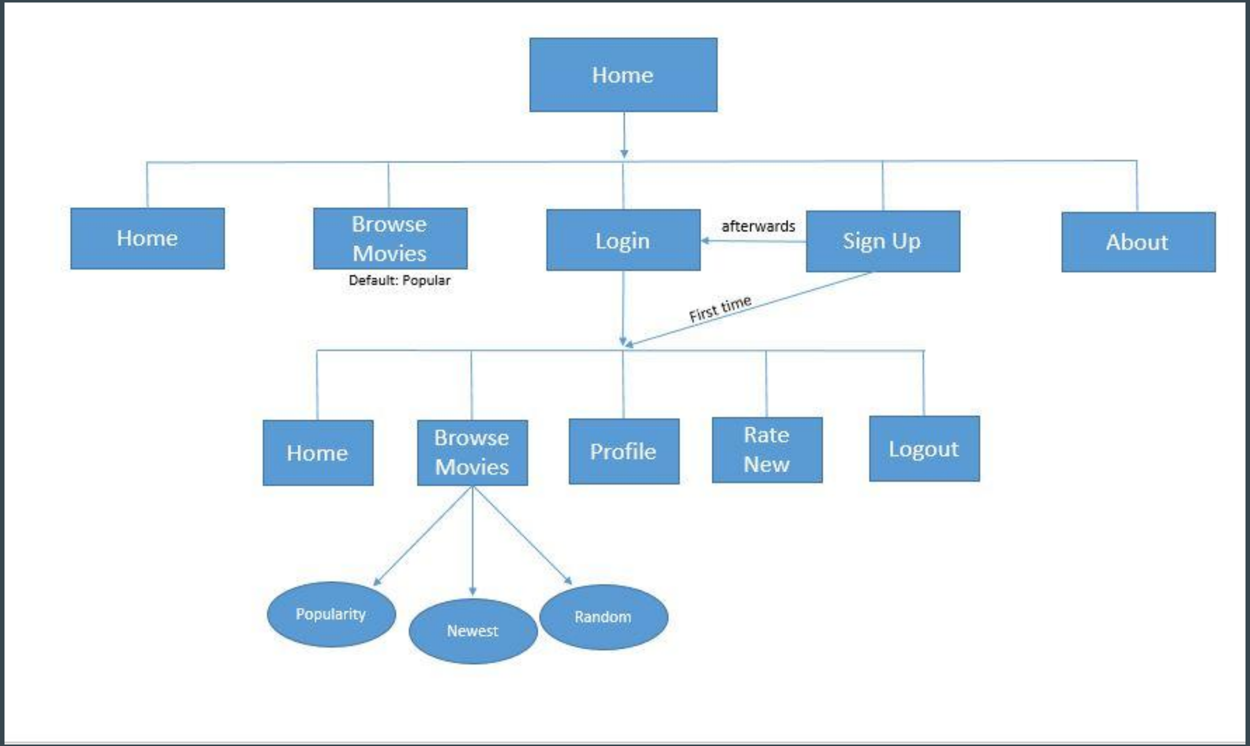


Review : 2

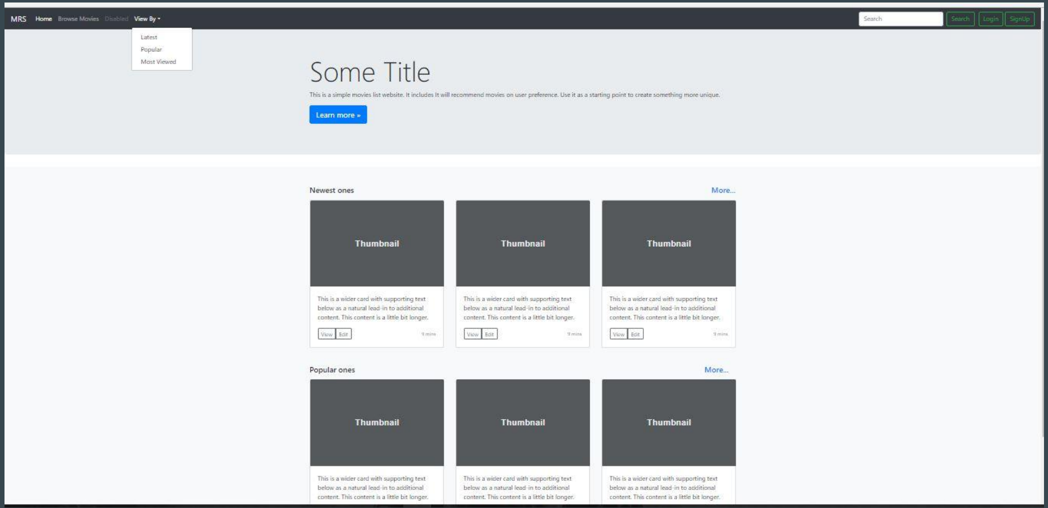


Previous Milestones - 1

Web structure & Flow diagram

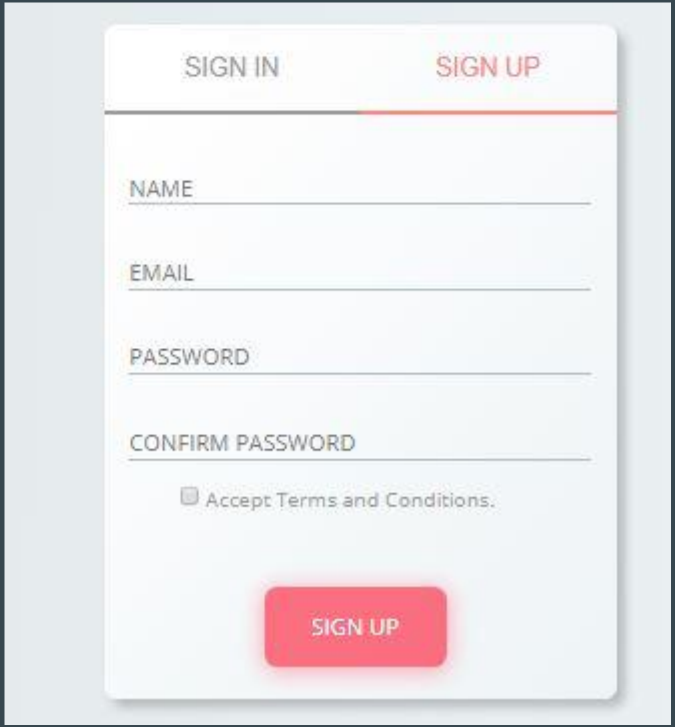


Previous Milestones - 2



Home Page framework

User specific login/SignUp module

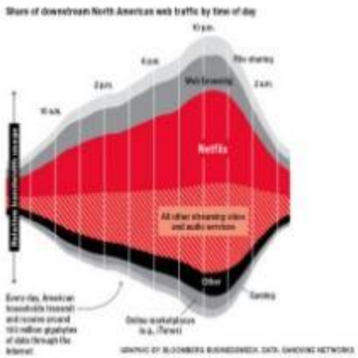


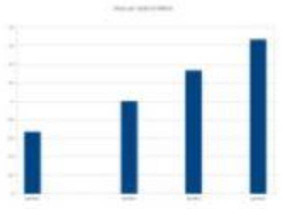
References

1. Knowledge based system 46 (2013) 109-132
2. Comparison of collaborative filtering algorithms: Limitations of Current techniques and proposals for scalable, high performance recommender systems
3. Evaluating recommender systems –Guy shani & Asela Gunawardana
4. MovieLens dataset: https://grouplens.org/datasets/movielens/
5. Netflix dataset: https://www.kaggle.com/netflix-inc/netflix-prize-data/data
6. Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model- Yehuda Koren
7. Tendencies-based collaborative filtering algorithm - Dilip Raj Baral
8. https://www.quora.com/Whats-the-difference-between-SVD-and-SVD++
9. Basic approach of CF: https://github.com/fastai/fastai/blob/master/courses/dl1/lesson5-movielens.ipynb
10. Pandas connection with PostgreSQL: https://www.youtube.com/watch?v=gC-0CaRzR48
11. Definition of recommendation system: https://www.slideshare.net/xamat/recommender-systems-machine-learning-summer-school-2014-cmu?qid=d81546be-e78c-4180-9437-cb5186197cd5&v=&b=&from_search=8

Extra Slides

Netflix Scale





- > 48M members
- > 40 countries
- > 1000 device types
- > 5B hours in Q3 2013
- Plays: > 50M/day
- Searches: > 3M/day
- Ratings: > 5M/day
- Log 100B events/day
- 34.2% of peak US downstream traffic

Extra slide - 2

- For algorithm detailed explanation visit extra slides of review 1
- 17770 x 4,40,189 = 8,532,958,530
  Given rating = 0,100,000,000
- CF final matrix =
  17770 x 17770 = 0,315,772,900