Team 4
Recommender Systems

21A530M - Pamela Sin Hui

21A466A - Ong Joo Kiat, Kenneth

21A471K - Shee Jing Le

Introduction

- Recommender system
- Three approaches
  - Content based filtering
  - Collaborative filtering - memory based
    - User-based
    - Item-based
  - Collaborative filtering - model based
    - SVD
- Evaluation
  - RMSE with ratings as a proxy

Dataset

- Movielens dataset
  - 26 million ratings
  - 45 thousand movies
  - 270 thousand users
- Movies metadata from TMDB

Content Based Filtering

- Leverages the features from items each user has previously rated to recommend items similarly preferred by that user explicitly.

Column name	Analysis	Final features
1. adult	99.9% of data are a Boolean False, hence we determine that this is not a useful feature to distinguish items from each other	
2. belongs_to_collection	A few clusters of movie collections (e.g. James Bond with 26 movies in this category). Majority of filled data have a single unique value. Further, 90.7% of data are NaN. Hence, we change this data to binary 0 and 1.	90.7% - 0 9.3% - 1
3. budget	80.5% of data are 0. As this is a numeric feature, we would prefer for the majority of cells to be filled.	
4. genres	Each movie item can have multiple genres. We find that this is a useful feature to determine the "essence" and theme of a movie.	Objective text: genres (Count vectoriser)
5. homepage	82.9% of data are NaN. Filled data are links to the movie homepage, and majority of filled data have a single unique homepage link (the top link which 12 movie items have is "http://www.georgecarlin.com". This may be a useful feature in determining movies with high budgets that have an exclusive known homepage. Hence, we change this data to binary 0 and 1.	82.9% - 0 17.1% - 1
6. id	Not a feature - Movie item identifier	
7. imdb_id	Not a feature - Movie item identifier	
8. original_language	89 unique values, with "en" being the top category with 71.0% of data. The next highest is "fr" with 5.36% of data. Quite a few categories only consist of 1 movie item, and these might be very unique movies with the language in an exotic tongue. We keep categories that have at least 200 movie items, and re-categorise the remaining as "original_language_infrequent".	71.0% - "en" 5.36% - "fr" . . 3.52% - "original_language_infrequent"
9. original_title	Comparing "original_title" and "title", we find that the data is the same for 74.9% of movie items. Hence we change this data to binary 0 and 1.	25.1% - 0 74.9% - 1
10. overview	About 143 movie items have "No Overview" or NaN. Filled data is varied with some having a short sentence of 10-plus words, while others have multiple sentences.These contain insight into the plot of the movie, hence we fill cells with no real data with an empty string and utilise this as a final feature	Subjective text: overview (TF-IDF vectoriser)

Content Based Filtering

11. popularity	Metric has its own method of determining popularity of the movie according to the movie database methodology.We discard this feature as we prefer to use other raw features to make a determination.	
12. poster_path	Poster webpage links that are typically unique. We decide this is not a useful feature.	
13. production_companies	Contains production companies involved in producing the movie. This may be a useful feature as each production company may produce a certain calibre film, or produce movies that perhaps certain audiences tend to rate highly.	Objective text: production company (Count vectoriser)
14. production_countries	Contains information about which countries where the movie was produced. The top 3 categories of filled data is 'United States of America' at 39.3%, 'United Kingdom' at 4.93% and 'France' at 3.63%.	Objective text
15. release_date	Not used	
16. revenue	Not used	
17. runtime	Length of the movie item. Feature is scaled using StandardScaler.	Scaled numeric feature
18. spoken_languages	Each movie item may have multiple spoken languages. 49.3% of	

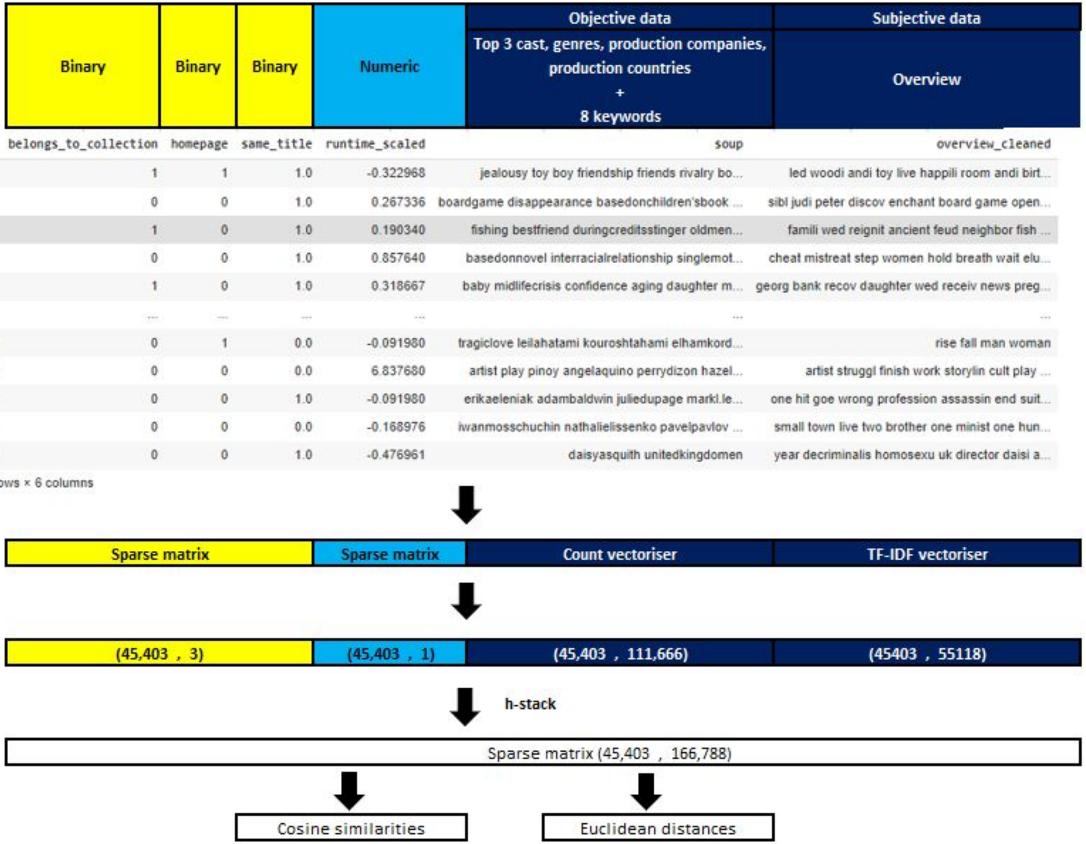
Content Based Filtering

19. status	Not used, majority of belong to one category.	
20. tagline	A simple few words that is a general teaser to the movie. Not used for final feature as we decide it may not be sufficient to capture things like the plot of the movie.	
21. title	Title of the movie. For example, we find that the title "Cinderella" was used for 11 movies, which might indicate different versions and adaptations of the story.	
22. video	99.8% of filled data are a Boolean False. Not a useful feature to distinguish items from each other	
23. vote_average	The average vote of the movie item based on the methodology of the movie database. This takes into account factors like how many people voted, where if a large number of people voted then their collective votes have a larger impact (versus a situation where for a movie there is only one vote, despite it having the top rating). We want to make the movie recommendation for a user independent of this in-house calculated metric, and thus count on other features to capture the essence of the movie.	
24. vote_count	The number of votes taken to derive the vote_average.	
25. movieId	Not a feature - Movie item identifier	
26. imdbId	Not a feature - Movie item identifier	
27. tmdbId	Not a feature - Movie item identifier	

Content Based Filtering

28. keywords	31.6% of movies have an empty list of keywords. Some movies only have a single keyword, for example 2.82% of movies have a single 'woman director'. Other movies have many keywords, where reading the keywords gives you an idea of how the movie starts and unfolds. For example, one movie had keywords like 'fire', 'bounty hunter', 'horseback riding', 'outlaw', 'unrequited love', 'pursuit', shot in the heart'.	
29. cast	Information like the actors and actresses that form part of the cast. The cast is the face of the movie and the audience sees the interpretation of the movie through the characters they play. We find that this is important information, for example what people generally deem as good or bad actors.	Objective text: Top 3 actors (Count vectoriser)
30. crew	Includes information like the name of the composer, editor and producer. We decide to use the name of the director as this is the person that brings the different aspects of the movie together.	Objective text: Director (Count vectoriser)

Content Based Filtering



	belongs_to_collection	homepage	same_title	runtime_scaled	soup	overview_cleaned
0	1	1	1.0	-0.322968	jealousy toy boy friendship friends rivalry bo...	led woodi andi toy live happili room andi birt...
1	0	0	1.0	0.267336	boardgame disappearance basedonchildren'sbook ...	sibl judi peter discov enchant board game open...
2	1	0	1.0	0.190340	fishing bestfriend duringcreditsstinger oldmen...	famili wed reignit ancient feud neighbor fish ...
3	0	0	1.0	0.857640	basedonnovel interracialrelationship singlemot...	cheat mistreat step women hold breath wait elu...
4	1	0	1.0	0.318667	baby midlifecrisis confidence aging daughter m...	georg bank recov daughter wed receiv news preg...
...	...	...	...	...	...	...
45398	0	1	0.0	-0.091980	tragiclove leilahatami kouroshtahami elhamkord...	rise fall man woman
45399	0	0	0.0	6.837680	artist play pinoy angelaquino perrydizon hazel...	artist struggl finish work storylin cult play ...
45400	0	0	1.0	-0.091980	erikaeleniak adambaldwin juliedupage markl.le...	one hit goe wrong profession assassin end suit...
45401	0	0	0.0	-0.168976	iwanmosschuchin nathalielissenko pavelpavlov ...	small town live two brother one minist one hun...
45402	0	0	1.0	-0.476961	daisyasquith unitedkingdomen	year decriminalis homosexu uk director daisi a...

45403 rows × 6 columns

- Binary and numeric data
- Objective data

  Count vectoriser to convert to matrix of token counts.

- Subjective data

  Remove stop words, lemmatization to reduce words to root form

  TF-IDF vectoriser scale down impact of tokens that occur very frequently and empirically less informative

$$idf(t) = \log [(1 + n) / (1 + df(t))] + 1$$

Content Based Filtering (Cosine similarity)

$$K(X, Y) = \langle X, Y \rangle / (||X||*||Y||)$$

1. 26 million ratings - 70% train, 30% test split
2. Cosine similarity scores with all movies computed (normalised dot product of each movie with other movies)

For each user rating of a movie in the test set:

- Cosine similarity scores with all the movies in the train set for that user is extracted, and sorted in decreasing order.
- Top three similarity scores obtained (three movies in the train set for that user that have the closest similarity to the test set movie in question for that user).
- Simple average and weighted average (against cosine similarities) determined

1000 users in the test set, RMSE and MAE determined

Content Based Filtering (Euclidean distances)

```
dist(x, y) = sqrt(dot(x, x) - 2 * dot(x, y) + dot(y, y))
```

Sklearn implementation modifies formula for computational efficiency in dealing with sparse matrices.

Generally same methodology as that using cosine similarity except that:

- Euclidean distances sorted in increasing order (smallest three distances obtained).
- To get the weighted average, we weight the three closest movie ratings to the inverse of each movie's euclidean distances (distance inversely proportional to similarity).

Content Based Filtering

```
# When viewing recommended movies for users, we notice that some movies tend to be commonly recommended.
# Viewing the features of these commonly recommended movies, we notice that the "soup" column tends to be short with common words with other movies like "unitedstatesofamericaen".
# The lack of other words and the appearance of common words is perhaps what leads to a resultant close similarity with other movies.
commonly_recommended = ['The Drunk','Superpower', 'To γάλα','Saved by the Bell: Wedding in Las Vegas',"The First Annual 'On Cinema' Oscar Special","The Fourth Annual 'On Cinema' Oscar Special", 'Made For Ea
movies_df2.loc[movies_df2.title.isin(commonly_recommended),['title','belongs_to_collection', 'homepage', 'same_title', "runtime_scaled",'soup', 'overview_cleaned']]
```

	title	belongs_to_collection	homepage	same_title	runtime_scaled	soup	overview_cleaned
22963	Superpower	0.0	1.0	1.0	0.652317	documentarysuperpowerproductions unitedstat...	superpow illustr unit state leverag posit ensu...
23199	Made For Each Other	0.0	0.0	1.0	0.164675	comedy romance unitedstatesofamericaen	pair creat lover stranger take marriag imposs
30343	The Drunk	0.0	1.0	1.0	-0.091980	unitedstatesofamericaen	hard drink grandson legendari labor leader get...

Content Based Filtering

	Content-based (euclidean distance,3-NN, simple average)	Content-based (euclidean distance,3-NN, weighted average)	Content-based (cosine similarity,3-NN, simple average)	Content-based (cosine similarity,3-NN, weighted average)
RMSE	1.0642	1.0623	0.9856	0.9816
MAE	0.8188	0.8172	0.7471	0.7442
Test set	30,327 predictions (~1000 users in test set)		30,327 predictions (~1000 users in test set)	
Time	Approximately 4.5 hours, high-ram setting on Colab Pro		Approximately 4.5 hours, high-ram setting on Colab Pro	

- Weighted average is marginally closer to the actual ratings
- Fine-tuning of the extent to which each of the three movies (in the train set for that user) are similar to the movie we want a prediction for

Content Based Filtering (Final thoughts)

- More **interpretable**. Ability to dissect the recommendations of the method based on the features that contribute more highly to that item.
- **Less prone to cold start** problem. New items suggested to a user with a history before substantial number of users rate.
- Recommend items to users with **unique tastes**, or unpopular with other users.
- **Features** need to be well-defined
- There needs to be a **history of ratings** for a user to start recommending
- This dataset has some information disparity
- Future exploration - Word vectors with rich semantic meaning (**Word2Vec**)

Collaborative filtering memory-based

- K-nearest neighbours
- Approximate nearest neighbours
- Cosine distance vs adjusted cosine distance
- KNN vs ANN
- Hyper parameters
- Results
- Recommendations
- Things I would have done differently

Collaborative filtering memory-based

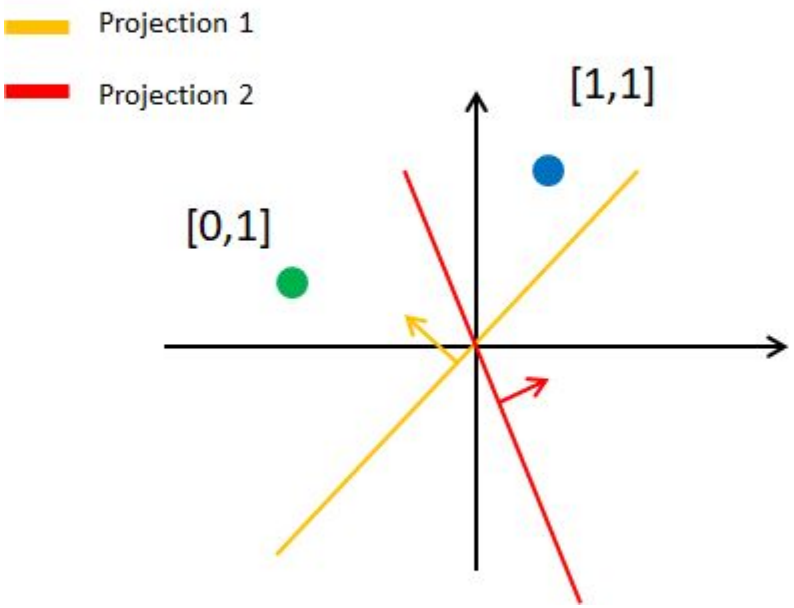
K-nearest neighbours

- Two types of neighbours
  - User-based
  - Item-based
- Challenges
  - Computational and memory intensive
  - Need to store and access the whole dataset

Collaborative filtering memory-based

Approximate nearest neighbours

- Locality sensitive hashing with random projections
  - N random hyperplanes are created
  - Each data point is hashed according to its relationship to each of the hyper planes

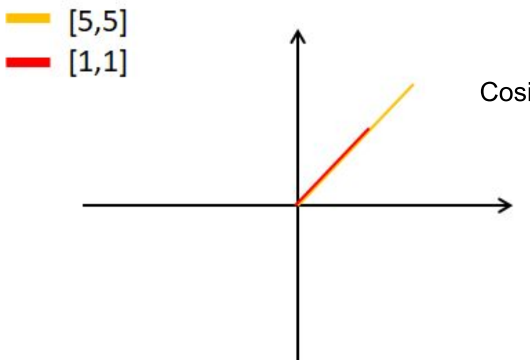


Collaborative filtering memory-based

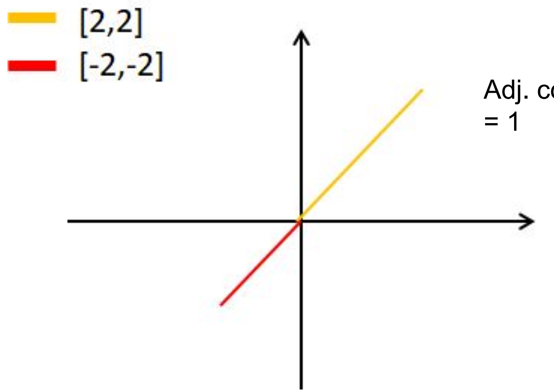
Cosine distance vs adjusted cosine distance

- Cosine distance
  - Does not takes into account user-mean and variance
- Adjusted cosine distance
  - Instead of the ratings, takes the difference between the rating and the user-mean

$$AC(i,j) = \frac{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \overline{r}_u)(r_{uj} - \overline{r}_u)}{\sqrt{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \overline{r}_u)^2 \sum_{u \in \mathcal{U}_{ij}} (r_{uj} - \overline{r}_u)^2}}.$$



Cosine distance = 0

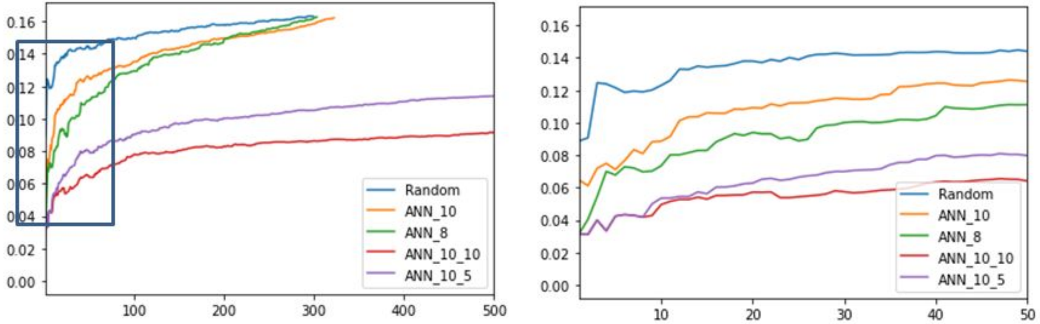


Adj. cosine distance = 1

Collaborative filtering memory-based

KNN vs ANN

user	distance	ranking	distance difference
99427	0.000000	0	0.000000
113398	0.782422	12	0.031259
136314	0.787963	21	0.031058
168310	0.797279	33	0.039950
265138	0.798431	39	0.033262
242346	0.811666	82	0.042379
17021	0.815594	104	0.043351
40369	0.816043	106	0.042834
202085	0.818018	121	0.041783
36045	0.829184	212	0.049945



Plot of distance differences between similarly ranked users.

	kNN	aNN	random
K = 10	14	36	54
K = 50	1	25	28
K =100	0	21	23
K = 417	0	17	16

kNN vs. aNN, number of movie ratings not found

Collaborative filtering memory-based

Hyper parameters

- No. of projections
- No. of hash-tables
- No. of nearest neighbours

Collaborative filtering memory-based

Results

	KNN Scaled, K =100	Item-based Scaled, Proj = 8, Tables = 5	User-based Scaled, Proj = 10, Tables = 5
Results	1.04/0.77	1.03/0.76	0.97/0.75

Collaborative filtering memory-based

Recommendations

- Filtering
  - Based on no. of neighbours
    - Too little neighbours
    - Too many neighbours

Collaborative filtering memory-based

Things I would have done differently

- Starting with a smaller dataset for experimentation/exploration
- Better planning of experiments

Collaborative filtering model-based

Think of the model as a salesman. Train a salesman and get the salesman to recommend the movies to the user.

Data
Model selection
Model description
Hyperparameter filtering
Results
Predictions
Summary

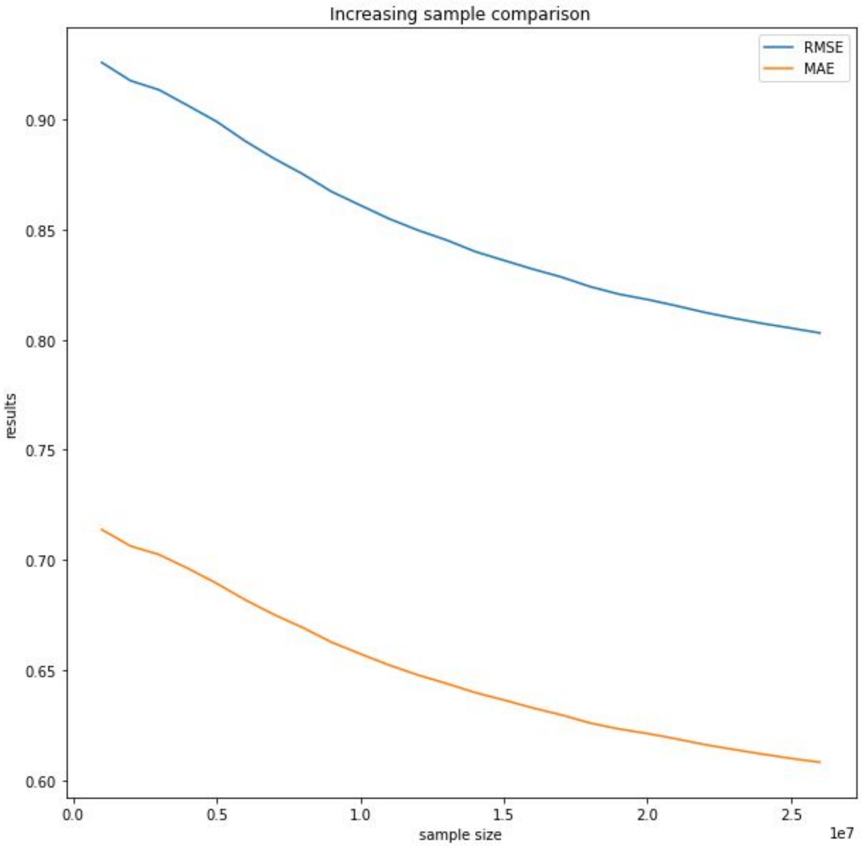
Collaborative filtering model-based

Data Size vs model incompetency:
Qns: How can we find the optimal data size?

Full dataset ~26million ratings

Attributes - ‘UserID’, ‘MovieID’, ‘ratings’

- X-axis => sample size
- Y-axis => results
- RMSE
- MAE



Collaborative filtering model-based

Which model to use?

ml-100k	RMSE	MAE	Time	Efficiency
SVD	0.936	0.738	0:00:13	12.1689
SVDpp	0.919	0.722	0:07:41	423.801
NMF	0.964	0.758	0:00:15	14.4581
SlopeOne	0.944	0.742	0:00:09	8.49888
KNNBasic	0.979	0.773	0:00:10	9.78931
KNNWithMeans	0.95	0.749	0:00:11	10.4518
KNNBaseline	0.93	0.733	0:00:13	12.0893
CoClustering	0.966	0.757	0:00:06	5.79391
BaselineOnly	0.944	0.748	0:00:01	0.943637
NormalPredictor	1.522	1.222	0:00:01	1.52214

ml-1m	RMSE	MAE	Time	Efficiency
SVD	0.874	0.686	0:02:13	116.204
SVDpp	0.862	0.672	2:30:04	7761.04
NMF	0.916	0.724	0:02:28	135.631
SlopeOne	0.907	0.714	0:03:07	169.517
KNNBasic	0.923	0.727	0:07:21	406.965
KNNWithMeans	0.929	0.738	0:07:29	417.188
KNNBaseline	0.895	0.706	0:07:54	424.176
CoClustering	0.916	0.718	0:01:10	64.0868
BaselineOnly	0.909	0.719	0:00:15	13.6293
NormalPredictor	1.506	1.207	0:00:16	24.0972

	RMSE			Efficiency		
	100k	1m	Improvement	100k	1m	Weighted time factor
SVD	0.936	0.874	7.09%	12.1689	116.204	9.55
SVDpp	0.919	0.862	6.61%	23.801	7761.04	326.08
NMF	0.964	0.916	5.24%	14.4581	135.631	9.38
SlopeOne	0.944	0.907	4.08%	8.49888	169.517	19.95
KNNBasic	0.979	0.923	6.07%	9.78931	406.965	41.57
KNNWithMeans	0.95	0.929	2.26%	10.4518	417.188	39.92
KNNBaseline	0.93	0.895	3.91%	12.0893	424.176	35.09
CoClustering	0.966	0.916	5.46%	5.79391	64.0868	11.06
BaselineOnly	0.944	0.909	3.85%	0.943637	13.6293	14.44
NormalPredictor	1.522	1.506	1.06%	1.52214	24.0972	15.83

Test the model using cross validation and smaller datasets to find the optimal model

Notice SVDpp has better results but the time taken is ridiculous

Collaborative filtering model-based

SVD Model

$$A = USV^T$$

Matrix factorisation technique

- U represents the relationship between users and latent factors
- S describes the strength of each latent factor
- V indicates the similarity between movies and latent factors

Collaborative filtering model-based

Hyperparameters Tuning:

Goal - reduce the following error term:

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2 \right)$$

4 keys hyperparameters:
n_epochs = the number of times the SGD procedure is iterated.
lr_all = the learning rate. The learning rate determine how fast the algo moves from 1 epoch to the next.
n_factors = the number of factors in the matrix.
reg_all = regularization factor. Higher means higher variance of predictions

Collaborative filtering model-based

GridSearchCV:

Used a smaller set

```
# Previous gridsearch result {'n_epochs': 30, 'lr_all': 0.0025, 'n_factors': 80}
sample = 1000000

grid = {'n_epochs': [30, 35, 40],
        'lr_all': [.0015, .0020, .0025, .003, .0035, .0040, .0045],
       'n_factors': [80, 90, 100, 110, 120]}

gs = GridSearchCV(SVD, grid, measures=['rmse', 'mae'], cv=5, n_jobs= -1)
ratingsmini = ratings.sample(n=sample, replace=False, random_state=42, )
reader = Reader()
rawmini = Dataset.load_from_df(ratingsmini[['userId','movieId','rating']],reader)
gs.fit(rawmini)

print(gs.best_score['mae'])
print(gs.best_score['rmse'])
print(gs.best_params['mae'])
print(gs.best_params['rmse'])
```

```
0.7119057726703276
0.9234525099674864
{'n_epochs': 30, 'lr_all': 0.0025, 'n_factors': 80}
{'n_epochs': 30, 'lr_all': 0.0025, 'n_factors': 80}
```

app.log X

1 to 10 of 18 entries Filter

Timestamp	Level	Message
Aug 23, 2021, 8:43:31 AM	WARNING	WARNING:root:kernel 7e3d6412-4f2b-44dd-8889-c288a0d2abc5 restarted
Aug 23, 2021, 8:43:31 AM	INFO	KernelRestarter: restarting kernel (1/5), keep random ports
Aug 23, 2021, 8:32:40 AM	INFO	Adapting to protocol v5.1 for kernel 7e3d6412-4f2b-44dd-8889-c288a0d2abc5
Aug 23, 2021, 8:32:37 AM	INFO	Kernel started: 7e3d6412-4f2b-44dd-8889-c288a0d2abc5
Aug 23, 2021, 8:25:08 AM	INFO	Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
Aug 23, 2021, 8:25:08 AM	INFO	http://172.28.0.12:9000/
Aug 23, 2021, 8:25:08 AM	INFO	The Jupyter Notebook is running at:
Aug 23, 2021, 8:25:08 AM	INFO	0 active kernels
Aug 23, 2021, 8:25:08 AM	INFO	Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
Aug 23, 2021, 8:25:08 AM	INFO	http://172.28.0.2:9000/

Show 10 per page 1 2

CRASHED!

Colab

```
from surprise.model_selection import GridSearchCV

grid = {'n_epochs': [5, 10, 15, 20, 25, 30],
        'lr_all': [.001, .0025, .005, .0075, .01],
       'n_factors': [80, 90, 100, 110, 120]}

gs = GridSearchCV(SVD, grid, measures=['rmse', 'mae'], cv=5, n_jobs= -1)
gs.fit(raw)

print(gs.best_score['mae'])
print(gs.best_score['rmse'])
print(gs.best_params['mae'])
print(gs.best_params['rmse'])
```

```
C:\Users\sjlal\anaconda3\lib\site-packages\joblib\externals\loky\process_executor.py:688: UserWarning: A worker stopped while s
ome jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.
  warnings.warn(
```

Jupyter

Collaborative filtering model-based

Hyperparameters Tuning results

Sample size full (~26m)							
	Epoch	LR	Factors	Reg	MAE	RMSE	
Default	20	0.005	100	0.02	0.6083	0.8036	Comments
Using HP of Gridsearch with sample	30	0.0025	80	0.02	0.6134	0.8079	Worst than default
Increase factors	30	0.0025	120	0.02	0.6131	0.8074	Improvement but marginal
Increase LR	30	0.005	120	0.02	0.6062	0.8029	Improvement and significant
Increase Reg	30	0.005	120	0.04	0.6080	0.8018	Improvement RMSE but not MAE
Decrease Reg	30	0.005	120	0.03	0.6026	0.7966	Improvement RMSE and MAE from reg = 0.02
Increase factors	30	0.005	140	0.03	0.6025	0.7964	Improvement but marginal
Increase factors	30	0.005	160	0.03	0.6024	0.7962	Improvement but marginal
Factors = 120, increase Epoch	40	0.005	120	0.03	0.5997	0.7938	Improvement and significant
Factors = 120, increase Epoch	50	0.005	120	0.03	0.5996	0.7940	Marginal Improvement MAE but not RSME
Factors = 120, increase LR	40	0.006	120	0.03	0.6006	0.7953	No improvement
3 HP fixed, testing factors	40	0.005	160	0.03	0.5996	0.7934	Marginal RMSE improvement
3 HP fixed, testing factors	40	0.005	200	0.03	0.5998	0.7934	No improvement
3 HP fixed, testing factors	40	0.005	180	0.03	0.5998	0.7935	No improvement

Collaborative filtering model-based

Predictions accuracy 1

-Compare if the predicted top 10 movies is in the top 10 movies of the testset

-Result of 78% is not meaningful because the predict list of movies is taken from the testset; we are just re-ranking the testset and comparing back to it

```
print(recommendedmovies[0])
print(truepreference[0])
```

```
[101382, [858, 1221, 527, 953, 356, 1387, 608, 1304, 1267, 1210]]
[101382, [1210, 364, 1263, 527, 1276, 356, 1302, 858, 508, 1304]]
```

```
def top_n_accuracy(recommendedmovies,truepreference):

    uid = []
    score = []
    a = 0
    for i in recommendedmovies:
        x = 0
        y = 0

        for j in i[1]:
            if j in truepreference[a][1]:
                x += 1
                y += 1
            else:
                y += 1

        acc = x/y
        a += 1
        uid.append(i[0])
        score.append(acc)

    return [sum(score)/len(score),uid,score]
```

```
prediction_score = top_n_accuracy(recommendedmovies,truepreference)
```

```
prediction_score[0]
```

```
0.7807301336442153
```

Collaborative filtering model-based

Predictions accuracy 2

- top 10 predictions of a user from the full movie database divided by the number of 5 star ratings of the same user

- Result of 0.4 and 0.8 for 2 different users is not meaningful because the predicted ratings is on the fullset of movies and part of it is the trainset which the user already watched

```
In [132]: user101382movie5results
Out[132]: defaultdict(list,
                      {101382: [(858, 5),
                        (1221, 5),
                        (1097, 5),
                        (260, 4.977891434033649),
                        (527, 4.918608089054963),
                        (2028, 4.909324110426823),
                        (912, 4.905609121316953),
                        (1193, 4.898487125892857),
                        (111, 4.884755531562192),
                        (1196, 4.861945850044397)]})
```

```
In [148]: len(user101382movie5reallist)
Out[148]: 48
```

```
In [144]: score = []
          for i in user101382movie5resultslist[0][1]:
              x = 0
              y = 0
              if i in user101382movie5reallist:
                  x += 1
                  y += 1
              else:
                  y += 1

              acc = x/y
              score.append(acc)

          print(sum(score)/len(score))

0.8
```

```
In [164]: all_predict = []
          for i in movieids:
                  all_predict.append(

          user270893movie5results = g
          user270893movie5real = rati
          user270893movie5reallist =
          user270893movie5resultslist
          # Append the recommended it
          for uid, user_ratings in us
              user270893movie5results

          score = []
          for i in user270893movie5resultslist[0][1
              x = 0
              y = 0
              if i in user270893movie5reallist:
                  x += 1
                  y += 1
              else:
                  y += 1

              acc = x/y
              score.append(acc)

          print(sum(score)/len(score))

0.4
```

Collaborative filtering model-based

Predictions accuracy 3

- top 10 unwatched movies divided by going to watch movies
- unwatched movies = not rated by user in trainset
- going to watch movies = list of movies in the testset

- Ran on 100 users, result of 8.6% is meaningful because the predictions come from the remaining arsenal of the salesman

```
    user_movie_results = get_top_n(all_predict,n=10)
    user_movie_results_list = []

    # Append the recommended items for each user
    for uid, user_ratings in user_movie_results.items():
        user_movie_results_list.append([uid, [iid for (iid, _) in user_ratings]])

    score = []
    for i in user_movie_results_list[0][1]:
        x = 0
        y = 0
        if i in movies_watched_later:
            x += 1
            y += 1
        else:
            y += 1

        acc = x/y
        score.append(acc)
    combinedscore.append(sum(score)/len(score))

print(sum(combinedscore)/len(combinedscore))
```

```
0.08686868686868685
```

Collaborative filtering model-based

Summary

Challenges:
- High computation power needed. Unable to run the optimal model - SVDpp
- Even SVD has to be run manually (colab and gridsearch crash on bigger databases)

Things to consider going forward:
- Look for better measures of accuracy (altho i think the prediction 3 i came out with is great)
- Intro a time decay to the ratings as preference changes over time

Comparison across models

- SVD, model-based CF has the lowest RMSE
- SVD, model-based CF also is the fastest
- Memory-based CF vs model-based CF
  - Memory-based CF is based on explicit features on either item or user
  - Model-based CF tries to find latent features in both item and user
- Content-based is highly dependent on the item features

	Content-based (euclidean distance,3-NN, weighted average)	Content-based (cosine similarity,3-NN, weighted average)	Memory-based CF - User-based KNN	Memory-based CF - User-based ANN	Memory-based CF - Item-based ANN	Model-based CF SVD
RMSE	1.0623	0.9816	1.0367	0.9666	1.0332	0.7934
MAE	0.8172	0.7442	0.7713	0.7466	0.7623	0.5996