Movie Recommendation System via Markovian Factorization of Matrix Process

DINTO DAVI T
S2 M.Tech CSE
Reg.No:TCR19CSCE08

Guided By:
RAHAMATHULLA K
Assistant Professor
Department of Computer Science and Engineering

GEC Thrissur

Outline

Introduction

- Recommendation systems which are becoming a norm for consumer industries such as books, music, clothing, movies, news articles, places, utilities, etc.
- There are majorly six types of recommender systems which work primarily in the Media and Entertainment industry: Collaborative Recommender system, Content-based recommender system, Demographic based recommender system, Utility based recommender system, Knowledge based recommender system and Hybrid recommender system.
- This project created using Markovian factorization of matrix process (MFMP) model.

Introduction(cont...)

- we call the first-order MFMP and the second-order MFMP. The two models assume the latent processes to be respectively first-order and second order Gaussian Markov processes on the inner-product processes is also assumed to be Gaussian, taking effect additively.
- Customer preferences for products are drifting over time. Product perception and popularity are constantly changing as new selection emerges. Similarly, customer inclinations are evolving, leading them to ever redefine their taste.
- MFMP model family are capable of capturing the temporal dynamics in the dataset.

Problem definition,problem analysis & design

- allows the model to evolve over time in order to capture the dynamics of "concept drift" in collaborative filtering
- Concept Drift: User purchase preferences that we want to predict are always drifting.
- For example, a man previously liked romantic movies. So he rated Bangalore Days , a romantic movie released in 2014, the highest score 5 three years ago. But he changes his interest as time goes by. Currently, he dislikes romantic movies. He rated Oru Adaar Love , a romantic movie released in 2019, the score 2 a month ago.

Problem definition,problem analysis & design (cont...)

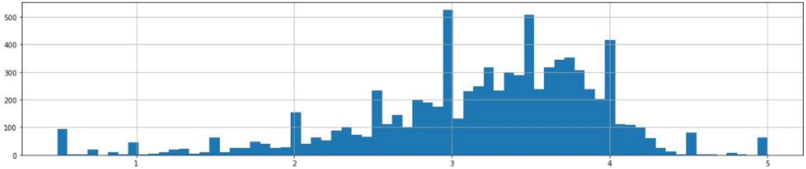
- Cold start happens when new users or new items arrive in e-commerce platforms.
- Classic recommender systems like collaborative filtering assumes that each user or item has some ratings so that we can infer ratings of similar users/items even if those ratings are unavailable.

Data Analysis

#total range of data set (i.e from minimum value to maximum value) is divided into 8 to 15 equal parts. These equal parts are known as bins

In [22]:
```
import matplotlib.pyplot as plt
%matplotlib inline

plt.figure(figsize=(20,4)) #width 20inches and height 4inches
ratings['rating'].hist(bins=70)
```

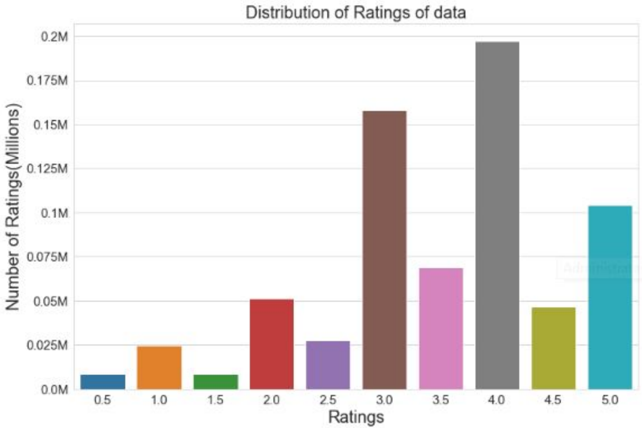
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x2c735312508>



it is gaussian normal distribution.PDF probability density function satisfy this

Data Analysis (cont...)

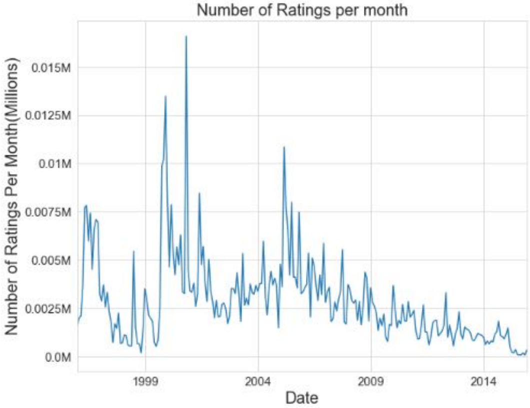
In [27]:
```
plt.figure(figsize = (12, 8))
ax = sns.countplot(x="rating", data=df)

ax.set_yticklabels([changingLabels(num) for num in ax.get_yticks()])

plt.tick_params(labelsize = 15)
plt.title("Distribution of Ratings of data", fontsize = 20)
plt.xlabel("Ratings", fontsize = 20)
plt.ylabel("Number of Ratings(Millions)", fontsize = 20)
plt.show()
```



Data Analysis (cont...)

Number of Ratings per month

In [35]:
```
plt.figure(figsize = (10,8))
ax = df.resample("M", on = "Date")["rating"].count().plot()
ax.set_yticklabels([changingLabels(num) for num in ax.get_yticks()])
ax.set_title("Number of Ratings per month", fontsize = 20)
ax.set_xlabel("Date", fontsize = 20)
ax.set_ylabel("Number of Ratings Per Month(Millions)", fontsize = 20)
plt.tick_params(labelsize = 15)
plt.show()
```



Result

Markov process

In [196]:
```
import mchmm as mc
```

Markov process in which the time is discrete.Markov chain is a stochastic process over a discrete state space satisfying the Markov property.The probability of moving from the current state to the next state depends solely on the present state.

In terms of probability distribution, given that the system is at time instance n, the conditional distribution of the states at the next time instance, n + 1, is conditionally independent of the state of the system at time instances {1, 2, . . ., n-1}.

$$Pr(X_{n+1} = x | X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = Pr(X_{n+1} = x | X_n = x_n)$$

In [198]:
```
states=MF.columns
```

In [199]:
```
states
```

Out[199]:
```
Index([''Round Midnight (1986)', ''Til There Was You (1997)',
       ''burbs, The (1989)', ''night Mother (1986)',
       '*batteries not included (1987)',
       '...All the Marbles (California Dolls, The) (1981)',
       '...And God Spoke (1993)', '...And Justice for All (1979)',
       '1-900 (06) (1994)', '10 (1979)',
       ...
       'Zoolander (2001)', 'Zoot Suit (1981)',
       'Zorba the Greek (Alexis Zorbas) (1964)', 'Zorro, the Gay Blade (1981)',
       'Zulu (1964)', 'Zus & Zo (2001)', 'eXistenZ (1999)', 'xXx (2002)',
       '¡Three Amigos! (1986)', 'À nous la liberté (Freedom for Us) (1931)'],
      dtype='object', name='title', length=8075)
```

In [200]:
```
transition_matrix=preds_df
```

In [201]:
```
transition_matrix = np.atleast_2d(transition_matrix)
states = states
index_dict = {states[index]: index for index in
                          range(len(states))}
state_dict = {index: states[index] for index in
                          range(len(states))}
```

Result (cont...)

```
In [208]: a = mc.MarkovChain().from_data(preds_df)

In [214]: a.observed_matrix

Out[214]: array([[0., 1., 0., ..., 0., 0., 0.],
                 [0., 0., 1., ..., 0., 0., 0.],
                 [0., 0., 0., ..., 0., 0., 0.],
                 ...,
                 [0., 0., 0., ..., 0., 1., 0.],
                 [0., 0., 0., ..., 0., 0., 1.],
                 [0., 0., 0., ..., 0., 0., 0.]])

In [219]: a.observed_p_matrix

Out[219]: array([[ 0.,  1.,  0., ...,  0.,  0.,  0.],
                 [ 0.,  0.,  1., ...,  0.,  0.,  0.],
                 [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                 ...,
                 [ 0.,  0.,  0., ...,  0.,  1.,  0.],
                 [ 0.,  0.,  0., ...,  0.,  0.,  1.],
                 [nan, nan, nan, ..., nan, nan, nan]])

In [223]: aa=pd.DataFrame(a.observed_matrix, index=a.states, columns=a.states, dtype=float)

In [227]: a.expected_matrix

Out[227]: array([[0.        , 0.00012385, 0.00012385, ..., 0.00012385, 0.00012385,
                  0.00012385],
                 [0.        , 0.00012385, 0.00012385, ..., 0.00012385, 0.00012385,
                  0.00012385],
                 [0.        , 0.00012385, 0.00012385, ..., 0.00012385, 0.00012385,
                  0.00012385],
                 ...,
                 [0.        , 0.00012385, 0.00012385, ..., 0.00012385, 0.00012385,
                  0.00012385],
                 [0.        , 0.00012385, 0.00012385, ..., 0.00012385, 0.00012385,
                  0.00012385],
                 [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                  0.        ]])

In [231]: a.n_order_matrix(a.observed_p_matrix, order=2)

Out[231]: array([[nan, nan, nan, ..., nan, nan, nan],
```

Result (cont...)

```
In [235]: a.chisquare(a.observed_matrix, a.expected_matrix, axis=None)
Out[235]: Power_divergenceResult(statistic=nan, pvalue=nan)

In [239]: ids, states = a.simulate(10, start="Home Alone (1990)", seed=100)

In [240]: ids
Out[240]: array([3367, 3368, 3369, 3370, 3371, 3372, 3373, 3374, 3375, 3376])

In [241]: states
Out[241]: array(['Home Alone (1990)', 'Home Alone 2: Lost in New York (1992)',
                 'Home Alone 3 (1997)', 'Home Fries (1998)', 'Home Page (1999)',
                 'Home Room (2002)', 'Home for the Holidays (1995)',
                 'Home of Our Own, A (1993)', 'Home on the Range (2004)',
                 'Homegrown (1998)'], dtype='<U134')

In [242]: ",".join(states)
Out[242]: 'Home Alone (1990),Home Alone 2: Lost in New York (1992),Home Alone 3 (1997),Home Fries (1998),Home Page (1999),Home Room (200
          2),Home for the Holidays (1995),Home of Our Own, A (1993),Home on the Range (2004),Homegrown (1998)'
```

PERFORMANCE METRIC

- Root Mean Square Error(RMSE)
  RMSE is the error of each point which is squared. Then mean is calculated. Finally root of that mean is taken as final value

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

PERFORMANCE METRIC(cont...)

```
In [335]: def run_surprise(algo, trainset,testset, model_name):
              startTime = datetime.now()

              train = dict()
              test = dict()

              algo.fit(trainset)

          #-----------------Evaluating Train Data-----------------#
              print("-"*50)
              print("TRAIN DATA")
              train_pred = algo.test(trainset.build_testset())
              # https://surprise.readthedocs.io/en/stable/getting_started.html"
              train_actual, train_predicted = get_ratings(train_pred)
              train_rmse, train_mape = get_error(train_pred)
              print("RMSE = {}".format(train_rmse))
              print("MAPE = {}".format(train_mape))
              print("-"*50)
              train = {"RMSE": train_rmse, "MAPE": train_mape, "Prediction": train_predicted}

          #----------------Evaluating Test Data-----------------#
              print("TEST DATA")
              test_pred = algo.test(testset)
              test_actual, test_predicted = get_ratings(test_pred)
              test_rmse, test_mape = get_error(test_pred)
              print("RMSE = {}".format(test_rmse))
              print("MAPE = {}".format(test_mape))
              print("-"*50)
              test = {"RMSE": test_rmse, "MAPE": test_mape, "Prediction": test_predicted}

              print("Time Taken = "+str(datetime.now() - startTime))
              make_table(model_name, train_rmse, train_mape, test_rmse, test_mape)
              return train,test
```

```
In [336]: error_table = pd.DataFrame(columns = ["Model", "Train RMSE", "Train MAPE", "Test RMSE", "Test MAPE"])
          model_train_evaluation = dict()
          model_test_evaluation = dict()
```

```
In [337]: def make_table(model_name, rmse_train, mape_train, rmse_test, mape_test):
              global error_table
              error_table = error_table.append(pd.DataFrame([[model_name, rmse_train, mape_train, rmse_test, mape_test]],
                                                            columns = ["Model", "Train RMSE", "Train MAPE", "Test RMSE", "Test MAPE"]))
              error_table.reset_index(drop = True, inplace = True)
```

PERFORMANCE METRIC(cont...)

In [338]:
```
algo = SVD(n_factors = gs.best_params['rmse']['n_factors'], biased=True, verbose=True)
train_result, test_result = run_surprise(algo, trainset,testset, "SVD")
model_train_evaluation["SVD"] = train_result
model_test_evaluation["SVD"] = test_result
```

```
Processing epoch 0
Processing epoch 1
Processing epoch 2
Processing epoch 3
Processing epoch 4
Processing epoch 5
Processing epoch 6
Processing epoch 7
Processing epoch 8
Processing epoch 9
Processing epoch 10
Processing epoch 11
Processing epoch 12
Processing epoch 13
Processing epoch 14
Processing epoch 15
Processing epoch 16
Processing epoch 17
Processing epoch 18
Processing epoch 19
--------------------------------------------------
TRAIN DATA
RMSE = 0.7297093669161853
MAPE = 23.55769316736481
--------------------------------------------------
TEST DATA
RMSE = 0.9862218968330987
MAPE = 33.31743951713742
--------------------------------------------------
Time Taken = 0:01:58.641797
```

PERFORMANCE METRIC(cont...)

In [339]:
```
param_grid  = {'n_factors': [5,7,10,15,20,25,35,50,70,90]}
gs = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=3)
gs.fit(data)
# best RMSE score
print(gs.best_score['rmse'])
# combination of parameters that gave the best RMSE score
print(gs.best_params['rmse'])
# best mae score
print(gs.best_score['mae'])
# combination of parameters that gave the best mae score
print(gs.best_params['mae'])
```

```
0.8501647493298675
{'n_factors': 50}
0.6532501331623201
{'n_factors': 50}
```

RMSE SCORE IS 0.8501647

```
In [280]: ### Average Rating Per User
          AvgRatingUser = getAverageRatings(TrainUISparseData, True)
          print("Average rating of user 5216 = {}".format(AvgRatingUser[5216]))
```

```
Average rating of user 5216 = 3.8855421686746987
```

```
In [292]: ### Average Rating Per Movie
          AvgRatingMovie = getAverageRatings(TrainUISparseData, False)
          print("Average rating of movie 4500 = {}".format(AvgRatingMovie[4500]))
```

```
Average rating of movie 4500 = 3.9722222222222223
```

```
In [278]: total_users = len(np.unique(df["userId"]))
          train_users = len(AvgRatingUser)
          uncommonUsers = total_users - train_users

          print("Total number of Users = {}".format(total_users))
          print("Number of Users in train data= {}".format(train_users))
          print("Number of Users not present in train data = {}({}%)".format(uncommonUsers, np.round((uncommonUsers/total_users)*100), 2))
```

```
Total number of Users = 5216
Number of Users in train data= 5216
Number of Users not present in train data = 0(0.0%)
```

Cold Start Problem with Movies

```
In [279]: total_movies = len(np.unique(df["movieId"]))
          train_movies = len(AvgRatingMovie)
          uncommonMovies = total_movies - train_movies

          print("Total number of Movies = {}".format(total_movies))
          print("Number of Movies in train data= {}".format(train_movies))
          print("Number of Movies not present in train data = {}({}%)"
                .format(uncommonMovies, np.round((uncommonMovies/total_movies)*100), 2))
```

```
Total number of Movies = 8075
Number of Movies in train data= 8075
Number of Movies not present in train data = 0(0.0%)
```

CONCLUSION

- More general models beyond those relying on Gaussian distributions and additive drifts should be considered when there is significant presence of such phenomenon in the datasets
- MFMP models are a rich family of probabilistic models that marry the framework of PMF with the framework of Hidden Markov Models.
- By varying the order of the latent Markov processes, the involved distributions and the dependency of observation on the latent process, a large variety of temporal dynamical models can be constructed for collaborative filtering problems.

References I

- Y Mao R Zhang.
  Movie recommendation via markovian factorization of matrix processes.
  IEEE Access, 2019.
- R. Bell Y. Koren and C. Volinsky.
  Matrix factorization techniques for recommender systems,.
  IEEE Comput., 42(8):30–37, aug 2009.
- S. Li Y. Xia Z. You Q. Zhu X. Luo, M. C. Zhou and H. Leung.
  An efficient second-order approach to factorize sparse matrices in recommender systems.
  IEEE Trans. Ind. Informat., 11:946–956, aug 2018.

References II

M. Lin J. Zhang, Y. Lin and J. Liu.
An effective collaborative filtering algorithm based on user preference clustering,.
In *Appl. Intell.*, volume 45, pages 230–240. IEEE, 2016.


Thank you