Recommendation engine system using 'surprise' package

Moon Chu 07/17/2021

Index

Data Prep

1. Inspecting the data

DATASET :

- Simple dataset with 4 columns : userID/productID/rating/timestamp

	userId	productId	rating	timestamp
0	AKM1MP6P0OYPR	0132793040	5.0	1365811200
1	A2CX7LUOHB2NDG	0321732944	5.0	1341100800
2	A2NWSAGRHCP8N5	0439886341	1.0	1367193600
3	A2WNBOD3WNDNKT	0439886341	3.0	1374451200
4	A1GI0U4ZRJA8WN	0439886341	1.0	1334707200

- Original Dataset : 4.2 mm
- Sampled Dataset : 15 K

```
sampled = df.sample(frac=0.002, replace=True, random_state=1)
sampled.columns = ['userId', 'productId','rating','timestamp']
```

```
#sampled one
sampled.nunique()
```

```
userId       15570
productId    11721
rating           5
timestamp     2793
dtype: int64
```

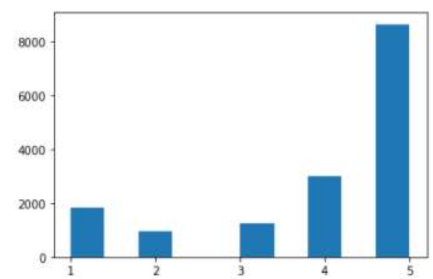
Data Prep

1. Inspecting the data

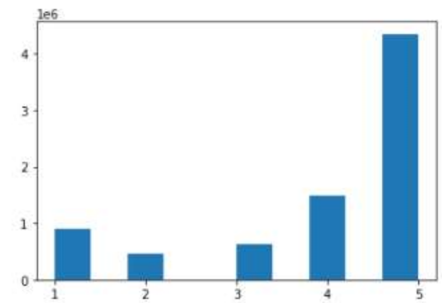
DATASET :

- Rating Distribution : Sampled vs Entire dataset is comping pretty close

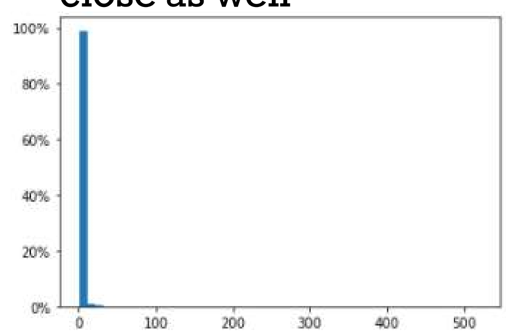
```
from pylab import hist, show,xticks

hist(sampled['rating'])
xticks(range(1, 6))
show()
```

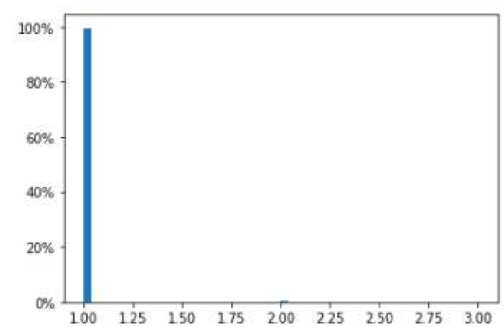



```
hist(df['rating'])
xticks(range(1, 6))
show()
```



- % of users with number of review Ratio Distribution: over 98% of users are 1 review users – Sampled vs Entire dataset's trend is close as well





Data Prep

2. Data Pre Process (optional)

DATASET :

- If the dataset is too sparse (e.g. users with low exposure, such as only 1 review writing) we can set arbitary threshold to filter this user out.

- e.g) below filters out users that reviewed less than 100 electronics

```
countdf=df['userId'].value_counts()
```

```
countdf.head(10)
```

```
A5JLAU2ARJ0BO     520
ADLVFFE4VBT8      501
A3OXHLG6DIBRW8    498
A6FIAB28IS79      431
A680RUE1FD08B     406
A1ODOGXEYECQQ8    380
A36K2N527TXXJN    314
A2AY4YUOX2N1BQ    311
AWPODHOB4GFWL     308
ARBKYIVNYWK3C     296
Name: userId, dtype: int64
```

```
len(df[df['userId'].isin(countdf[countdf >= 100].index)])
```

```
44209
```

Data Prep

3. Data Loading

Surprise package needs to be fed with 3 inputs. 1) User 2) Product 3) Review/Rating.

For the review we have to specify the min and max value of the raw data in the reader class in surprise.

```
sampled['rating'].min(), sampled['rating'].max()
```

```
(1.0, 5.0)
```

```
# our rating scale start from 1 and end at 5. so rating_scale = (1,5)
reader = Reader(rating_scale=(1, 5))

# Load the data
data = Dataset.load_from_df(sampled[['userId','productId','rating']], reader)
```

Theoretical Part

Collaborative filtering

- Approach : use the "wisdom of crowd" to recommend items
- Basic Assumption : customers who had similar tastes in the past, will have the similar tastes in the future.
- Input : user- item matrix
- Output : Numerical prediction indicating what degree the user will like an item.

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

Theoretical Part

Collaborative filtering :
1. **user based** (also memory based : rating matrix directly used for making prediction)

- Measuring Simliarity in user based CF :

- **A popular similarity measure in user-based CF: Pearson correlation**
  - a, b : users
  - $r_{a,p}$: rating of user a for item p
  - P : set of items, rated both by a and b
  - Possible similarity values between -1 and 1

$$sim(a,b) = \frac{\sum_{p \in P}(r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P}(r_{a,p} - \bar{r}_a)^2}\sqrt{\sum_{p \in P}(r_{b,p} - \bar{r}_b)^2}}$$

	Item1	Item2	Item3	Item4	Item5	
Alice	5	3	4	4	?	
User1	3	1	2	3	3	sim = 0,85
User2	4	3	4	3	5	sim = 0,00
User3	3	3	1	5	4	sim = 0,70
User4	1	5	5	2	1	sim = -0,79

Theoretical Part

Collaborative filtering :
1. **user based** (also memory based : rating matrix directly used for making prediction)

- **Making Predictions**

- A common prediction function:

$$pred(a,p) = \overline{r_a} + \frac{\sum_{b \in N} sim(a,b) * (r_{b,p} - \overline{r_b})}{\sum_{b \in N} sim(a,b)}$$

- Calculate, whether the neighbors' ratings for the unseen item i are higher or lower than their average
- Combine the rating differences – use the similarity with a as a weight
- Add/subtract the neighbors' bias from the active user's average and use this as a prediction

Theoretical Part

Collaborative filtering :
2. **item** based (also model based : based on item's offline processing)

- Item – item collaboration filtering
- This means that instead of using user simliarity, we use "item" simliarity measure to calculate prediction
- Look for items that are simliar to item 5
- Take Alice's ratings for these items to predict the rating for item 5

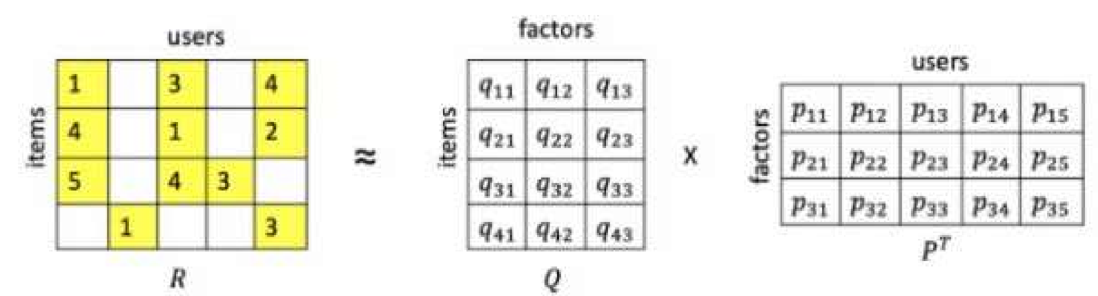
	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

Theoretical Part

Matrix Factorization

Rating = Matrix product of Item x User factors

Algorithm like SVD builds a recommendation system by allowing us to “fill in the gaps” in the rating matrix, predicting the ratings that each user would assign to each item in the dataset



Theoretical Part

Matrix Factorization

While collaborative filtering (user-user based, or item-item based) is simple and intuitive, matrix factorization techniques are usually more effective because they allow us to discover the latent features between interactions between users and items.

SVD (Single Vector Decomposition) uses the gradient descent to minimize squared error (predicted vs actual) to eventually get the best model.

The prediction $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

If user u is unknown, then the bias b_u and the factors p_u are assumed to be zero. The same applies for item i with b_i and q_i.

To estimate all the unknown, we minimize the following regularized squared error:

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

The minimization is performed by a very straightforward stochastic gradient descent:

$$\begin{aligned} b_u &\leftarrow b_u + \gamma(e_{ui} - \lambda b_u) \\ b_i &\leftarrow b_i + \gamma(e_{ui} - \lambda b_i) \\ p_u &\leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda p_u) \\ q_i &\leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda q_i) \end{aligned}$$

Modeling

Training Algorithm and Prediction

Load Surprise Package and try with SVD which is one type of max factorization for intial review

```
from surprise import SVD
from surprise import Dataset
from surprise import accuracy
from surprise.model_selection import train_test_split

# sample random trainset and testset
# test set is made of 25% of the ratings.
trainset, testset = train_test_split(data, test_size=.25)

# We'll use the famous SVD algorithm.
algo = SVD()

# Train the algorithm on the trainset, and predict ratings for the testset
algo.fit(trainset)
predictions = algo.test(testset)

# Then compute RMSE
accuracy.rmse(predictions)
```

```
RMSE: 1.3825
```

Modeling

Model Comparison

Feed all available methods and its output

Algorithm	test_rmse	test_mae	fit_time	test_time
SVDpp	1.377803	1.095880	1.059564	0.034984
KNNBaseline	1.378202	1.095874	2.432896	0.031995
BaselineOnly	1.378851	1.097196	0.050977	0.030989
SVD	1.379675	1.097729	0.734374	0.034712
CoClustering	1.387084	1.102525	1.994677	0.066081
KNNWithMeans	1.387436	1.103432	2.505972	0.041654
KNNBasic	1.387747	1.103736	2.336442	0.095003
SlopeOne	1.387864	1.103250	1.541768	0.040649
KNNWithZScore	1.387890	1.103885	2.965362	0.051003
NMF	1.388518	1.105515	2.076149	0.039602
NormalPredictor	1.766150	1.353648	0.016650	0.041642

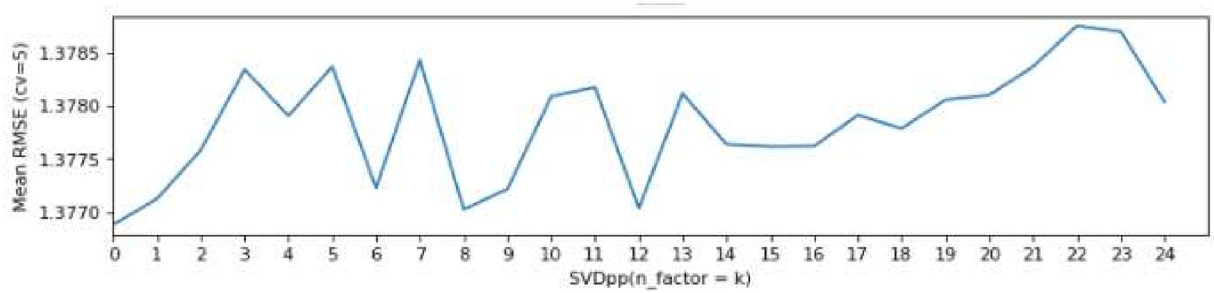
Models available in Surprise Package

Algorithm class name	Description
random_pred.NormalPredictor	Algorithm predicting a random rating based on the distribution of the training set, which is assumed to be normal.
baseline_only.BaselineOnly	Algorithm predicting the baseline estimate for given user and item.
knns.KNNBasic	A basic collaborative filtering algorithm.
knns.KNNWithMeans	A basic collaborative filtering algorithm, taking into account the mean ratings of each user.
knns.KNNBaseline	A basic collaborative filtering algorithm taking into account a baseline rating.
matrix_factorization.SVD	The famous SVD algorithm, as popularized by Simon Funk during the Netflix Prize.
matrix_factorization.SVDpp	The SVD++ algorithm, an extension of SVD taking into account implicit ratings.
matrix_factorization.NMF	A collaborative filtering algorithm based on Non-negative Matrix Factorization.
slope_one.SlopeOne	A simple yet accurate collaborative filtering algorithm.
co_clustering.CoClustering	A collaborative filtering algorithm based on co-clustering.

Comparison

Model Tuning

- Among 11 models, SVDpp performed best.
- We will try to make SVDpp better by finding optimal k to minimize rmse.



- It looks like k= 8, 12 seems to be minimizing rmse.

Comparison

Model Tuning

- Upon feeding 6,8,9,12 (by looking at k that had dips), k=6 gave the best optimal result that makes the model generalizable, i.e. avoid over and underfitting, the grid algorithm finds n_factors = 6 optimal.

```
param_grid = {'n_factors': [6,8,9,12]}
gs = GridSearchCV(SVD, param_grid, measures=['rmse'], cv=5)
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'])
```

```
1.3779308666276702
{'n_factors': 6}
```

Comparison

Historical vs Prediction comparison

- Comparing Historical data and Prediction
- Step1 :Map the predictions to each user.
- Step2 :

  - i.) recommendations for any given userId and

  - ii.) the user's historical ratings
- #Step3 :Return the above objects with specific reference in a readable format (i.e. tidy DataFrame)

```
algo_SVDpp = SVDpp(n_factors = 6)
algo_SVD.fit(trainset)
# Then predict ratings for all pairs (u, i) that are NOT in the training set.
testset = trainset.build_anti_testset()

predictions = algo.test(testset)

top_n = get_top_n(predictions, n=10)

# Print the recommended items for each user
for uid, user_ratings in top_n.items():
    print(uid, [iid for (iid, _) in user_ratings])
```