DATA WORKSHOP

DW Poznań - projekt filmweb-rekomendacje #4

2019-12-17
Rekomendacje

http://bit.ly/2LWGi6X

Agenda

01. **Algorytmy Ewaluacji**

02. Cosine Similarity i Mise en Scene

03. Collaborative Filtering

04. Algorytmy SVD

https://github.com/alexiej/filmweb-rekomendacje

https://colab.research.google.com/drive/16Jfvhe_viHdFVq9QAIbO29LCVNNpyvxz

01. Algorytmy Ewaluacji (dataset)

- Ładowanie dataset
  - train_set
  - test_set

```
userId,movieId,rating,timestamp
1,1,4.0,964982703
1,3,4.0,964981247
1,6,4.0,964982224
1,47,5.0,964983815
1,50,5.0,964982931
1,70,3.0,964982400
1,101,5.0,964980868
1,110,4.0,964982176
1,151,5.0,964984041
```

```
[26] from surprise import Dataset
     from surprise import Reader

     np.random.seed(0)
     random.seed(0)
```

```
[27] reader = Reader(line_format='user item rating timestamp', sep=',', skip_lines=1)
     """
     line_format - list of columns
     sep - separator for csv file
     skip_lines - start from the second line
     """
```

```
'\nline_format - list of columns\nsep - separator for csv file\nskip_lines - start from the second line\n'
```

```
dataset = Dataset.load_from_file(RATING_PATH, reader=reader)
```

```
[ ] %%time
    from surprise.model_selection import train_test_split
    from surprise.model_selection import LeaveOneOut
    from surprise import KNNBaseline

    class RecommendationDataSet:
        def __init__(self, dataset, test_size = .25):
          self.dataset = dataset
          self.full_dataset = dataset.build_full_trainset()

          # Train Set, Test Set to test results
          self.train_set, self.test_set = train_test_split(dataset, test_size=test_size, random_state=1)
```

https://colab.research.google.com/drive/16Jfvhe_viHdFVq9QAIbO29LCVNNpyvxz

01. Algorytmy Ewaluacji

- MAE (Mean Absolute Error)

$$\frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

- RMSE (Root Mean Absolute Error)

$$\sqrt{\frac{\sum_{i=1}^{n} (y_i - x_i)^2}{n}}$$

```
[29] print('Rest Set:', recommendation_dataset.test_set[:10])
     predictions = svd.test(recommendation_dataset.test_set)
     print('Predictions: ', predictions[:10])

     from surprise import accuracy

     accuracy.mae(predictions, verbose=True)
     accuracy.rmse(predictions, verbose=True)
```

```
Rest Set: [('167', '1196', 4.5), ('605', '2291', 3.5), ('472', '3298', 4.0),
Predictions:  [Prediction(uid='167', iid='1196', r_ui=4.5, est=4.150335799784
MAE:  0.6732
RMSE: 0.8779
0.87790565300794
```

https://colab.research.google.com/drive/16Jfvhe_viHdFVq9QAIbO29LCVNNpyvxz

01. Algorytmy Ewaluacji

- Top-n HitRate

User	Movie	Position
User1	Godzilla	0
User1	StarGate	1 (HIT)
User1	Independance Day	2
User2	Godzilla	0
User2	Terminator	1 (HIT)
User2	Independance Day	2

$$top\_n\_hitrate = \frac{hits}{users} = \frac{2}{2} = 1.0$$

Divide by the number of recommendation in the top N (N=3). Our hit rate is 33%

https://colab.research.google.com/drive/16Jfvhe_viHdFVq9QAIbO29LCVNNpyvxz

01. Algorytmy Ewaluacji (Test-Set)

1. Dataset (train_set, test_set)

	i1	i2	i3	i4	i5
u10	5	2	3		2
u20	2	3		2	

```
np.random.seed(0)
random.seed(0)

dataframe = pd.DataFrame(
    [[10, 1, 5],
     [10, 2, 2],
     [10, 3, 3],
     [10, 5, 2],
     [20, 1, 2],
     [20, 2, 3],
     [20, 4, 2]
     ], columns=['uid','iid', 'rating'])

sample_dataset = Dataset.load_from_df(dataframe,reader=Reader(rating_scale=(1,5)))
train_set, test_set = train_test_split(sample_dataset, test_size=0.2, random_state=1)
```

2. Leave One Out Z naszego zbioru usuwamy jeden element i budujemy zbiór treningowy i testowy

	i1	i2	i3	i4	i5
u10	5	2	3		2
u20	2	3		2	

```
loo  = LeaveOneOut(n_splits=1, random_state=1)
# svd.test(leave_one_out_test_set)
loo_train_set, loo_test_set = list(loo.split(sample_dataset))[0]
print('Leave One Out Test Set', loo_test_set)
```

```
2. Leave One Out Test Set [(10, 2, 2.0), (20, 1, 2.0)]
```

https://colab.research.google.com/drive/16Jfvhe_viHdFVq9QAIbO29LCVNNpyvxz

01. Algorytmy Ewaluacji (Test-Set)

3. Anti-Test Set

	i1	i2	i3	i4	i5
u10	5	2	3	X	2
u20	2	3	X	2	X

```
loo_anti_test_set =  loo_train_set.build_anti_testset()
print('3. Leave one out: Anti Test Set', loo_anti_test_set)
```

```
3. Leave one out: Anti Test Set [(10, 2, 3.0), (10, 4, 3.0), (20, 1, 3.0),
```

4. Algorytmowi stworzyć predykcję zwrócić top-n elementów

u1	item	pred
1.	i2	3.5
2.	i4	3.5

u1	item	pred
1.	i1	3.5
2.	i3	3.5

```
loo_anti_test_prediction = svd.test(loo_anti_test_set)
loo_anti_test_topn = get_top_n(loo_anti_test_prediction, 2, 1.0)
print('4. Leave one out predictions', loo_anti_test_topn)
```

```
4. Leave one out predictions defaultdict(<class 'list'>, {'10': [('2', 3.5048725984106204)
```

https://colab.research.google.com/drive/16Jfvhe_viHdFVq9QAIbO29LCVNNpyvxz

01. Algorytmy Ewaluacji (Test-Set)

5. Test Hit rate on the anti-test topn predictions

```
print('Train by SVD on train_set')
svd_sample = SVD(random_state=10)
svd_sample.fit(loo_train_set)


print('5. Hit Rate: ', HitRate(loo_anti_test_topn, loo_test_set))
```

```
Train by SVD on train_set
5. Hit Rate:  1.0
```

https://colab.research.google.com/drive/16Jfvhe_viHdFVq9QAIbO29LCVNNpyvxz

01. Algorytmy Ewaluacji

- Cumulative Hit Rate (cHR) - Polega tylko i wyłącznie na tym że usuwamy w leave-One-Out te elementy których ocena przez użytkownika jest poniżej wartości.

	i1	i2	i3	i4	i5
u10	5	4	3		2
u20	2	3		2	

```
35] loo_test_prediction = svd_sample.test(loo_test_set)
    print('Cumulative Hit Rate: ',
          CumulativeHitRate(loo_anti_test_topn, loo_test_prediction, 4.0))

    Cumulative Hit Rate:  1.0
```

https://colab.research.google.com/drive/16Jfvhe_viHdFVq9QAIbO29LCVNNpyvxz

01. Algorytmy Ewaluacji

- Rating Hit Rate (rHR) - Sprawdzamy HitRate w podziale na wszystkie możliwe oceny

```
print('Rating Hit Rate: ',
      RatingHitRate(loo_anti_test_topn, loo_test_prediction))
```

```
Rating Hit Rate:  {2.0: 1.0, 4.0: 1.0}
```

https://colab.research.google.com/drive/16Jfvhe_viHdFVq9QAIbO29LCVNNpyvxz

01. Algorytmy Ewaluacji

- Average Reciprocal HitRate (ARHR)

$$arhr = \frac{\sum_{i=1}^{n} \frac{1}{rank_i}}{users}$$

	i1	i2	i3	i4	i5
u10	5	2	3		2
u20	2	3		2	

u1	item	pred
1.	i2	3.5
2.	i4	3.5

u20	item	pred
1.	i1	3.5
2.	i3	3.5

```
print('Average Reciprocal HitRate (ARHR): ',
      AverageReciprocalHitRank(loo_anti_test_topn, loo_test_prediction))


loo_anti_test_topn2 = {
 10: [(2,3.5), (4,3.5)],
 20: [(3,3.5), (1,3.5)]
}

print('\nAverage Reciprocal HitRate (ARHR): \n\
(1/rank_user10 + 1/rank_user20)/2 = (1/1 + 1/2)/2 = \n',
      AverageReciprocalHitRank(loo_anti_test_topn2, loo_test_prediction))
```

```
Average Reciprocal HitRate (ARHR):  1.0

Average Reciprocal HitRate (ARHR):
(1/rank_user10 + 1/rank_user20)/2 = (1/1 + 1/2)/2 =
 0.75
```

https://colab.research.google.com/drive/16Jfvhe_viHdFVq9QAIbO29LCVNNpyvxz

01. Algorytmy Ewaluacji

- **User Coverage**
  Oblicza się jako procent par <user,item> które Mogą zostać przewidziane z wartością wyższą niż **treshold** podzielona przez liczbę użytkowników.
  Jeśli użytkownik ma co najmniej 1 film który został trafiony uważa się to za 1.0

	i1	i2	i3	i4	i5
u10	5	2	3	2.98	2
u20	2	3	2.77	2	2.59

u10	item	pred
1.	i4	2.98

u20	item	pred
1.	i3	2.77
2.	i5	2.59

$$user\_coverage(>= 2.9) = \frac{user1\_hits + user2\_hits}{num\_users} = \frac{1+0}{2} = 0.5$$

$$user\_coverage(>= 2.0) = \frac{user1\_hits + user2\_hits}{num\_users} = \frac{1+1}{2} = 1.0$$

01. Algorytmy Ewaluacji

- **Diversity**: Dla każdej pary filmów w topn wyliczane jest similarity (S) i dzielone przez Liczbę wszystkich wyliczeń.

  Diversity to (1-S)

	i1	i2	i3	i5	i4
i1	1	0.96	1	1	1
i2	0.96	1	1	1	1
i3	1	1	1	1	0
i5	1	1	1	1	0
i4	1	1	0	0	1

```
For user10: There is no pair. Similarity = 0.0
For user20: Similarity between i3 and i5 is:  1.0
Diversity is (1-1/1) =  0.0
```

```
Sample 2 {10: [(2, 4.0), (1, 3.0)], 20: [(3, 4.0), (5, 3.0)]}
For user10: There is pair(i2,i1). Similarity = 0.965
For user20: There is pair(i3,i5). Similarity = 1.0
Diversity is: (1- (0.965+1.0)/2) = 0.0175
Computing the cosine similarity matrix...
Done computing similarity matrix.
Diversity:  0.01719212073966514
```

https://colab.research.google.com/drive/16Jfvhe_viHdFVq9QAIbO29LCVNNpyvxz

01. Algorytmy Ewaluacji

u10	item	pred
1.	i4	2.98

u20	item	pred
1.	i3	2.77
2.	i5	2.59

- **Novelty** - Sum wszystkich rankingów filmów z topn (miejsce w ilości wszystkich ocen) podzielona przez ilość ocen topn

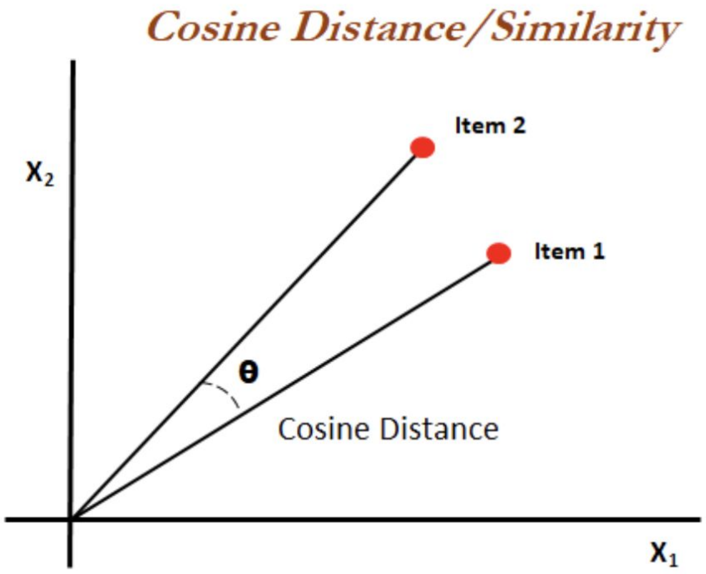
```
print('Popularity rankings: \n', get_popularity_ranking(dataset))
print('Ranking: ', sample_anti_topn)
print('For Movie 4 rank is: 5, movie 3 is: 4, movie 5 is 3')
print('Novelty is: (5+4+3)/3 = 12/3 = 4.0')
print('Novelty is: ', Novelty(sample_anti_topn, get_popularity_ranking(dataset)))
```

```
Popularity rankings:
 defaultdict(<class 'int'>, {1: 1, 2: 2, 3: 3, 5: 4, 4: 5})
Ranking:  defaultdict(<class 'list'>, {10: [(4, 2.9810168611713936)], 20: [(3, 2.77784218
For Movie 4 rank is: 5, movie 3 is: 4, movie 5 is 3
Novelty is: (5+4+3)/3 = 12/3 = 4.0
Novelty is:  4.0
```

ranking	movie	Ilość ocen
1	i1	2
2	i2	2
3	i3	1
4	i4	1
5	i5	1

02. Cosine Similarity

$$\cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$




```
def calculate_feature_similarity(self, feature1, feature2):
    # feature1 = np.array(list(feature1))
    # feature2 = np.array(list(feature2))
    intersection = feature1 * feature2
    return intersection.sum() / math.sqrt(feature1.sum() * feature2.sum())

def compute_year_similarity(self, y1, y2):
    diff = abs(y2 - y1)
    sim = math.exp(-diff / 10.0)
    return sim

def compute_mise_en_scene_similarity(self, movie1, movie2, mise):
    if not movie1 in mise.keys():
      return 1.0
    if not movie2 in mise.keys():
      return 1.0

    mes1 = mise[movie1]
    mes2 = mise[movie2]


    shotLengthDiff = math.fabs(mes1[0] - mes2[0])
    colorVarianceDiff = math.fabs(mes1[1] - mes2[1])
    motionDiff = math.fabs(mes1[3] - mes2[3])
    lightingDiff = math.fabs(mes1[5] - mes2[5])
    numShotsDiff = math.fabs(mes1[6] - mes2[6])
    return shotLengthDiff * colorVarianceDiff * motionDiff * lightingDiff * numShotsDiff
```

https://colab.research.google.com/drive/16Jfvhe_viHdFVq9QAIbO29LCVNNpyvxz

02. Cosine Similarity (top-k similar ratings)

```
def estimate(self, u, i):

    if not (self.trainset.knows_user(u) and self.trainset.knows_item(i)):
        raise PredictionImpossible('User and/or item is unkown.')

    # Build up similarity scores between this item and everything the user rated
    neighbors = []
    for rating in self.trainset.ur[u]: #w datasesie user rating
        genreSimilarity = self.similarity[i,rating[0]]
        neighbors.append( (genreSimilarity, rating[1]) )

    # Extract the top-K most-similar ratings
    k_neighbors = heapq.nlargest(40, neighbors, key=lambda t: t[0])

    # Compute average sim score of K neighbors weighted by user ratings
    simTotal = weightedSum = 0
    for (simScore, rating) in k_neighbors:
        if (simScore > 0):
            simTotal += simScore
            weightedSum += simScore * rating

    if (simTotal == 0):
        raise PredictionImpossible('No neighbors')

    predictedRating = weightedSum / simTotal


    #add to results
    self.results[int(self.trainset.to_raw_iid(i))] = predictedRating
    return predictedRating
```

```
Evaluating RMSE, MAE of algorithm SimilarityAlgorithm on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9932  0.9833  0.9688  0.9874  0.9757  0.9817  0.0086
MAE (testset)     0.7673  0.7587  0.7466  0.7630  0.7539  0.7579  0.0072
Fit time          715.63  739.89  726.09  748.31  719.81  729.95  12.32
Test time         7.99    7.68    7.92    7.58    8.15    7.87    0.21
CPU times: user 1h 1min 24s, sys: 17.6 s, total: 1h 1min 42s
Wall time: 1h 1min 29s
```

https://www.youtube.com/watch?v=6mGMBipt7kU

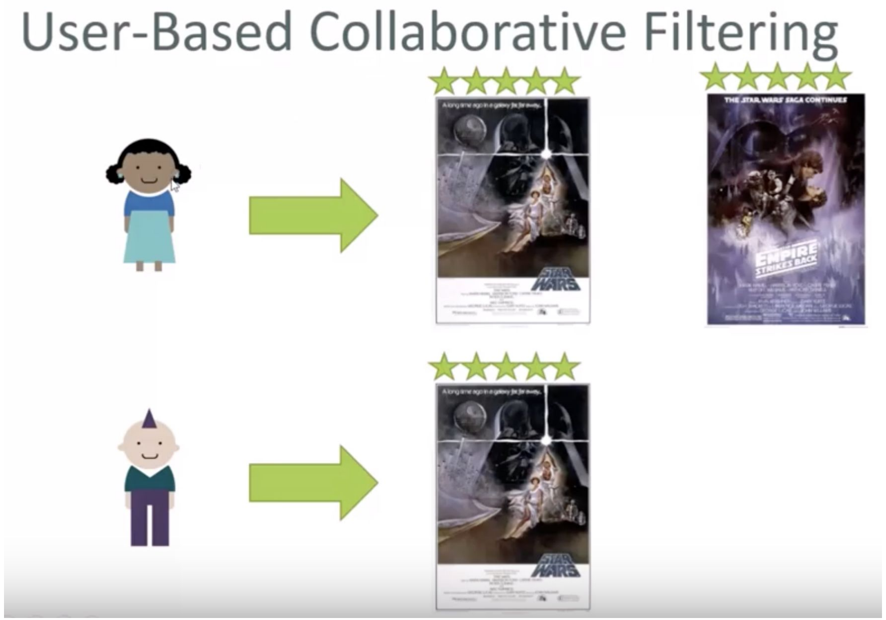
03. Collaborative Filtering (User-Based)

```python
from surprise import KNNBasic
```

```python
sim_options = {'name': 'cosine',
               'user_based': True
               }

model = KNNBasic(sim_options=sim_options)
model.fit(recommendation_dataset.train_set)
```

```
Mean Absolute Error: 0.754729310903918
Root Mean Square Error: 0.9786509473598148
Hit Rate (HR): 0.0
Cumulative Hit Rate (cHR): 0.0
Average Reciprocal HitRate  (ARHR): 0.0
Rating  HitRate  (rHR): {}
Coverage: 1.0
Diversity: 0.8882370462217049
Novelty: 6126.978196721311
```



- Budujemy macierz user/user
- Obliczamy simialirity score
- Znajdujemy podobnych użytkowników
- Rekomendujemy te które najbliżsi użytkownicy obejrzeli

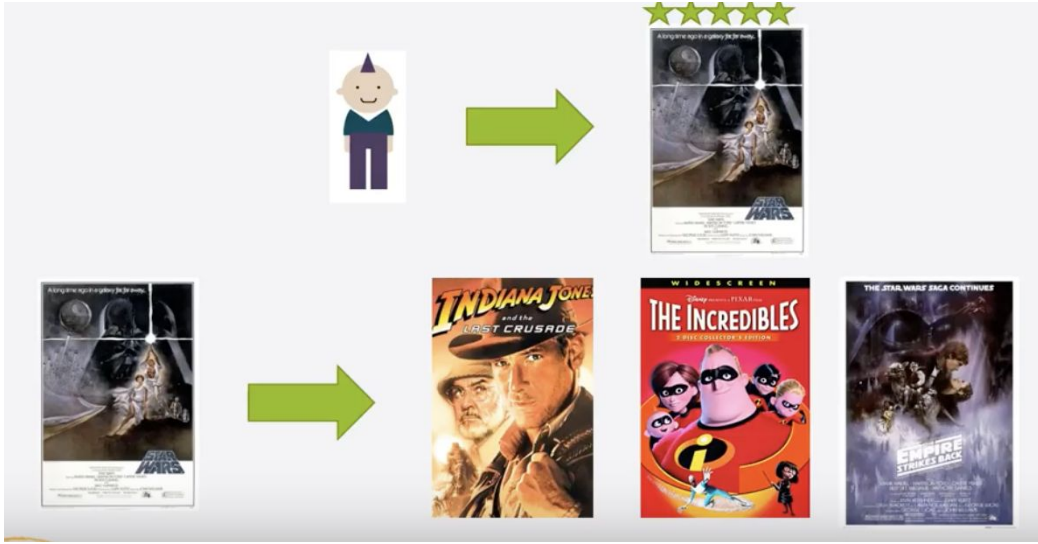
https://www.youtube.com/watch?v=Ty3Ui_mQtxY

03. Collaborative Filtering (Item-Based)

```python
sim_options = {'name': 'cosine',
               'user_based': False
               }

model_item = KNNBasic(sim_options=sim_options)
model_item.fit(recommendation_dataset.train_set)
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Mean Absolute Error: 0.7610169078515616
Root Mean Square Error: 0.9787736703737636
Hit Rate (HR): 0.0
Cumulative Hit Rate (cHR): 0.0
Average Reciprocal HitRate  (ARHR): 0.0
Rating  HitRate  (rHR): {}
Coverage: 0.9885245901639345
Diversity: 0.7204060469128939
Novelty: 6933.130047106326
```



- Budujemy macierz movie/movie
- Obliczamy simialirity score
- Znajdujemy najbardziej podobne filmy
- Rekomendujemy te które najbliżej tym które obejrzyliśmy

https://www.slideshare.net/xamat/recommender-systems-machine-learning-summer-school-2014-cmu

https://www.youtube.com/watch?v=8wLKuscyO9I

04. Algorytmy SVD

$$R = M\Sigma U^T$$

singular value decomposition (svd)

```
from surprise import SVD

svd = SVD(random_state=10)
svd.fit(recommendation_dataset.train_set)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7f7d51d75278>
```

```
%%time
print("\n\n", get_evaluation(svd, recommendation_dataset))
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Mean Absolute Error: 0.49840804714417075
Root Mean Square Error: 0.6436712599229663
Hit Rate (HR): 0.036065573770491806
Cumulative Hit Rate (cHR): 0.036065573770491806
Average Reciprocal HitRate  (ARHR): 0.013333333333333332
Rating  HitRate  (rHR): {2.5: 0.06666666666666667, 3.0: 0.008695652173913044, 4.0: 0.044444444444444446, 4.5
Coverage: 0.9245901639344263
Diversity: 0.03138572161157538
Novelty: 504.3873857062885


 {'MAE': 0.49840804714417075, 'RMSE': 0.6436712599229663, 'HR': 0.036065573770491806, 'cHR': 0.0360655737704
CPU times: user 2min 17s, sys: 2.88 s, total: 2min 20s
Wall time: 2min 20s
```

Podsumowanie

- Istnieje wiele algorytmów do ewaluacji algorytmów rekomendacji
- Nie zawsze warto patrzeć na MAE i RMSE, ponieważ rekomendacje nie są zadaniem polegającym na minimalizacji błędu, a bardziej na dopasowaniu najlepszego top-n
- Content based jest obiecującym algorytmem i można go użyć razem z kombinacją z innymi algorytmami
- SVD jest obecnie najbardziej popularnym algorytmem używanym przez wiele firm np: Netflix.

Kolejne kroki

- Użycie wykresów
- Przetestowanie algorytmów do filmweb-rekomendacje

https://github.com/dataworkshop/dw-poznan-project

Dziękuję