Practical implementation of the recommender system

Lara Komel (89191056)

For reading the ratings I implemented the class **UserItemData**, which saves the data frame given with its path, creates a new column ‘date’ from columns ‘date_day’, ‘date_month’ and ‘date_year’, creates new column ‘num_ratings’, where it counts how many times each movie was rated, and saves the desired dataframe with given parameters (from date, to date and min ratings) as self.df.

```
x = UserItemData("data/user_ratedmovies.dat", from_date = '12.1.2007',
to_date='16.2.2008',min_ratings=1000)
print(x.nratings())
y = UserItemData("data/user_ratedmovies.dat")
print(y.nratings())
>>22021
>>855598
```

Class **MovieData** returns the title of the movie given its id.

```
md = MovieData('data/movies.dat')
print(md.get_title(1))
>>Toy story
```

RandomPredictor class gives a random rating to every movie in the data frame and returns a dictionary.

```
md = MovieData('data/movies.dat')
uim = UserItemData('data/user_ratedmovies.dat')
rp = RandomPredictor(1, 5)
rp.fit(uim)
pred = rp.predict(78)
print(type(pred))
items = [1, 3, 20, 50, 100]
for item in items:
    print("Movie: {}, score: {}".format(md.get_title(item), pred[item]))
>> Movie: Toy story, score: 1
>> Movie: Grumpy Old Men, score: 1
>> Movie: Money Train, score: 4
>> Movie: The Usual Suspects, score: 2
>> Movie: City Hall, score: 2
```

Class **Recommender** takes as a parameter a predictor of our choice, and takes the rating predictions dictionary that the predict method of the chosen predictor returned, and modifies it based on our needs: first it sorts the predictions in a descending order, than it removes the ones the user has already seen, if that is what we specified in the parameters, and then in returns a dictionary of 'n' best predictions (first n predictions of the dictionary).

```
md = MovieData('data/movies.dat')
uim = UserItemData('data/user_ratedmovies.dat')
rp = RandomPredictor(1, 5)
rec = Recommender(rp)
rec.fit(uim)
rec_items = rec.recommend(78, n=7, rec_seen=False)
for idmovie, val in rec_items.items():
    print("Movie: {}, score: {}".format(md.get_title(idmovie), val))


>> Movie: Grumpy Old Men, score: 5
Movie: Die Hard: With a Vengeance, score: 5
Movie: The Abyss, score: 5
Movie: Death Becomes Her, score: 5
Movie: Hostage, score: 5
Movie: Star Wreck: In the Pirkinning, score: 5
Movie: Pirates of the Caribbean: Dead Man's Chest, score: 5
```

Its evaluate method calculates mae, rmse, precision, recall, accuracy, f1 according to their formulas.

The **averagePredictor** calculates the average for each movie with the formula avg = (vs + b * g_avg) / (n + b)， and returns a dictionary of movieIDs and their averages. Using it with the Recommender class, we would get recommended n movies with the highest averages.

```
uim = UserItemData('data/user_ratedmovies.dat')
av = AvaragePredictor(0)
av_pr = av.fit(uim, 10, 78, False)
for idmovie, val in av_pr.items():
    print("Movie: {}, score: {}".format(md.get_title(idmovie), val))


>> Movie: Brother Minister: The Assassination of Malcolm X, score: 5.0
Movie: Synthetic Pleasures, score: 5.0
Movie: Adam & Steve, score: 5.0
Movie: Gabbeh, score: 5.0
Movie: Eve and the Fire Horse, score: 5.0
Movie: Saikaku ichidai onna, score: 5.0
Movie: Maradona by Kusturica, score: 5.0
Movie: The Princess and the Cobbler, score: 5.0
Movie: Vals Im Bashir, score: 5.0
Movie: Ko to tamo peva, score: 5.0
```

The **ViewsPredictor** counts how many times has each movie in the data frame been reviewed, and returns a dictionary with movieIDs and the counts. Using it with the recommender class, we would get n number of movies, that were reviewed (watched) the highest amount of times.

```
md = MovieData('data/movies.dat')
uim = UserItemData('data/user_ratedmovies.dat')
vp = ViewsPredictor()
vp.fit(uim)
most_watched = vp.predict(78, False)
for idmovie, val in most_watched.items():
    print("Movie: {}, score: {}".format(md.get_title(idmovie), val))

>> Movie: The Lord of the Rings: The Fellowship of the Ring, score: 1576
Movie: The Lord of the Rings: The Two Towers, score: 1528
Movie: The Lord of the Rings: The Return of the King, score: 1457
Movie: The Silence of the Lambs, score: 1431
Movie: Shrek, score: 1404
```

The **ItemBasedPredictor** class predicts the values using similarity between movies. It compares every movie with every movie that the user has already rated, and using the formula for predictions predicts what the rating should be.
The similarity method uses cosine distance to calculate how similar two movies are. For every user it checks whether they have rated both movies we are interested in, and if they have it uses their ratings in the formula for cosine distance.
The similarItems method calculates the similarity between given movie and every other movie in the table and returns n most similar ones.

```
print("Similarity between the movies 'Men in black'(1580) and 'Ghostbusters'(2716): ",
rp.similarity(1580, 2716))
print("Similarity between the movies 'Men in black'(1580) and 'Schindler's List'(527):
", rp.similarity(1580, 527))
print("Similarity between the movies 'Men in black'(1580) and 'Independence day'(780):
", rp.similarity(1580, 780))
>> Similarity between the movies 'Men in black'(1580) and 'Ghostbusters'(2716):
0.2339552317675661
Similarity between the movies 'Men in black'(1580) and 'Schindler's List'(527):  0
Similarity between the movies 'Men in black'(1580) and 'Independence day'(780):
0.42466125844687547

md = MovieData('data/movies.dat')
uim = UserItemData('data/user_ratedmovies.dat', min_ratings=1300)
rp = ItemBasedPredictor()
rec = Recommender(rp)
rec.fit(uim)
print("Predictions for 78: ")
```

```
rec_items = rec.recommend(78, n=15, rec_seen=False)
for idmovie, val in rec_items.items():
    print("Movie: {}, score: {}".format(md.get_title(idmovie), val))

>> Predictions for 78:
Movie: The Silence of the Lambs, score: 4.295443265242239
Movie: The Lord of the Rings: The Two Towers, score: 3.892857142857143
Movie: The Lord of the Rings: The Return of the King, score: 3.892857142857143
Movie: The Lord of the Rings: The Fellowship of the Ring, score: 3.6164546725232953
Movie: Men in Black, score: 2.741750751346995
Movie: Shrek, score: 2.690269582045972
Movie: Gladiator, score: 2.6338631132952295
Movie: Pirates of the Caribbean: The Curse of the Black Pearl, score:
2.4999999999999996
```

The reccMyself method combines the old data frame with one of my movie ratings, and with one predictor (this case random one) recommends movies for me.

The most_similar method calculates similarity between each pair of movies and returns ones with highest similarity.

```
md = MovieData('data/movies.dat')
uim = UserItemData('data/user_ratedmovies.dat', min_ratings=1300)
rp = ItemBasedPredictor()
rec = Recommender(rp)
rec.fit(uim)
rec_items = rp.similarItems(4993, 5)
print('Movies similar to "The Lord of the Rings: The Fellowship of the Ring": ')
for idmovie, val in rec_items.items():
    print("Movie: {}, score: {}".format(md.get_title(idmovie), val))
>> Movies similar to "The Lord of the Rings: The Fellowship of the Ring":
Movie: The Lord of the Rings: The Two Towers, score: 0.7579414271936223
Movie: The Lord of the Rings: The Return of the King, score: 0.7344850182581967
Movie: Star Wars: Episode V - The Empire Strikes Back, score: 0.020951961999877797
Movie: Star Wars, score: 0.0063615789285414006
Movie: Pulp Fiction, score: 0
```

The **SlopeOnePredictor** class predicts the rating using slope one method using deviation between all movies and movies rated by the user. The div method first checks whether the user rated both movies and if they did calculates the deviation with the formula.

```
md = MovieData('data/movies.dat')
uim = UserItemData('data/user_ratedmovies.dat', min_ratings=1300)
```

```
rp = SlopeOnePredictor()
rec = Recommender(rp)
rec.fit(uim)
print("Predictions for 78: ")
rec_items = rec.recommend(78, n=5, rec_seen=False)
for idmovie, val in rec_items.items():
    print("Movie: {}, score: {}".format(md.get_title(idmovie), val))
>> Predictions for 78:
Movie: The Lord of the Rings: The Fellowship of the Ring, score: 4.014217872241849
Movie: The Lord of the Rings: The Return of the King, score: 4.013807322254155
Movie: The Silence of the Lambs, score: 3.9833508047111055
Movie: The Lord of the Rings: The Two Towers, score: 3.945820061301761
Movie: Gladiator, score: 3.712324245824807
```