cf-03-rzepinskip-results-analysis

January 21, 2019

1 Important

`make scores` has to be run before running any notebook cell

2 Imports

```
In [1]: import pandas as pd
        import seaborn as sns
```

3 Visualization settings

```
In [2]: sns.set(context='paper', font_scale=1.2, style='ticks', palette='muted',
                rc={"axes.labelsize":16, "ytick.labelsize": 14, "xtick.labelsize":14,
                    "font.family": "sans-serif"})
```

4 Accuracy

4.1 Results

```
In [3]: df_accuracy = pd.read_csv("../results/cf-accuracy-results.csv", index_col='model')

In [4]: df_accuracy.sort_values('rmse')
```

Out[4]:

model	rmse	mae	fcp
knn-model.pkl	0.793526	0.600802	0.706112
svd-model.pkl	0.809727	0.619696	0.685492
slopeone-model.pkl	0.847128	0.658539	0.646655

4.2 Distribution

```
In [5]: df_pred_so = pd.read_csv("../models/predictions/cf-results/testset/slopeone-testset-
        predictions.csv")
        df_pred_knn = pd.read_csv("../models/predictions/cf-results/testset/knn-testset-
        predictions.csv")
        df_pred_svd = pd.read_csv("../models/predictions/cf-results/testset/svd-testset-
        predictions.csv")

In [6]: df_pred_so['err'] = abs(df_pred_so.est - df_pred_so.rating)
        df_pred_knn['err'] = abs(df_pred_knn.est - df_pred_knn.rating)
        df_pred_svd['err'] = abs(df_pred_svd.est - df_pred_svd.rating)
```

As the KNN and SVD results are very similar, the SlopeOne and KNN results are compared.

4.3 Worst and best scenarios

```
In [7]: df_pred_so.sort_values('err').tail()
```

Out[7]:

	user_id	book_id	rating	est	details	err
577810	51577	802	5.0	1.0	{'was_impossible': False}	4.0
581339	51905	7847	1.0	5.0	{'was_impossible': False}	4.0
412375	36271	4340	1.0	5.0	{'was_impossible': False}	4.0
484598	42878	176	1.0	5.0	{'was_impossible': False}	4.0
538834	47913	39	1.0	5.0	{'was_impossible': False}	4.0

```
In [8]: df_pred_svd.sort_values('err').tail()
```

Out[8]:

	user_id	book_id	rating	est	details	err
174075	14997	50	1.0	5.0	{'was_impossible': False}	4.0
119629	10258	51	1.0	5.0	{'was_impossible': False}	4.0
494743	43802	1616	1.0	5.0	{'was_impossible': False}	4.0
463993	40993	3309	1.0	5.0	{'was_impossible': False}	4.0
484598	42878	176	1.0	5.0	{'was_impossible': False}	4.0

```
In [9]: df_pred_knn[df_pred_so.err >= 3.5].head()
```

Out[9]:

	user_id	book_id	rating	est	details	err
3108	290	47	1.0	4.714256	{'actual_k': 30, 'was_impossible': False}	3.714256
9387	831	886	1.0	4.629693	{'actual_k': 30, 'was_impossible': False}	3.629693
17734	1560	179	1.0	4.532832	{'actual_k': 30, 'was_impossible': False}	3.532832
25803	2239	104	1.0	4.835458	{'actual_k': 30, 'was_impossible': False}	3.835458
42189	3641	38	1.0	4.285513	{'actual_k': 30, 'was_impossible': False}	3.285513

```
In [10]: df_pred_knn[df_pred_knn.err >= 3.5].sort_values('err', ascending=False).head()
```

Out[10]:

	user_id	book_id	rating	est	details	err
577805	51577	342	5.0	1.0	{'actual_k': 30, 'was_impossible': False}	4.0
577810	51577	802	5.0	1.0	{'actual_k': 30, 'was_impossible': False}	4.0
345274	30126	257	1.0	5.0	{'actual_k': 30, 'was_impossible': False}	4.0
341365	29771	1	1.0	5.0	{'actual_k': 30, 'was_impossible': False}	4.0
442434	39020	2950	1.0	5.0	{'actual_k': 30, 'was_impossible': False}	4.0

```
In [11]: df_pred_so[df_pred_knn.err >= 3.5].head()
```

Out[11]:

	user_id	book_id	rating	est	details	err
3108	290	47	1.0	4.617554	{'was_impossible': False}	3.617554
9387	831	886	1.0	4.503122	{'was_impossible': False}	3.503122
15841	1397	354	1.0	4.358887	{'was_impossible': False}	3.358887
17734	1560	179	1.0	4.709507	{'was_impossible': False}	3.709507
25803	2239	104	1.0	4.589768	{'was_impossible': False}	3.589768

4.4 Estimates distributions

```
In [12]: df_pred_so.est.describe()
```

```
Out[12]: count    597648.000000
         mean          3.916693
         std           0.522656
         min           1.000000
         25%           3.573050
         50%           3.922227
         75%           4.274658
         max           5.000000
         Name: est, dtype: float64
```

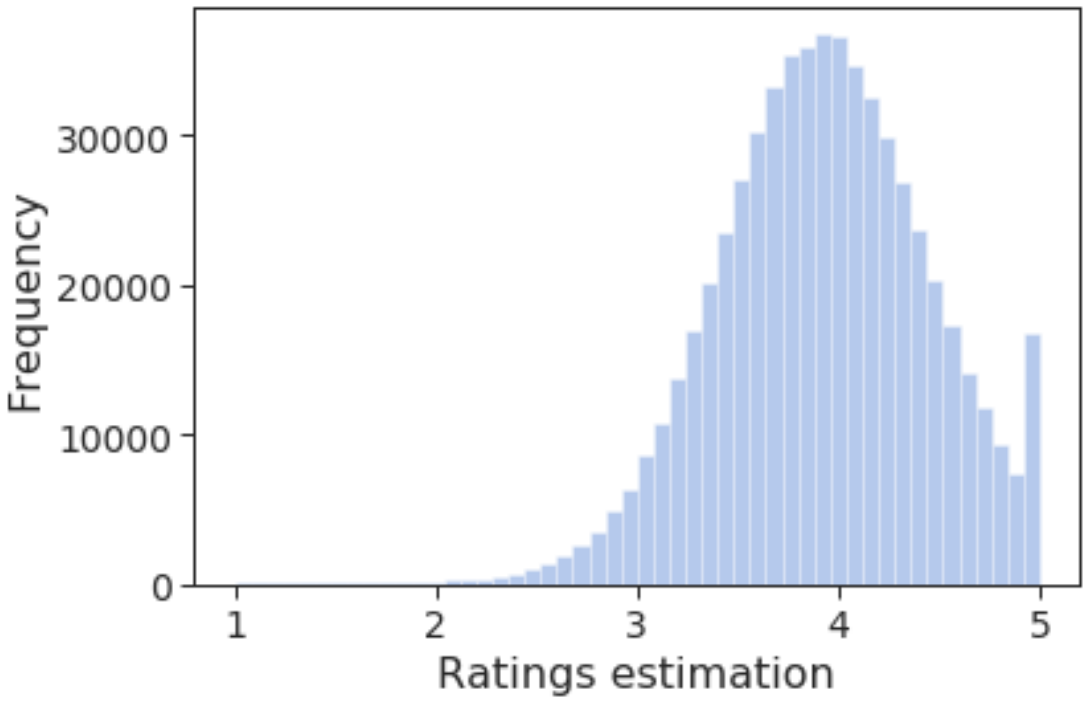
```
In [13]: df_pred_knn.est.describe()
```

```
Out[13]: count    597648.000000
         mean          3.951805
         std           0.581040
         min           1.000000
         25%           3.574497
         50%           3.972945
         75%           4.365363
         max           5.000000
         Name: est, dtype: float64
```

```
In [14]: so_dist_plot = sns.distplot(df_pred_so.est, kde=False)
         so_dist_plot.set(xlabel='Ratings estimation', ylabel='Frequency')
```

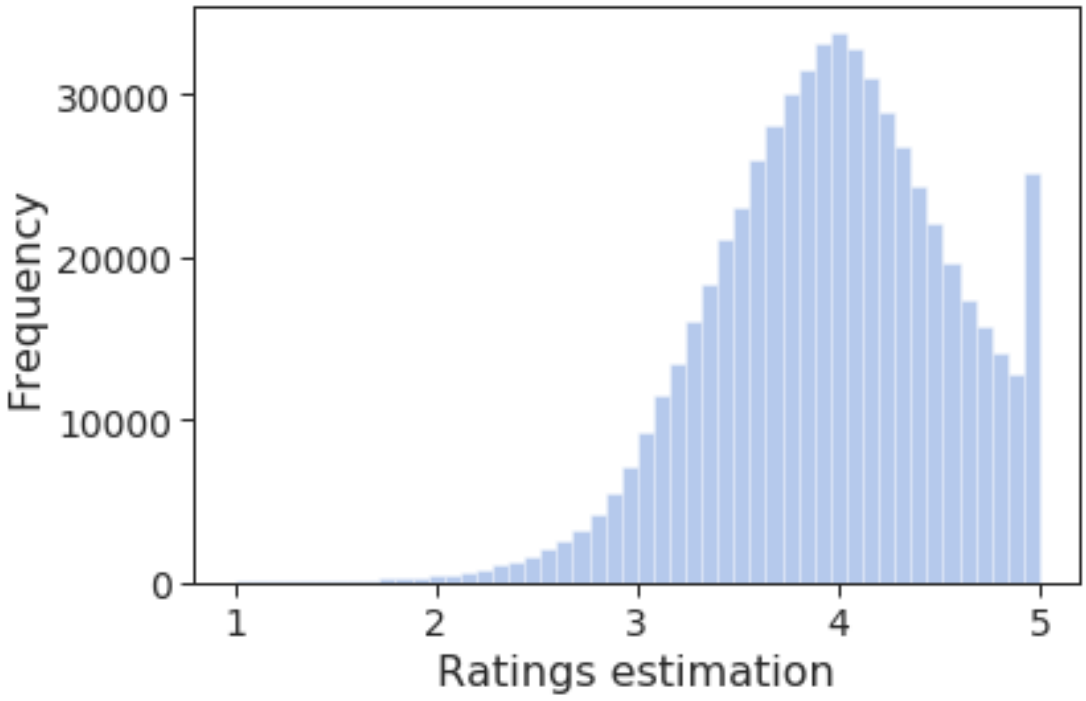
```
/home/rzepinskip/Documents/Inzynierka/Recommendation-system/rs-
venv/lib/python3.7/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a
non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]`
instead of `arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

```
Out[14]: [Text(0, 0.5, 'Frequency'), Text(0.5, 0, 'Ratings estimation')]
```




```
In [15]: knn_dist_plot = sns.distplot(df_pred_knn.est, kde=False)
         knn_dist_plot.set(xlabel='Ratings estimation', ylabel='Frequency')
```

```
Out[15]: [Text(0, 0.5, 'Frequency'), Text(0.5, 0, 'Ratings estimation')]
```



```
In [16]: df_pred_so.err.describe()
```

```
Out[16]: count    597648.000000
         mean          0.658539
         std           0.532872
         min           0.000000
         25%           0.253293
         50%           0.549107
         75%           0.934457
         max           4.000000
         Name: err, dtype: float64
```

```
In [17]: df_pred_knn.err.describe()
```

```
Out[17]: count    597648.000000
         mean          0.600802
         std           0.518383
         min           0.000000
         25%           0.209945
         50%           0.478377
         75%           0.854820
         max           4.000000
         Name: err, dtype: float64
```

4.5 Neighbors requirement

```
In [18]: k_vals = df_pred_knn['details'].apply(lambda x : dict(eval(x))).apply(pd.Series)
```

```
In [19]: df_pred_knn_full = pd.merge(df_pred_knn, k_vals, left_index=True, right_index=True)
```

```
In [20]: df_pred_knn_full.head()
```

```
Out[20]:    user_id  book_id  rating       est  \
         0        1       11     5.0  3.661642
         1        1       13     4.0  3.912217
         2        1       40     2.0  2.898356
         3        1       66     4.0  3.734994
         4        1      102     5.0  3.748537

                                          details       err  actual_k  \
         0  {'actual_k': 30, 'was_impossible': False}  1.338358        30
         1  {'actual_k': 30, 'was_impossible': False}  0.087783        30
         2  {'actual_k': 30, 'was_impossible': False}  0.898356        30
         3  {'actual_k': 30, 'was_impossible': False}  0.265006        30
         4  {'actual_k': 30, 'was_impossible': False}  1.251463        30

            was_impossible
         0           False
```

```
1           False
2           False
3           False
4           False
```

```
In [21]: k_vals['actual_k'].describe()
```

```
Out[21]: count    597648.000000
         mean         29.510687
         std           2.201143
         min           0.000000
         25%          30.000000
         50%          30.000000
         75%          30.000000
         max          30.000000
         Name: actual_k, dtype: float64
```

```
In [22]: k_vals[k_vals < 10].count() / len(k_vals)
```

```
Out[22]: actual_k          0.001211
         was_impossible    1.000000
         dtype: float64
```

```
In [23]: df_pred_knn_full[df_pred_knn_full.err >= 3].head(1000).actual_k.describe()
```

```
Out[23]: count    1000.000000
         mean       29.008000
         std         3.315559
         min         5.000000
         25%        30.000000
         50%        30.000000
         75%        30.000000
         max        30.000000
         Name: actual_k, dtype: float64
```

5 Effectiveness

```
In [24]: df_eff = pd.read_csv("../results/cf-effectiveness-results-n.csv", index_col='model')
```

```
In [25]: df_eff[df_eff.n == 20]
```

```
Out[25]:                            n  precision-to_read  precision-testset  \
         model
         knn-predictions.csv       20           0.004129           0.005537
         slopeone-predictions.csv  20           0.002108           0.001966
         svd-predictions.csv       20           0.006798           0.011364

                                   recall-to_read  recall-testset
         model
         knn-predictions.csv              0.004862        0.009691
         slopeone-predictions.csv         0.002213        0.003578
         svd-predictions.csv              0.007694        0.019977
```

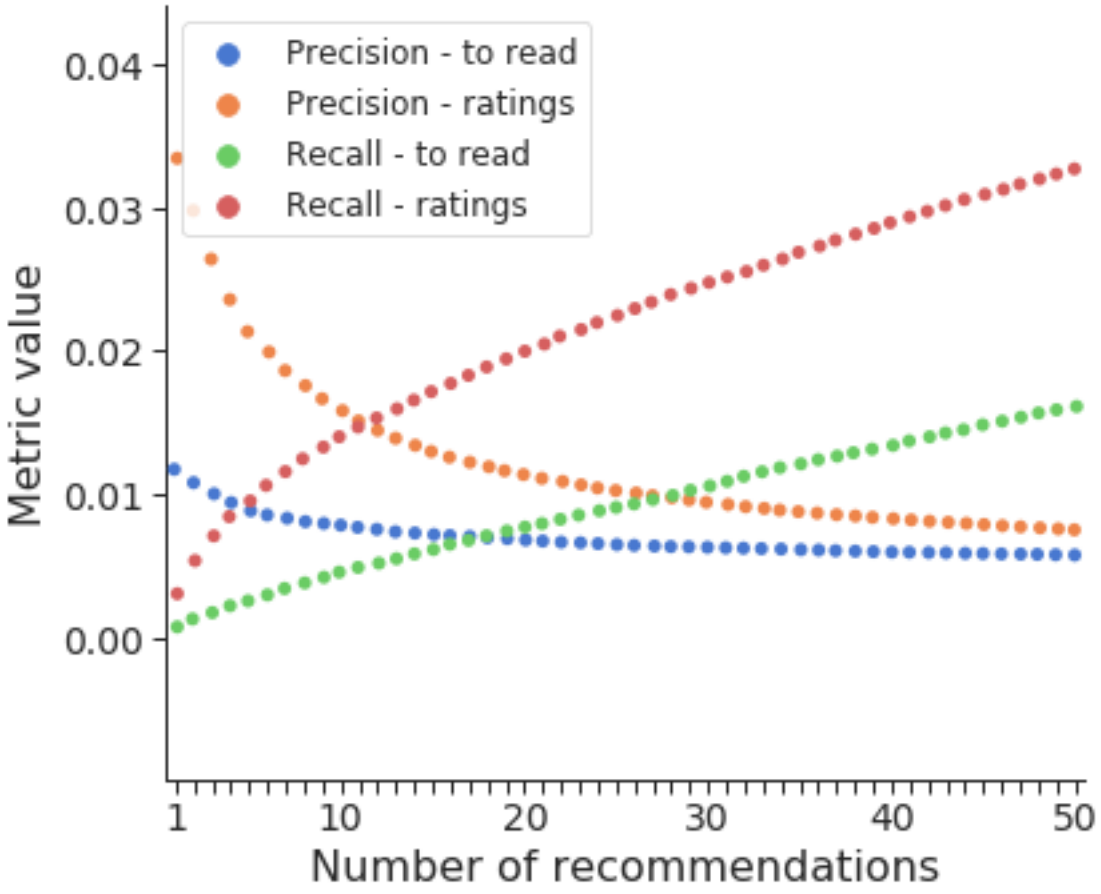
In [26]:
```
eff_over_n = df_eff[df_eff.index == 'svd-predictions.csv'].melt('n', var_name='cols',
value_name='vals')
```

In [27]:
```
import matplotlib.pyplot as plt
g = sns.catplot(x="n", y="vals", hue='cols', data=eff_over_n, aspect=1.2, legend=False)

for ax in g.axes.flat:
    labels = ax.get_xticklabels()
    for i,l in enumerate(labels):
        if((i+1)%10 != 0 and i != 0): labels[i] = ''
    ax.set_xticklabels(labels)

new_labels = ['Precision - to read', 'Precision - ratings', 'Recall - to read', 'Recall
- ratings']
g.set(xlabel ='Number of recommendations', ylabel ='Metric value')
g.ax.legend(loc=0)
handles, labels = ax.get_legend_handles_labels()
g.ax.legend(loc=2, handles=handles, labels=new_labels, fontsize='12')
```

Out[27]: <matplotlib.legend.Legend at 0x7f73544d3320>



In [ ]: