Comparison of two algorithms

We will see in this notebook how we can compare the prediction accuracy of two algorithms.

In [1]:

```python
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
import pickle
import os

import pandas as pd

from surprise import SVD
from surprise import KNNBasic
from surprise import Dataset
from surprise import Reader
from surprise import dump
from surprise.accuracy import rmse
```

In [2]:

```python
# We will train and test on the u1.base and u1.test files of the movielens-100k dataset.
# if you haven't already, you need to download the movielens-100k dataset
# You can do it manually, or by running:

# Dataset.load_builtin('ml-100k')

# Now, let's load the dataset
train_file = os.path.expanduser(
    '~') + '/.surprise_data/ml-100k/ml-100k/u1.base'
test_file = os.path.expanduser('~') + '/.surprise_data/ml-100k/ml-100k/u1.test'
data = Dataset.load_from_folds([(train_file, test_file)], Reader('ml-100k'))

# We'll use the well-known SVD algorithm and a basic nearest neighbors approach.
algo_svd = SVD()
algo_knn = KNNBasic()

for trainset, testset in data.folds():
    algo_svd.train(trainset)
    predictions_svd = algo_svd.test(testset)

    algo_knn.train(trainset)
    predictions_knn = algo_knn.test(testset)

    rmse(predictions_svd)
    rmse(predictions_knn)

    dump.dump('./dump_SVD', predictions_svd, algo_svd)
    dump.dump('./dump_KNN', predictions_knn, algo_knn)
```

```
/home/user/workspace/.venv/lib/python3.6/site-
```

```
/home/user/workspace/.venv/lib/python3.6/site-
packages/surprise/dataset.py:193: UserWarning: Using data.split() or using
load_from_folds() without using a CV iterator is now deprecated.
  UserWarning)
/home/user/workspace/.venv/lib/python3.6/site-
packages/surprise/prediction_algorithms/algo_base.py:51: UserWarning: train
() is deprecated. Use fit() instead
  warnings.warn('train() is deprecated. Use fit() instead', UserWarning)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.9533
RMSE: 0.9889
```

In [3]:

```
# The dumps have been saved and we can now use them whenever we want.

predictions_svd, algo_svd = dump.load('./dump_SVD')
predictions_knn, algo_knn = dump.load('./dump_KNN')

df_svd = pd.DataFrame(predictions_svd,
                      columns=['uid', 'iid', 'rui', 'est', 'details'])
df_knn = pd.DataFrame(predictions_knn,
                      columns=['uid', 'iid', 'rui', 'est', 'details'])

df_svd['err'] = abs(df_svd.est - df_svd.rui)
df_knn['err'] = abs(df_knn.est - df_knn.rui)
```

We now have two dataframes with the all the predictions for each algorithm. The cool thing is that, as both algorithm have been tested on the same testset, the indexes of the two dataframes are the same!

In [4]:

```
df_svd.head()
```

Out[4]:

	uid	iid	rui	est	details	err
0	1	6	5.0	3.755857	{'was_impossible': False}	1.244143
1	1	10	3.0	3.807071	{'was_impossible': False}	0.807071
2	1	12	5.0	4.462403	{'was_impossible': False}	0.537597
3	1	14	5.0	3.675166	{'was_impossible': False}	1.324834
4	1	17	3.0	3.284802	{'was_impossible': False}	0.284802

In [5]:

```
df_knn.head()
```

Out[5]:

	uid	iid	rui	est	details	err
0	1	6	5.0	3.468613	{'actual_k': 20, 'was_impossible': False}	1.531387
1	1	10	3.0	3.866290	{'actual_k': 40, 'was_impossible': False}	0.866290
2	1	12	5.0	4.538194	{'actual_k': 40, 'was_impossible': False}	0.461806
3	1	14	5.0	4.235741	{'actual_k': 40, 'was_impossible': False}	0.764259
4	1	17	3.0	3.228002	{'actual_k': 40, 'was_impossible': False}	0.228002

In [6]:

```
# Let's check how good are the KNN predictions when the SVD has a huge
error:
df_knn[df_svd.err >= 3.5]
```

Out[6]:

	uid	iid	rui	est	details	err
1905	38	211	1.0	4.136955	{'actual_k': 40, 'was_impossible': False}	3.136955
1925	38	432	1.0	4.064878	{'actual_k': 40, 'was_impossible': False}	3.064878
1930	38	526	1.0	4.115078	{'actual_k': 40, 'was_impossible': False}	3.115078
6512	141	1142	1.0	4.126349	{'actual_k': 34, 'was_impossible': False}	3.126349
7390	167	169	1.0	4.664991	{'actual_k': 40, 'was_impossible': False}	3.664991
7861	181	25	5.0	3.173767	{'actual_k': 40, 'was_impossible': False}	1.826233
13972	295	183	1.0	4.202611	{'actual_k': 40, 'was_impossible': False}	3.202611
15306	312	265	1.0	4.131875	{'actual_k': 40, 'was_impossible': False}	3.131875
19140	405	575	5.0	2.410506	{'actual_k': 36, 'was_impossible': False}	2.589494

In [7]:

```
# Well... Not much better.
# Now, let's look at the predictions of SVD on the 10 worst predictions for
KNN
df_svd.iloc[df_knn.sort_values(by='err')[-10:].index]
```

Out[7]:

	uid	iid	rui	est	details	err
9406	208	302	1.0	4.258081	{'was_impossible':	3.258081

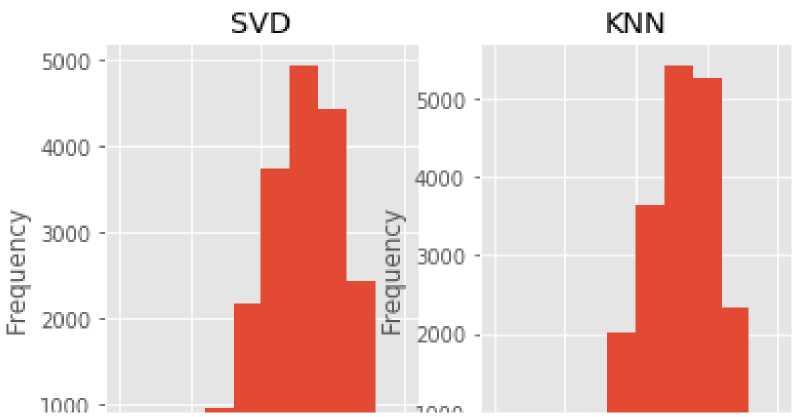
	uid	iid	rui	est	details	err
19089	405	169	1.0	2.843449	{'was_impossible': False}	1.843449
19785	436	132	1.0	4.493084	{'was_impossible': False}	3.493084
157	2	315	1.0	4.253176	{'was_impossible': False}	3.253176
8503	193	56	1.0	3.768274	{'was_impossible': False}	2.768274
5531	113	976	5.0	2.961022	{'was_impossible': False}	2.038978
7917	181	408	1.0	2.928082	{'was_impossible': False}	1.928082
7390	167	169	1.0	4.835424	{'was_impossible': False}	3.835424
7412	167	1306	5.0	3.751609	{'was_impossible': False}	1.248391
5553	114	1104	5.0	2.912709	{'was_impossible': False}	2.087291

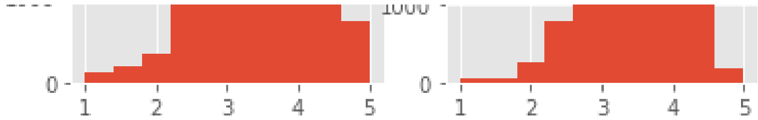
In [8]:

```
# How different are the predictions from both algorithms ?
# Let's count the number of predictions for each rating value

import matplotlib.pyplot as plt
import matplotlib
%matplotlib notebook
%matplotlib inline
matplotlib.style.use('ggplot')

figure, (ax1, ax2) = plt.subplots(1, 2)

df_svd.est.plot(kind='hist', title='SVD', ax=ax1)
df_knn.est.plot(kind='hist', title='KNN', ax=ax2)

# As expected, one of the drawbacks of the NN algorithms is that their pred
ictions are often
# quite concentrated around the mean. The SVD algorithm seems more
confortable predicting extreme rating values.
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2a2e1a4fd0>
```





In [9]:

```
# Question: when a user has rated only a small number of items (less than 1
0), which algorithm
# gives the best predictions on average?


def get_Iu(uid):
    """Return the number of items rated by given user

    Args:
        uid: The raw id of the user.
    Returns:
        The number of items rated by the user.
    """

    try:
        return len(trainset.ur[trainset.to_inner_uid(uid)])
    except ValueError:  # user was not part of the trainset
        return 0


df_knn['Iu'] = df_knn.uid.apply(get_Iu)
df_svd['Iu'] = df_svd.uid.apply(get_Iu)

df_knn[df_knn.Iu < 10].err.mean(), df_svd[df_svd.Iu < 10].err.mean()
```

Out[9]:

```
(1.0382962702232326, 0.994751228782901)
```