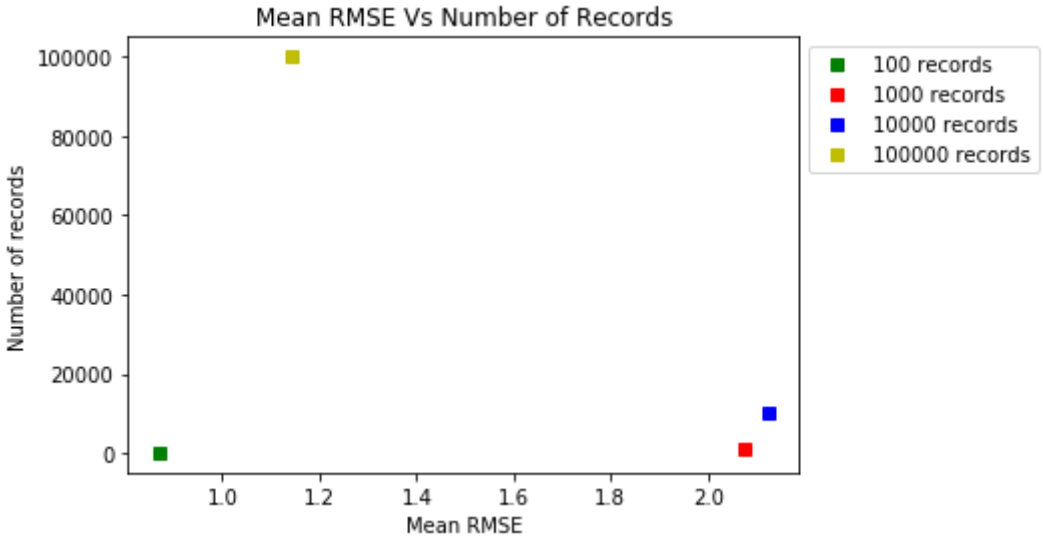
```
Python 3.6.1 |Anaconda custom (64-bit)| (default, May 11 2017, 13:25:24) [MSC v.1900 64 bit
(AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

Restarting kernel...
```

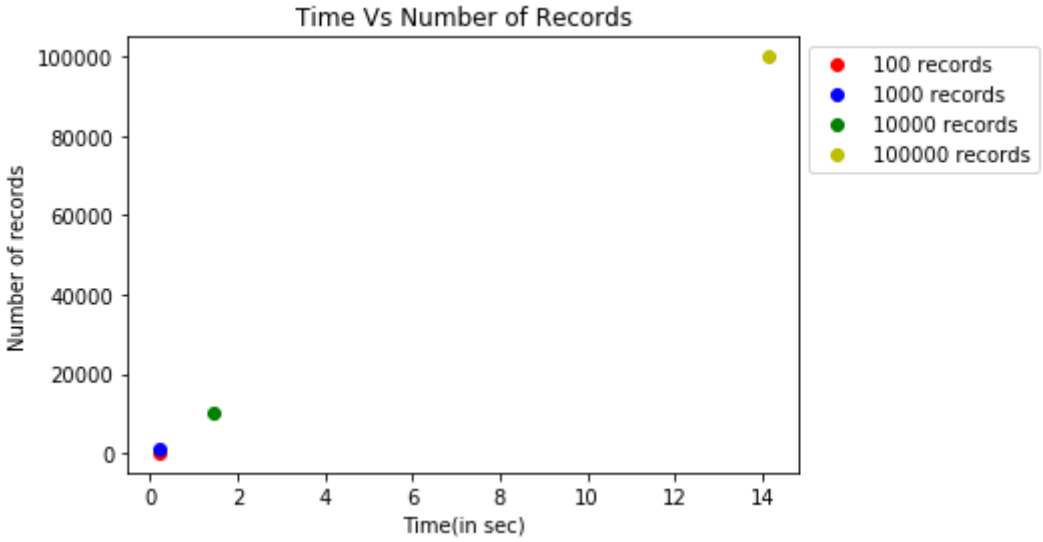
```
In [1]: import numpy as np
   ...: import surprise
   ...: import pandas as pd
   ...: from surprise import Reader
   ...: from surprise import Dataset
   ...: import time
   ...: import matplotlib.pyplot as plt
   ...: import psutil

In [2]: class MatrixFacto(surprise.AlgoBase):
   ...:     '''A basic rating prediction algorithm based on matrix factorization.'''
   ...:     skip_train=0
   ...:
   ...:     def __init__(self, learning_rate, n_epochs, n_factors):
   ...:
   ...:         self.lr = learning_rate  # learning rate for SGD
   ...:         self.n_epochs = n_epochs  # number of iterations of SGD
   ...:         self.n_factors = n_factors  # number of factors
   ...:
   ...:     def train(self, trainset):
   ...:         '''Learn the vectors p_u and q_i with SGD'''
   ...:
   ...:         print('Fitting data with SGD...')
   ...:
   ...:         # Randomly initialize the user and item factors.
   ...:         p = np.random.normal(0, .1, (trainset.n_users, self.n_factors))
   ...:         q = np.random.normal(0, .1, (trainset.n_items, self.n_factors))
   ...:
   ...:         # SGD procedure
   ...:         for _ in range(self.n_epochs):
   ...:             for u, i, r_ui in trainset.all_ratings():
   ...:                 err = r_ui - np.dot(p[u], q[i])
   ...:                 # Update vectors p_u and q_i
   ...:                 p[u] += self.lr * err * q[i]
   ...:                 q[i] += self.lr * err * p[u]
   ...:                 # Note: in the update of q_i, we should actually use the previous (non-
updated) value of p_u.
   ...:                 # In practice it makes almost no difference.
   ...:
   ...:         self.p, self.q = p, q
   ...:         self.trainset = trainset
   ...:
   ...:     def estimate(self, u, i):
   ...:         '''Return the estmimated rating of user u for item i.'''
   ...:
   ...:         # return scalar product between p_u and q_i if user and item are known,
   ...:         # else return the average of all ratings
   ...:         if self.trainset.knows_user(u) and self.trainset.knows_item(i):
   ...:             return np.dot(self.p[u], self.q[i])
   ...:         else:
   ...:             return self.trainset.global_mean

In [3]: timex=[]
   ...: mem=[]
```

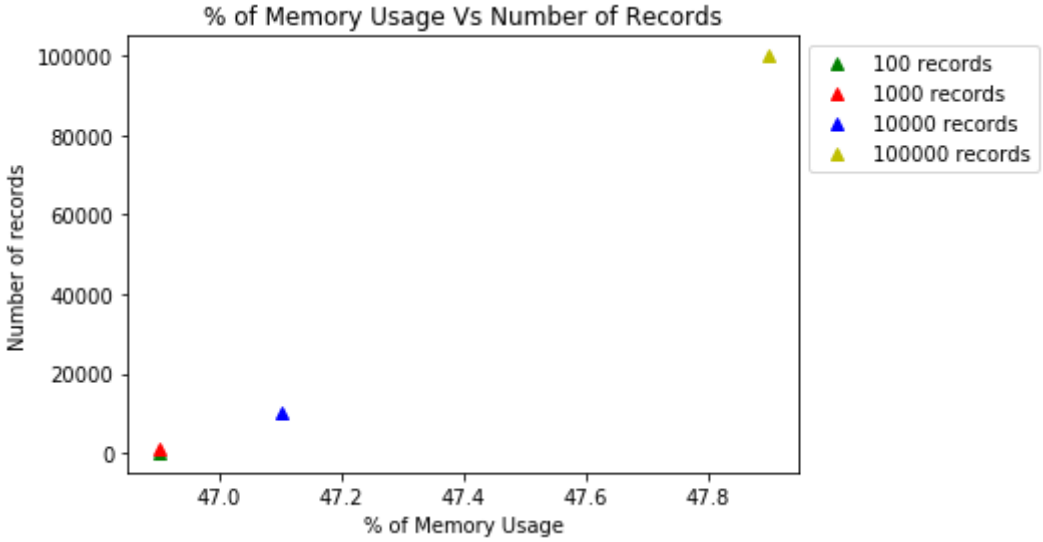
```
   ...: m1=psutil.virtual_memory().percent
   ...:
   ...: #For 100 record dataset
   ...: start = time.time()
   ...: df1 = pd.read_csv('C:/Users/dell pc/Desktop/Project/ratings_1million1.csv', dtype=
{'rating': float})
   ...: reader = Reader(rating_scale=(1, 5))
   ...: data = Dataset.load_from_df(df1[['user_id','book_id','rating']], reader)
   ...: data.split(2)
   ...: algo = MatrixFacto(learning_rate=.01, n_epochs=10, n_factors=10)
   ...: result1 = surprise.evaluate(algo, data, measures=['RMSE'])
   ...: end = time.time()
   ...: print("Time1",end - start)
   ...: timex.append(end-start)
   ...: m2=psutil.virtual_memory().percent
   ...: #print(m2)
   ...: mem.append(m2)
   ...:
   ...:
   ...: #For 1000 record dataset
   ...: start = time.time()
   ...: df2 = pd.read_csv('C:/Users/dell pc/Desktop/Project/ratings_1million2.csv', dtype=
{'rating': float})
   ...: reader = Reader(rating_scale=(1, 5))
   ...: data = Dataset.load_from_df(df2[['user_id','book_id','rating']], reader)
   ...: data.split(2)
   ...: algo = MatrixFacto(learning_rate=.01, n_epochs=10, n_factors=10)
   ...: result2 = surprise.evaluate(algo, data, measures=['RMSE'])
   ...: end = time.time()
   ...: print("Time2",end - start)
   ...: timex.append(end-start)
   ...: m3=psutil.virtual_memory().percent
   ...: #print(m2)
   ...: mem.append(m3)
   ...:
   ...: #For 10000 record dataset
   ...: start = time.time()
   ...: df3 = pd.read_csv('C:/Users/dell pc/Desktop/Project/ratings_1million3.csv', dtype=
{'rating': float})
   ...: reader = Reader(rating_scale=(1, 5))
   ...: data = Dataset.load_from_df(df3[['user_id','book_id','rating']], reader)
   ...: data.split(2)
   ...: algo = MatrixFacto(learning_rate=.01, n_epochs=10, n_factors=10)
   ...: result3 = surprise.evaluate(algo, data, measures=['RMSE'])
   ...: end = time.time()
   ...: print("Time3",end - start)
   ...: timex.append(end-start)
   ...: m4=psutil.virtual_memory().percent
   ...: #print(m2)
   ...: mem.append(m4)
   ...:
   ...: #For 100000 record dataset
   ...: start = time.time()
   ...: df4 = pd.read_csv('C:/Users/dell pc/Desktop/Project/ratings_1million4.csv', dtype=
{'rating': float})
   ...: reader = Reader(rating_scale=(1, 5))
   ...: data = Dataset.load_from_df(df4[['user_id','book_id','rating']], reader)
   ...: data.split(2)
   ...: algo = MatrixFacto(learning_rate=.01, n_epochs=10, n_factors=10)
   ...: result4 = surprise.evaluate(algo, data, measures=['RMSE'])
   ...: end = time.time()
   ...: print("Time4",end - start)
   ...: timex.append(end-start)
   ...: m5=psutil.virtual_memory().percent
   ...: #print(m2)
   ...: mem.append(m5)
   ...:
   ...:
   ...: #Plotting the Mean RMSE Vs Number of Records
```

```
   ...: y = [len(df1),len(df2),len(df3),len(df4)]
   ...: plt.plot( np.mean(result1['rmse']),y[0],'gs',label='100 records')
   ...: plt.plot( np.mean(result2['rmse']),y[1],'rs',label='1000 records')
   ...: plt.plot( np.mean(result3['rmse']),y[2],'bs',label='10000 records')
   ...: plt.plot( np.mean(result4['rmse']),y[3],'ys',label='100000 records')
   ...: legend = plt.legend(loc='upper left',bbox_to_anchor=(1, 1))
   ...: frame = legend.get_frame()
   ...: plt.xlabel('Mean RMSE')
   ...: plt.ylabel('Number of records')
   ...: plt.title('Mean RMSE Vs Number of Records')
   ...: plt.show()
   ...:
   ...:
   ...: #Plotting the Time Vs Number of Records
   ...: y = [len(df1),len(df2),len(df3),len(df4)]
   ...: plt.plot( timex[0],y[0],'ro',label='100 records')
   ...: plt.plot( timex[1], y[1],'bo',label='1000 records')
   ...: plt.plot( timex[2],y[2],'go',label='10000 records')
   ...: plt.plot( timex[3], y[3],'yo',label='100000 records')
   ...: legend = plt.legend(loc='upper left',bbox_to_anchor=(1, 1))
   ...: frame = legend.get_frame()
   ...: plt.xlabel('Time(in sec)')
   ...: plt.ylabel('Number of records')
   ...: plt.title('Time Vs Number of Records')
   ...: plt.show()
   ...:
   ...:
   ...: #Plotting the % of Memory Usage Vs Number of Records
   ...: y = [len(df1),len(df2),len(df3),len(df4)]
   ...: plt.plot( mem[0],y[0],'g^',label='100 records')
   ...: plt.plot( mem[1],y[1],'r^',label='1000 records')
   ...: plt.plot( mem[2],y[2],'b^',label='10000 records')
   ...: plt.plot( mem[3],y[3],'y^',label='100000 records')
   ...: legend = plt.legend(loc='upper left',bbox_to_anchor=(1, 1))
   ...: frame = legend.get_frame()
   ...: plt.xlabel('% of Memory Usage')
   ...: plt.ylabel('Number of records')
   ...: plt.title('% of Memory Usage Vs Number of Records')
   ...: plt.show()
U:\Anaconda3\lib\site-packages\surprise\evaluate.py:66: UserWarning: The evaluate() method is
deprecated. Please use model_selection.cross_validate() instead.
  'model_selection.cross_validate() instead.', UserWarning)
U:\Anaconda3\lib\site-packages\surprise\dataset.py:193: UserWarning: Using data.split() or
using load_from_folds() without using a CV iterator is now deprecated.
  UserWarning)
Evaluating RMSE of algorithm MatrixFacto.

------------
Fold 1
Fitting data with SGD...
RMSE: 0.8476
------------
Fold 2
Fitting data with SGD...
RMSE: 0.8899
------------
------------
Mean RMSE: 0.8688
------------
------------
Time1 0.2051386833190918
Evaluating RMSE of algorithm MatrixFacto.

------------
Fold 1
Fitting data with SGD...
RMSE: 2.0987
------------
Fold 2
```

```
Fitting data with SGD...
RMSE: 2.0479
------------
------------
Mean RMSE: 2.0733
------------
------------
Time2 0.19412517547607422
Evaluating RMSE of algorithm MatrixFacto.

------------
Fold 1
Fitting data with SGD...
RMSE: 2.0935
------------
Fold 2
Fitting data with SGD...
RMSE: 2.1563
------------
------------
Mean RMSE: 2.1249
------------
------------
Time3 1.436955451965332
Evaluating RMSE of algorithm MatrixFacto.

------------
Fold 1
Fitting data with SGD...
RMSE: 1.1451
------------
Fold 2
Fitting data with SGD...
RMSE: 1.1395
------------
------------
Mean RMSE: 1.1423
------------
------------
Time4 14.143411636352539
```




Time Vs Number of Records
Number of records
Time(in sec)
100 records
1000 records
10000 records
100000 records


% of Memory Usage Vs Number of Records
Number of records
% of Memory Usage
100 records
1000 records
10000 records
100000 records

In [4]: