In [41]:

```
from scipy.sparse import rand
import numpy as np
import scipy
import matplotlib.pyplot as plt
```

Data Generation

1a) Here I have used random.choice method to generate ratings 1 to7 in equal distribution in such a way that 200 out of 500 will be having ratings from 1 to 7. Considering empty ratings as 0 and so it's distribution is 0.6 which accumulates for 300 entries out of 500 with random seed

In [2]:

```
data = np.random.RandomState(43).choice([0,1,2,3,4,5,6,7],500,p=[0.6,.4/7,.4/7,.4/7,.4/7,.4/7,.4/7,.4
/7])
len(data)
np.count_nonzero(data)
```

Out[2]:

```
201
```

In [3]:

```
data.resize(25,20)
data
```

Out[3]:

```
array([[0, 1, 0, 0, 0, 5, 2, 0, 0, 3, 0, 4, 0, 0, 5, 0, 0, 0, 0, 5],
       [5, 7, 0, 7, 0, 2, 0, 6, 0, 0, 0, 0, 0, 1, 7, 6, 6, 0, 0, 0],
       [7, 0, 5, 2, 1, 0, 6, 0, 0, 7, 3, 0, 0, 0, 3, 0, 2, 2, 1, 0],
       [0, 0, 4, 0, 1, 0, 0, 0, 0, 0, 0, 0, 3, 3, 2, 0, 7, 0, 6, 0],
       [0, 7, 7, 0, 0, 0, 0, 0, 0, 1, 5, 5, 7, 3, 0, 0, 6, 0, 0, 1],
       [0, 0, 0, 0, 3, 3, 6, 4, 0, 6, 0, 5, 6, 5, 0, 0, 0, 2, 0, 0],
       [1, 0, 3, 0, 0, 0, 7, 2, 0, 0, 0, 3, 6, 0, 0, 3, 0, 0, 4, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 4, 0, 0, 6, 2, 7, 0, 0],
       [7, 0, 0, 6, 0, 0, 6, 0, 7, 7, 0, 0, 7, 7, 0, 0, 0, 2, 1, 2],
       [7, 2, 0, 0, 0, 4, 0, 0, 2, 0, 1, 0, 4, 0, 0, 5, 7, 0, 0, 7],
       [7, 6, 5, 0, 0, 0, 0, 6, 1, 7, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0],
       [0, 1, 0, 2, 0, 0, 0, 0, 5, 0, 0, 0, 0, 6, 0, 4, 5, 0, 0, 4],
       [0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 2, 3, 0, 0, 0, 7, 4, 7, 6, 0],
       [0, 1, 0, 0, 0, 0, 0, 7, 0, 2, 0, 0, 0, 0, 0, 0, 4, 0, 6, 0],
       [4, 5, 0, 1, 4, 6, 0, 4, 0, 0, 0, 0, 1, 0, 0, 7, 0, 0, 4, 0],
       [0, 0, 0, 0, 0, 6, 7, 0, 2, 0, 3, 4, 0, 0, 7, 2, 3, 1, 0, 0],
       [0, 4, 7, 0, 0, 0, 7, 1, 0, 7, 3, 0, 0, 1, 7, 4, 3, 3, 0, 0],
       [1, 0, 0, 0, 6, 0, 0, 0, 2, 0, 0, 0, 5, 3, 0, 1, 0, 0, 0, 0],
       [6, 0, 4, 1, 4, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 6, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 2, 4, 3, 0, 0, 0, 6, 3, 2, 0, 0, 0, 7],
       [0, 5, 4, 0, 6, 6, 5, 0, 0, 0, 0, 0, 5, 0, 0, 4, 0, 4, 0, 4],
       [0, 0, 0, 0, 1, 3, 6, 0, 6, 7, 0, 0, 7, 4, 0, 2, 5, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 4, 0, 7, 0, 0, 7, 5, 7, 7, 0, 0, 0, 0],
       [4, 2, 0, 6, 0, 0, 0, 0, 1, 4, 0, 0, 0, 4, 0, 7, 0, 0, 0, 0],
       [4, 1, 6, 7, 4, 0, 7, 3, 3, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0]])
```

Data in Tabular form (default index and column names)

In [4]:

```
from pandas import *
print(DataFrame(data))
```

```
    0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  \
0    0   1   0   0   0   5   2   0   0   3   0   4   0   0   5   0   0   0
1    5   7   0   7   0   2   0   6   0   0   0   0   0   1   7   6   6   0
```

```
2     7    0    5    2    1    0    6    0    0    7    3    0    0    0    3    0    2    2
3     0    0    4    0    1    0    0    0    0    0    0    0    3    3    2    0    7    0
4     0    7    7    0    0    0    0    0    0    1    5    5    7    3    0    0    6    0
5     0    0    0    0    3    3    6    4    0    6    0    5    6    5    0    0    0    2
6     1    0    3    0    0    0    7    2    0    0    0    3    6    0    0    3    0    0
7     0    0    0    0    0    0    0    0    0    0    5    0    4    0    0    6    2    7
8     7    0    0    6    0    0    6    0    7    7    0    0    7    7    0    0    0    2
9     7    2    0    0    0    4    0    0    2    0    1    0    4    0    0    5    7    0
10    7    6    5    0    0    0    0    6    1    7    2    0    0    0    0    0    0    2
11    0    1    0    2    0    0    0    0    5    0    0    0    0    6    0    4    5    0
12    0    0    0    0    0    0    6    0    0    0    2    3    0    0    0    7    4    7
13    0    1    0    0    0    0    0    7    0    2    0    0    0    0    0    0    4    0
14    4    5    0    1    4    6    0    4    0    0    0    0    1    0    0    7    0    0
15    0    0    0    0    0    6    7    0    2    0    3    4    0    0    7    2    3    1
16    0    4    7    0    0    0    7    1    0    7    3    0    0    1    7    4    3    3
17    1    0    0    0    6    0    0    0    2    0    0    0    5    3    0    1    0    0
18    6    0    4    1    4    0    0    0    0    0    0    2    2    0    0    6    0    0
19    0    0    0    0    0    0    0    2    4    3    0    0    0    6    3    2    0    0
20    0    5    4    0    6    6    5    0    0    0    0    0    5    0    0    4    0    4
21    0    0    0    0    1    3    6    0    6    7    0    0    7    4    0    2    5    0
22    0    0    0    0    0    0    0    4    0    7    0    0    7    5    7    7    0    0
23    4    2    0    6    0    0    0    0    1    4    0    0    0    4    0    7    0    0
24    4    1    6    7    4    0    7    3    3    0    0    0    0    0    4    0    0    0

     18   19
0     0    5
1     0    0
2     1    0
3     6    0
4     0    1
5     0    0
6     4    0
7     0    0
8     1    2
9     0    7
10    0    0
11    0    4
12    6    0
13    6    0
14    4    0
15    0    0
16    0    0
17    0    0
18    0    1
19    0    7
20    0    4
21    0    0
22    0    0
23    0    0
24    0    0
```

- As data contains 201 non-zero elements making one element as 0 so that data will have exactly 200 non-zero ratings

In [5]:

```python
data[24][3]=0
```

Checking each row and each column is having atleast 3 non-zero or rating between 1 to 7

In [6]:

```python
columns1 = (data != 0).sum(0)
rows1    = (data != 0).sum(1)
print(columns1)
print(rows1)
```

```
[11 12  9  7  9  8 11 10 10 12  8  7 13 12  9 16 12  9  7  8]
[ 7  9 11  7  9  9  8  5 10  9  8  7  7  5  9  9 11  6  8  7  9  9  6  7
  8]
```

In [7]:

```
sparsity = float(len(data.nonzero()[0]))
sparsity /= (data.shape[0] * data.shape[1])
sparsity *= 100
print('Sparsity: ',format(sparsity))
```

```
Sparsity:  40.0
```

In [8]:

```
def train_test_split(ratings,seed):
    test = np.zeros(ratings.shape)
    train = ratings.copy()
    for user in range(ratings.shape[0]):
        test_ratings = np.random.RandomState(seed).choice(ratings[user, :].nonzero()[0],
                                                          size=10)
        train[user, test_ratings] = 0
        test[user, test_ratings] = ratings[user, test_ratings]

    # Test and training are truly disjoint
    assert(np.all((train * test) == 0))
    return train, test
```

1b) Randomly selecting 150 of the 200 ratings as the training dataset and the remaining 50 as the test dataset.

In [9]:

```
ts1,tr1=train_test_split(data,44)
```

In [10]:

```
np.count_nonzero(ts1)
```

Out[10]:

```
50
```

In [11]:

```
np.count_nonzero(tr1)
```

Out[11]:

```
150
```

In [12]:

```
columns1 = (tr1 != 0).sum(0)
rows1    = (tr1 != 0).sum(1)
print(columns1)
print(rows1)
```

```
[11 12  7  3  8  7  6  9  8 10  5  6  6  7  7 10 11  8  5  4]
[6 6 7 6 6 6 6 5 6 6 6 6 6 5 6 6 7 6 6 6 6 6 6 6 6]
```

In [13]:

```
columns1 = (ts1 != 0).sum(0)
rows1    = (ts1 != 0).sum(1)
print(columns1)
print(rows1)
```

```
[0 0 2 4 1 1 5 1 2 2 3 1 7 5 2 6 1 1 2 4]
[1 3 4 1 3 3 2 0 4 3 2 1 1 0 3 3 4 0 2 1 3 3 0 1 2]
```

Test Data in tabular form

In [14]:

```
print(DataFrame(ts1))
```

```
    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17  \
0   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
1   0  0  0  7  0  0  0  0  0  0  0  0  0  1  7  0  0  0
2   0  0  0  2  0  0  0  0  0  7  3  0  0  0  0  0  0  2
3   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
4   0  0  0  0  0  0  0  0  0  1  0  0  7  3  0  0  0  0
5   0  0  0  0  0  0  6  0  0  0  0  5  6  0  0  0  0  0
6   0  0  0  0  0  0  7  0  0  0  0  0  0  0  0  3  0  0
7   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
8   0  0  0  0  0  0  6  0  0  0  0  0  7  7  0  0  0  0
9   0  0  0  0  0  4  0  0  0  0  0  0  4  0  0  5  0  0
10  0  0  5  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0
11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
12  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
13  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
14  0  0  0  1  0  0  0  4  0  0  0  0  1  0  0  0  0  0
15  0  0  0  0  0  0  0  0  2  0  0  0  0  0  7  2  0  0
16  0  0  0  0  0  0  7  0  0  0  3  0  0  1  0  0  3  0
17  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
18  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  6  0  0
19  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
20  0  0  0  0  6  0  0  0  0  0  0  0  5  0  0  4  0  0
21  0  0  0  0  0  0  6  0  0  0  0  0  7  4  0  0  0  0
22  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
23  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  7  0  0
24  0  0  6  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0

    18 19
0   0  5
1   0  0
2   0  0
3   6  0
4   0  0
5   0  0
6   0  0
7   0  0
8   0  2
9   0  0
10  0  0
11  0  4
12  6  0
13  0  0
14  0  0
15  0  0
16  0  0
17  0  0
18  0  0
19  0  7
20  0  0
21  0  0
22  0  0
23  0  0
24  0  0
```

Train Data in tabular form

In [15]:

```
print(DataFrame(tr1))
```

```
     0    1    2    3    4    5    6    7    8    9    10   11   12   13   14  \
0  0.0  1.0  0.0  0.0  0.0  5.0  2.0  0.0  0.0  3.0  0.0  4.0  0.0  0.0  5.0
1  5.0  7.0  0.0  0.0  0.0  2.0  0.0  6.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  7.0  0.0  5.0  0.0  1.0  0.0  6.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  3.0
3  0.0  0.0  4.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  3.0  3.0  2.0
4  0.0  7.0  7.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  5.0  5.0  0.0  0.0  0.0
5  0.0  0.0  0.0  0.0  3.0  3.0  0.0  4.0  0.0  6.0  0.0  0.0  0.0  5.0  0.0
6  1.0  0.0  3.0  0.0  0.0  0.0  0.0  2.0  0.0  0.0  0.0  3.0  6.0  0.0  0.0
```

```
7   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  5.0  0.0  4.0  0.0  0.0
8   7.0  0.0  0.0  6.0  0.0  0.0  0.0  0.0  7.0  7.0  0.0  0.0  0.0  0.0  0.0
9   7.0  2.0  0.0  0.0  0.0  0.0  0.0  0.0  2.0  0.0  1.0  0.0  0.0  0.0  0.0
10  7.0  6.0  0.0  0.0  0.0  0.0  0.0  6.0  1.0  7.0  0.0  0.0  0.0  0.0  0.0
11  0.0  1.0  0.0  2.0  0.0  0.0  0.0  0.0  5.0  0.0  0.0  0.0  0.0  6.0  0.0
12  0.0  0.0  0.0  0.0  0.0  0.0  6.0  0.0  0.0  0.0  2.0  3.0  0.0  0.0  0.0
13  0.0  1.0  0.0  0.0  0.0  0.0  0.0  7.0  0.0  2.0  0.0  0.0  0.0  0.0  0.0
14  4.0  5.0  0.0  0.0  4.0  6.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
15  0.0  0.0  0.0  0.0  0.0  6.0  7.0  0.0  0.0  0.0  3.0  4.0  0.0  0.0  0.0
16  0.0  4.0  7.0  0.0  0.0  0.0  0.0  1.0  0.0  7.0  0.0  0.0  0.0  0.0  7.0
17  1.0  0.0  0.0  0.0  6.0  0.0  0.0  0.0  2.0  0.0  0.0  0.0  5.0  3.0  0.0
18  6.0  0.0  4.0  0.0  4.0  0.0  0.0  0.0  0.0  0.0  0.0  2.0  2.0  0.0  0.0
19  0.0  0.0  0.0  0.0  0.0  0.0  0.0  2.0  4.0  3.0  0.0  0.0  0.0  6.0  3.0
20  0.0  5.0  4.0  0.0  0.0  6.0  5.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
21  0.0  0.0  0.0  0.0  1.0  3.0  0.0  0.0  6.0  7.0  0.0  0.0  0.0  0.0  0.0
22  0.0  0.0  0.0  0.0  0.0  0.0  0.0  4.0  0.0  7.0  0.0  0.0  7.0  5.0  7.0
23  4.0  2.0  0.0  6.0  0.0  0.0  0.0  0.0  1.0  4.0  0.0  0.0  0.0  4.0  0.0
24  4.0  1.0  0.0  0.0  4.0  0.0  7.0  3.0  0.0  0.0  0.0  0.0  0.0  0.0  4.0

     15   16   17   18   19
0   0.0  0.0  0.0  0.0  0.0
1   6.0  6.0  0.0  0.0  0.0
2   0.0  2.0  0.0  1.0  0.0
3   0.0  7.0  0.0  0.0  0.0
4   0.0  6.0  0.0  0.0  1.0
5   0.0  0.0  2.0  0.0  0.0
6   0.0  0.0  0.0  4.0  0.0
7   6.0  2.0  7.0  0.0  0.0
8   0.0  0.0  2.0  1.0  0.0
9   0.0  7.0  0.0  0.0  7.0
10  0.0  0.0  2.0  0.0  0.0
11  4.0  5.0  0.0  0.0  0.0
12  7.0  4.0  7.0  0.0  0.0
13  0.0  4.0  0.0  6.0  0.0
14  7.0  0.0  0.0  4.0  0.0
15  0.0  3.0  1.0  0.0  0.0
16  4.0  0.0  3.0  0.0  0.0
17  1.0  0.0  0.0  0.0  0.0
18  0.0  0.0  0.0  0.0  1.0
19  2.0  0.0  0.0  0.0  0.0
20  0.0  0.0  4.0  0.0  4.0
21  2.0  5.0  0.0  0.0  0.0
22  7.0  0.0  0.0  0.0  0.0
23  0.0  0.0  0.0  0.0  0.0
24  0.0  0.0  0.0  0.0  0.0
```

Functions to implement Matrix factorisation using Alternating Least Squares and SGD

Note:

```
* From link -http://www.diva-portal.se/smash/get/diva2:929350/FULLTEXT01.pdf, to solve for
the solution of problem (2.2) one can fix either P or Q for the optimal solution.
* Both ALS and SGD starts by initializing Q and P with random values, thus the results will
not be the same when running multiple times.
* ALS tries to solve 2.2 by alternating between holding P and Q fixed untilconvergence. By
doing so the iterations are able to run independently which can potentially speed up the co
mputations by using multiple cores.
* In SGD, for each iteration pick a random rating Su,i and update Q and P according to thes
e formulae of eq 2.5 & 2.6.
* Hyperparameter        | SGD                  | ALS
------------------------------------------------------------
  Learning rate         | Between1 0.02 - 0.06| Unsupported
  Linear regularization| 1e-10                |1e-10
  Regularization(⸍)     | 1e-8                 |1e-8
```

In [20]:

```python
#np.transpose(np.nonzero(ts1))
from numpy.linalg import solve
```

```python
class ExplicitMF():
    def __init__(self,
                 ratings,
                 n_factors=40,
                 learning='sgd',
                 item_fact_reg=0.0,
                 user_fact_reg=0.0,
                 item_bias_reg=0.0,
                 user_bias_reg=0.0,
                 verbose=False):
        """
        Train a matrix factorization model to predict empty
        entries in a matrix. The terminology assumes a
        ratings matrix which is ~ user x item

        Params
        ======
        ratings : (ndarray)
            User x Item matrix with corresponding ratings

        n_factors : (int)
            Number of latent factors to use in matrix
            factorization model
        learning : (str)
            Method of optimization. Options include
            'sgd' or 'als'.

        item_fact_reg : (float)
            Regularization term for item latent factors

        user_fact_reg : (float)
            Regularization term for user latent factors

        item_bias_reg : (float)
            Regularization term for item biases

        user_bias_reg : (float)
            Regularization term for user biases

        verbose : (bool)
            Whether or not to printout training progress
        """

        self.ratings = ratings
        self.n_users, self.n_items = ratings.shape
        self.n_factors = n_factors
        self.item_fact_reg = item_fact_reg
        self.user_fact_reg = user_fact_reg
        self.item_bias_reg = item_bias_reg
        self.user_bias_reg = user_bias_reg
        self.learning = learning
        if self.learning == 'sgd':
            self.sample_row, self.sample_col = self.ratings.nonzero()
            self.n_samples = len(self.sample_row)
        self._v = verbose

    def als_step(self,
                 latent_vectors,
                 fixed_vecs,
                 ratings,
                 _lambda,
                 type='user'):
        """
        One of the two ALS steps. Solve for the latent vectors
        specified by type.
        """
        if type == 'user':
            # Precompute
            YTY = fixed_vecs.T.dot(fixed_vecs)
            lambdaI = np.eye(YTY.shape[0]) * _lambda

            for u in range(latent_vectors.shape[0]):
                latent_vectors[u, :] = solve((YTY + lambdaI),
                                             ratings[u, :].dot(fixed_vecs))
        elif type == 'item':
            # Precompute
            XTX = fixed_vecs.T.dot(fixed_vecs)
```

```
        lambdaI = np.eye(XTX.shape[0]) * _lambda

        for i in range(latent_vectors.shape[0]):
            latent_vectors[i, :] = solve((XTX + lambdaI),
                                         ratings[:, i].T.dot(fixed_vecs))
    return latent_vectors

def train(self, n_iter=10, learning_rate=0.1):
    """ Train model for n_iter iterations from scratch."""
    # initialize latent vectors
    self.user_vecs = np.random.normal(scale=1./self.n_factors,\
                                      size=(self.n_users, self.n_factors))
    self.item_vecs = np.random.normal(scale=1./self.n_factors,
                                      size=(self.n_items, self.n_factors))

    if self.learning == 'als':
        self.partial_train(n_iter)
    elif self.learning == 'sgd':
        self.learning_rate = learning_rate
        self.user_bias = np.zeros(self.n_users)
        self.item_bias = np.zeros(self.n_items)
        self.global_bias = np.mean(self.ratings[np.where(self.ratings != 0)])
        self.partial_train(n_iter)


def partial_train(self, n_iter):
    """
    Train model for n_iter iterations. Can be
    called multiple times for further training.
    """
    ctr = 1
    while ctr <= n_iter:
        if ctr % 10 == 0 and self._v:
            print('\tcurrent iteration:'+str(ctr))
        if self.learning == 'als':
            self.user_vecs = self.als_step(self.user_vecs,
                                           self.item_vecs,
                                           self.ratings,
                                           self.user_fact_reg,
                                           type='user')
            self.item_vecs = self.als_step(self.item_vecs,
                                           self.user_vecs,
                                           self.ratings,
                                           self.item_fact_reg,
                                           type='item')
        elif self.learning == 'sgd':
            self.training_indices = np.arange(self.n_samples)
            np.random.shuffle(self.training_indices)
            self.sgd()
        ctr += 1

def sgd(self):
    for idx in self.training_indices:
        u = self.sample_row[idx]
        i = self.sample_col[idx]
        prediction = self.predict(u, i)
        e = (self.ratings[u,i] - prediction) # error

        # Update biases
        self.user_bias[u] += self.learning_rate * \
                             (e - self.user_bias_reg * self.user_bias[u])
        self.item_bias[i] += self.learning_rate * \
                             (e - self.item_bias_reg * self.item_bias[i])

        #Update latent factors
        self.user_vecs[u, :] += self.learning_rate * \
                                (e * self.item_vecs[i, :] - \
                                 self.user_fact_reg * self.user_vecs[u,:])
        self.item_vecs[i, :] += self.learning_rate * \
                                (e * self.user_vecs[u, :] - \
                                 self.item_fact_reg * self.item_vecs[i,:])
def predict(self, u, i):
    """ Single user and item prediction."""
    if self.learning == 'als':
        return self.user_vecs[u, :].dot(self.item_vecs[i, :].T)
    elif self.learning == 'sgd':
        prediction = self.global_bias + self.user_bias[u] + self.item_bias[i]
```

```
            prediction += self.user_vecs[u, :].dot(self.item_vecs[i, :].T)
            return prediction

    def predict_all(self):
        """ Predict ratings for every user and item."""
        predictions = np.zeros((self.user_vecs.shape[0],
                                self.item_vecs.shape[0]))
        for u in range(self.user_vecs.shape[0]):
            for i in range(self.item_vecs.shape[0]):
                predictions[u, i] = self.predict(u, i)

        return predictions

    def calculate_learning_curve(self, iter_array, test, learning_rate=0.1):
        """
        Keep track of MSE as a function of training iterations.

        Params
        ======
        iter_array : (list)
            List of numbers of iterations to train for each step of
            the learning curve. e.g. [1, 5, 10, 20]
        test : (2D ndarray)
            Testing dataset (assumed to be user x item).

        The function creates two new class attributes:

        train_mse : (list)
            Training data MSE values for each value of iter_array
        test_mse : (list)
            Test data MSE values for each value of iter_array
        """
        iter_array.sort()
        self.train_mse =[]
        self.test_mse = []
        iter_diff = 0
        for (i, n_iter) in enumerate(iter_array):
            if self._v:
                print('Iteration:='+str(n_iter))
            if i == 0:
                self.train(n_iter - iter_diff, learning_rate)
            else:
                self.partial_train(n_iter - iter_diff)

            predictions = self.predict_all()

            self.train_mse += [get_mse(predictions, self.ratings)]
            self.test_mse += [get_mse(predictions, test)]
            if self._v:
                print('Train mse: ' + str(self.train_mse[-1]))
                print('Test  mse: ' + str(self.test_mse[-1]))
            iter_diff = n_iter
```

In [21]:

```
from sklearn.metrics import mean_squared_error

def get_mse(pred, actual):
    # Ignore nonzero terms.
    pred = pred[actual.nonzero()].flatten()
    actual = actual[actual.nonzero()].flatten()
    return mean_squared_error(pred, actual)

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

def plot_learning_curve(iter_array, model):
    plt.plot(iter_array, model.train_mse, \
             label='Training', linewidth=5)
    plt.plot(iter_array, model.test_mse, \
             label='Test', linewidth=5)

    plt.xticks(fontsize=16);
    plt.yticks(fontsize=16);
```

```
    plt.xlabel('iterations', fontsize=20);
    plt.ylabel('(total ratings including zeros) MSE', fontsize=20);
    plt.legend(loc='best', fontsize=10);
```

1c) To find a matrix factorization of the training data and include only two latent factors Using ALS without regularization

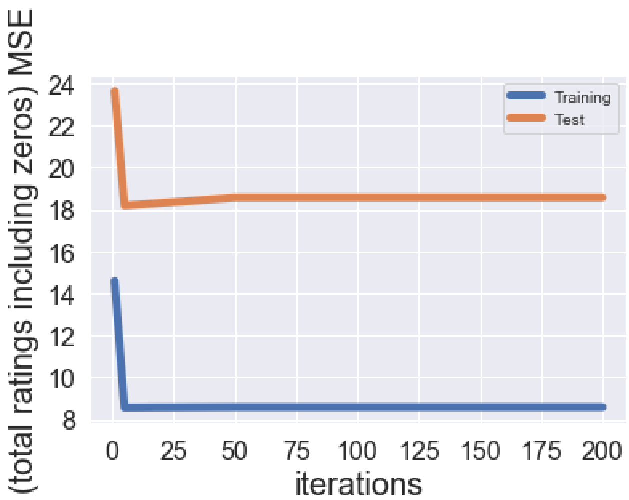
In [22]:

```
MF_ALS = ExplicitMF(tr1, 2, learning='als', verbose=True)
iter_array1 = [1, 5, 50, 100, 200]
MF_ALS.calculate_learning_curve(iter_array1, ts1, learning_rate=0.01)
```

```
Iteration:=1
Train mse: 14.583764881645214
Test  mse: 23.625956435732075
Iteration:=5
Train mse: 8.55528494292431
Test  mse: 18.168304550631145
Iteration:=50
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
Train mse: 8.579917562079256
Test  mse: 18.554979109898692
Iteration:=100
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
Train mse: 8.57991971138061
Test  mse: 18.554988356364504
Iteration:=200
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
 current iteration:60
 current iteration:70
 current iteration:80
 current iteration:90
 current iteration:100
Train mse: 8.579919711399462
Test  mse: 18.554988356445598
```

In [23]:

```
plot_learning_curve(iter_array1, MF_ALS)
```



1d) Calculating MSE by considering only test data and also considering only non-zero elements of that test data

In [35]:

```
list1_org=[]
list2_pred=[]
predictions = np.zeros((len(ts1), len(ts1[0])),dtype=object)
ts1_pred=ts1.copy();
for u in range(len(ts1)):
    for i in range(len(ts1[0])):
        if ts1[u][i]!=0:
            list1_org.append(round(MF_ALS.predict(u,i),2))
            list2_pred.append(ts1[u][i])
            predictions[u][i]=str(ts1[u][i])+' | '+str(round(MF_ALS.predict(u,i),2))
print("TEST DATA Ratings MSE Using ALS without regularization "+str(mean_squared_error(list2_pred,
list1_org)))
```

```
TEST DATA Ratings MSE Using ALS without regularization 18.55587
```

1d) Printing predictions and original ratings of non-zero ratings of test data side by side (left side--> original and right side of pipe is predicted value)

In [26]:

```
predictions
```

Out[26]:

```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '5 | 0.38'],
       [0, 0, 0, '7 | 0.39', 0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 0.9',
        '7 | 1.68', 0, 0, 0, 0, 0],
       [0, 0, 0, '2 | -0.04', 0, 0, 0, 0, 0, '7 | 0.4', '3 | 1.32', 0, 0,
        0, 0, 0, 0, '2 | 1.56', 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '6 | 0.41',
        0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | -1.29', 0, 0, '7 | 0.1',
        '3 | -0.96', 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '6 | -0.6', 0, 0, 0, 0, '5 | -0.36',
        '6 | 1.34', 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | 0.69', 0, 0, 0, 0, 0, 0, 0, 0, '3 | 1.03',
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '6 | -0.68', 0, 0, 0, 0, 0, '7 | 1.75',
        '7 | 3.07', 0, 0, 0, 0, 0, '2 | -0.28'],
       [0, 0, 0, 0, 0, '4 | 1.86', 0, 0, 0, 0, 0, 0, '4 | 0.48', 0, 0,
        '5 | 2.15', 0, 0, 0, 0],
       [0, 0, '5 | 1.23', 0, 0, 0, 0, 0, 0, 0, '2 | 0.24', 0, 0, 0, 0, 0,
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '4 | 0.21'],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '6 | 0.63',
        0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, '1 | 0.28', 0, 0, 0, '4 | 1.5', 0, 0, 0, 0, '1 | 0.79',
        0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, '2 | -0.64', 0, 0, 0, 0, 0, '7 | -0.05',
        '2 | 1.37', 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | 0.89', 0, 0, 0, '3 | 0.41', 0, 0,
        '1 | 2.33', 0, 0, '3 | 2.27', 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, '1 | 0.12', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '6 | 1.38',
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '7 | -0.39'],
       [0, 0, 0, 0, '6 | 0.66', 0, 0, 0, 0, 0, 0, 0, '5 | 0.08', 0, 0,
        '4 | 1.81', 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '6 | 0.17', 0, 0, 0, 0, 0, '7 | 1.33',
        '4 | 2.12', 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '7 | 1.79', 0, 0, 0,
        0],
       [0, 0, '6 | 1.5', 0, 0, 0, 0, 0, '3 | 0.5', 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0]], dtype=object)
```

```
        0, 0, 0, 0]], dtype=object)
```

To find a matrix factorization of the training data and include only two latent factors Using ALS with regularization

In [27]:

```
MF_ALS2 = ExplicitMF(tr1, n_factors=2, \
                    user_fact_reg=1, item_fact_reg=1)

iter_array2 = [1, 2, 5, 10, 25, 50, 100]
MF_ALS2.calculate_learning_curve(iter_array2, ts1)
```

In [36]:

```
list1_org=[]
list2_pred=[]
predictions = np.zeros((len(ts1), len(ts1[0])),dtype=object)
ts1_pred=ts1.copy();
for u in range(len(ts1)):
    for i in range(len(ts1[0])):
        if ts1[u][i]!=0:
            list1_org.append(round(MF_ALS2.predict(u,i),2))
            list2_pred.append(ts1[u][i])
            predictions[u][i]=str(ts1[u][i])+' | '+str(round(MF_ALS2.predict(u,i),2))
print("TEST DATA Ratings MSE Using ALS with regularization "+str(mean_squared_error(list2_pred,lis
t1_org)))
```

```
TEST DATA Ratings MSE Using ALS with regularization 6.906573999999999
```

Predictions of test data using vectors obtained by considering regularization for ALS

In [30]:

```
predictions
```

Out[30]:

```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '5 | 2.07'],
       [0, 0, 0, '7 | 6.07', 0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 6.56',
        '7 | 6.15', 0, 0, 0, 0, 0],
       [0, 0, 0, '2 | 3.97', 0, 0, 0, 0, 0, '7 | 4.74', '3 | 1.54', 0, 0,
        0, 0, 0, 0, '2 | 2.06', 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '6 | 1.41',
        0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 7.41', 0, 0, '7 | 6.19',
        '3 | 7.34', 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '6 | 5.54', 0, 0, 0, 0, '5 | 3.23', '6 | 4.4',
        0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | 4.32', 0, 0, 0, 0, 0, 0, 0, 0, '3 | 3.24',
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '6 | 6.96', 0, 0, 0, 0, 0, '7 | 5.06',
        '7 | 6.24', 0, 0, 0, 0, 0, '2 | 2.51'],
       [0, 0, 0, 0, 0, '4 | 5.56', 0, 0, 0, 0, 0, 0, '4 | 6.24', 0, 0,
        '5 | 5.16', 0, 0, 0, 0],
       [0, 0, '5 | 5.97', 0, 0, 0, 0, 0, 0, 0, '2 | 2.84', 0, 0, 0, 0, 0,
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '4 | 1.64'],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '6 | 4.36',
        0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, '1 | 5.97', 0, 0, 0, '4 | 4.68', 0, 0, 0, 0, '1 | 5.79',
        0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, '2 | 4.39', 0, 0, 0, 0, 0, '7 | 5.25',
        '2 | 4.62', 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | 6.34', 0, 0, 0, '3 | 2.69', 0, 0,
        '1 | 5.82', 0, 0, '3 | 4.67', 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, '1 | 4.03', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '6 | 3.48',
```

```
         0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         '7 | 0.74'],
        [0, 0, 0, 0, '6 | 4.65', 0, 0, 0, 0, 0, 0, 0, '5 | 5.65', 0, 0,
         '4 | 5.09', 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, '6 | 5.62', 0, 0, 0, 0, 0, '7 | 3.85',
         '4 | 4.94', 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '7 | 3.33', 0, 0, 0,
         0],
        [0, 0, '6 | 4.48', 0, 0, 0, 0, 0, '3 | 3.49', 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0]], dtype=object)
```

To find a matrix factorization of the training data and include only two latent factors Using SGD

In [31]:

```
MF_SGD = ExplicitMF(tr1, 2, learning='sgd', verbose=True)
iter_array = [1, 2, 5, 10, 25, 50, 100, 200]
MF_SGD.calculate_learning_curve(iter_array, ts1, learning_rate=0.001)
```

```
Iteration:=1
Train mse: 4.195871507930667
Test  mse: 4.643430457174636
Iteration:=2
Train mse: 4.171717388845075
Test  mse: 4.636769086176312
Iteration:=5
Train mse: 4.101454487684428
Test  mse: 4.617592047357322
Iteration:=10
Train mse: 3.9910891705179656
Test  mse: 4.588195518323753
Iteration:=25
 current iteration:10
Train mse: 3.702901639071473
Test  mse: 4.516362794083029
Iteration:=50
 current iteration:10
 current iteration:20
Train mse: 3.32804391285417
Test  mse: 4.437677739380121
Iteration:=100
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
Train mse: 2.802844725907474
Test  mse: 4.3697558745254925
Iteration:=200
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
 current iteration:60
 current iteration:70
 current iteration:80
 current iteration:90
 current iteration:100
Train mse: 2.104254694641882
Test  mse: 4.385814342898085
```

In [39]:

```
list1_org=[]
list2_pred=[]
sgd_predictions = np.zeros((len(ts1), len(ts1[0])),dtype=object)
ts1_pred=ts1.copy();
for u in range(len(ts1)):
    for i in range(len(ts1[0])):
        if ts1[u][i]!=0:
```

```
            list1_org.append(round(MF_SGD.predict(u,i),2))
            list2_pred.append(ts1[u][i])
            sgd_predictions[u][i]=str(ts1[u][i])+' | '+str(round(MF_SGD.predict(u,i),2))
print("TEST DATA Ratings MSE Using SGD "+str(mean_squared_error(list2_pred,list1_org)))
```

```
TEST DATA Ratings MSE Using SGD 4.385426
```

Predictions of test data using SGD

In [40]:

```
sgd_predictions
```

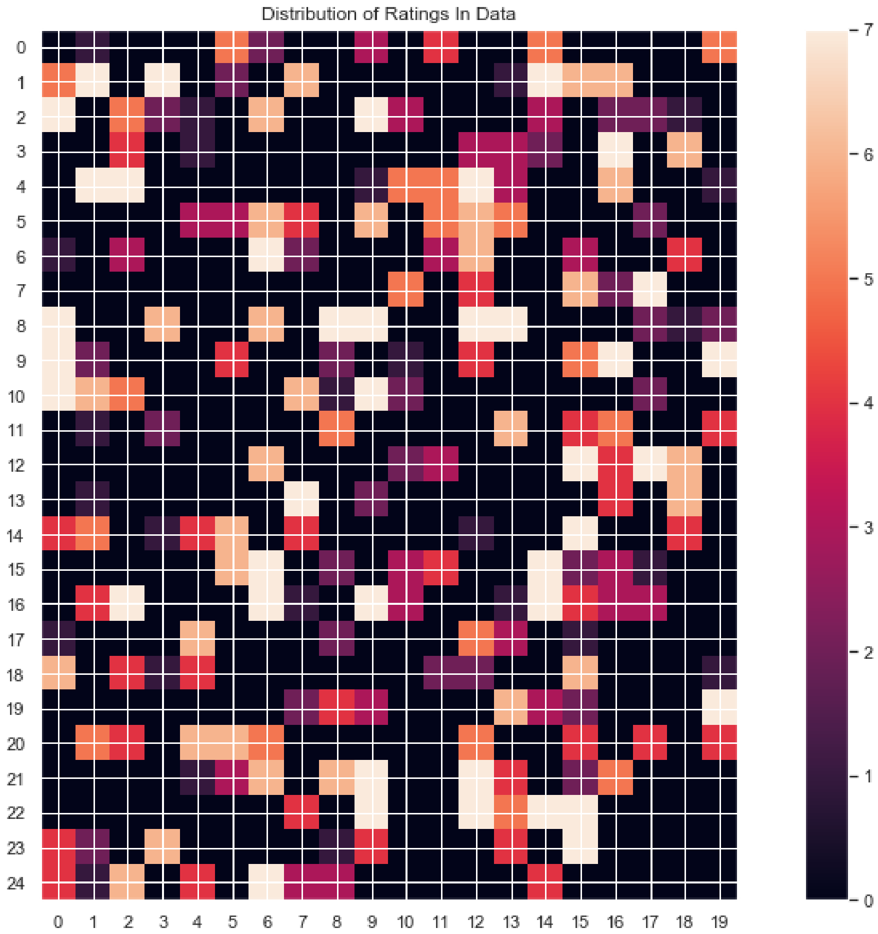
Out[40]:

```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '5 | 2.73'],
       [0, 0, 0, '7 | 5.56', 0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 3.75',
        '7 | 5.65', 0, 0, 0, 0, 0],
       [0, 0, 0, '2 | 4.16', 0, 0, 0, 0, 0, '7 | 5.27', '3 | 2.44', 0, 0,
        0, 0, 0, 0, '2 | 2.59', 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '6 | 3.51',
        0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 5.4', 0, 0, '7 | 4.76',
        '3 | 3.68', 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '6 | 4.96', 0, 0, 0, 0, '5 | 3.51', '6 | 4.52',
        0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | 4.51', 0, 0, 0, 0, 0, 0, 0, 0, '3 | 3.15',
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '6 | 5.5', 0, 0, 0, 0, 0, '7 | 5.69',
        '7 | 6.03', 0, 0, 0, 0, 0, '2 | 4.0'],
       [0, 0, 0, 0, 0, '4 | 4.7', 0, 0, 0, 0, 0, 0, '4 | 4.61', 0, 0,
        '5 | 3.91', 0, 0, 0, 0],
       [0, 0, '5 | 5.17', 0, 0, 0, 0, 0, 0, 0, '2 | 3.52', 0, 0, 0, 0, 0,
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '4 | 3.13'],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '6 | 4.19',
        0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, '1 | 5.27', 0, 0, 0, '4 | 5.04', 0, 0, 0, 0, '1 | 5.04',
        0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, '2 | 4.35', 0, 0, 0, 0, 0, '7 | 4.62',
        '2 | 3.5', 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | 5.45', 0, 0, 0, '3 | 3.55', 0, 0,
        '1 | 5.05', 0, 0, '3 | 4.53', 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, '1 | 3.33', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '6 | 2.16',
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '7 | 3.15'],
       [0, 0, 0, 0, '6 | 4.01', 0, 0, 0, 0, 0, 0, 0, '5 | 4.52', 0, 0,
        '4 | 3.81', 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '6 | 4.92', 0, 0, 0, 0, 0, '7 | 4.66',
        '4 | 4.84', 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '7 | 2.83', 0, 0, 0,
        0],
       [0, 0, '6 | 4.67', 0, 0, 0, 0, 0, '3 | 3.07', 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0]], dtype=object)
```

In [48]:

```
%matplotlib inline
plt.figure(figsize=(15,10))
plt.imshow(data)
plt.xticks(range(len(data[0])))
plt.yticks(range(len(data)))
ax = plt.gca()
ax.set_xticklabels(range(len(data[0])))
ax.set_yticklabels(range(len(data)))
plt.title("Distribution of Ratings In Data")
```

```
plt.colorbar();
```



2a) Populate the Ratings matrix in step 1.a in such a way (you are free to choose the rating values) that the MSE obtained in 1.d is minimized. Repeat all steps of Q#1 and submit the answers.

Note: Here Assumption is that if we consider all equal ratings, then MSE may decrease which is not practical in real time. Here every user rated the item in similar way and for all movies given same rating.

In [51]:

```
data_same = np.random.RandomState(43).choice([0,1,2,3,4,5,6,7],500,p=[0.6,0.4,0,0,0,0,0,0])
len(data_same)
np.count_nonzero(data_same)
```

Out[51]:

```
201
```

In [52]:

```
data_same.resize(25,20)
```

In [53]:

```
data_same[24][3]=0
```

In [54]:

```
ts1_same,tr1_same=train_test_split(data_same,44)
```

In [55]:

```
np.count_nonzero(ts1_same)
```

Out[55]:

```
50
```

In [56]:

```
MF_ALS_same = ExplicitMF(tr1_same, 2, learning='als', verbose=True)
iter_array1 = [1, 5, 50, 100, 200]
MF_ALS_same.calculate_learning_curve(iter_array1, ts1_same, learning_rate=0.01)
```

```
Iteration:=1
Train mse: 0.6432517070142811
Test  mse: 0.9767374397558167
Iteration:=5
Train mse: 0.3507550450877766
Test  mse: 0.6181377953927407
Iteration:=50
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
Train mse: 0.3348180654538059
Test  mse: 0.6946923410459895
Iteration:=100
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
Train mse: 0.3348180625659062
Test  mse: 0.6946923712213617
Iteration:=200
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
 current iteration:60
 current iteration:70
 current iteration:80
 current iteration:90
 current iteration:100
Train mse: 0.3348180625659049
Test  mse: 0.694692371221376
```

In [57]:

```
list1_org=[]
list2_pred=[]
predictions_same = np.zeros((len(ts1_same), len(ts1_same[0])),dtype=object)
ts1_same_pred=ts1_same.copy();
for u in range(len(ts1_same)):
    for i in range(len(ts1_same[0])):
        if ts1_same[u][i]!=0:
            list1_org.append(round(MF_ALS_same.predict(u,i),2))
            list2_pred.append(ts1_same[u][i])
            predictions_same[u][i]=str(ts1_same[u][i])+' | '+str(round(MF_ALS_same.predict(u,i),2))
print("TEST DATA (50 RATINGS) MSE "+str(mean_squared_error(list2_pred,list1_org)))
```

```
TEST DATA (50 RATINGS) MSE 0.6953800000000001
```

In [58]:

```
predictions_same
```

Out[58]:

```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '1 | 0.22'],
       [0, 0, 0, '1 | 0.13', 0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 0.32',
        '1 | 0.35', 0, 0, 0, 0, 0],
       [0, 0, 0, '1 | 0.03', 0, 0, 0, 0, 0, '1 | 0.17', '1 | 0.45', 0, 0,
        0, 0, 0, 0, '1 | 0.43', 0, 0],
```

```
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 0.18',
        0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | -0.21', 0, 0, '1 | 0.19',
        '1 | -0.24', 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '1 | 0.03', 0, 0, 0, 0, '1 | -0.03',
        '1 | 0.21', 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '1 | 0.3', 0, 0, 0, 0, 0, 0, 0, 0, '1 | 0.31',
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '1 | 0.01', 0, 0, 0, 0, 0, '1 | 0.19',
        '1 | 0.46', 0, 0, 0, 0, 0, '1 | -0.03'],
       [0, 0, 0, 0, 0, '1 | 0.37', 0, 0, 0, 0, 0, 0, '1 | 0.22', 0, 0,
        '1 | 0.33', 0, 0, 0, 0],
       [0, 0, '1 | 0.13', 0, 0, 0, 0, 0, 0, 0, '1 | -0.04', 0, 0, 0, 0,
        0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '1 | 0.03'],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 0.19',
        0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, '1 | 0.13', 0, 0, 0, '1 | 0.41', 0, 0, 0, 0, '1 | 0.23',
        0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, '1 | -0.16', 0, 0, 0, 0, 0, '1 | 0.08',
        '1 | 0.11', 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '1 | 0.17', 0, 0, 0, '1 | 0.08', 0, 0,
        '1 | 0.5', 0, 0, '1 | 0.41', 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, '1 | -0.01', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '1 | 0.21', 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '1 | -0.2'],
       [0, 0, 0, 0, '1 | 0.28', 0, 0, 0, 0, 0, 0, 0, '1 | 0.19', 0, 0,
        '1 | 0.2', 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '1 | 0.15', 0, 0, 0, 0, 0, '1 | 0.23',
        '1 | 0.41', 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 0.59', 0, 0, 0,
        0],
       [0, 0, '1 | 0.28', 0, 0, 0, 0, 0, '1 | 0.37', 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0]], dtype=object)
```

In [59]:

```
MF_SGD_same = ExplicitMF(tr1_same, 2, learning='sgd', verbose=True)
iter_array1 = [1, 5, 50, 100, 200]
MF_SGD_same.calculate_learning_curve(iter_array1, ts1_same, learning_rate=0.01)
```

```
Iteration:=1
Train mse: 0.09676750737306158
Test  mse: 0.09095687529668112
Iteration:=5
Train mse: 0.06449363468043266
Test  mse: 0.07903922797725894
Iteration:=50
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
Train mse: 0.009769679995539612
Test  mse: 0.042971147497439376
Iteration:=100
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
Train mse: 0.0037593381081249292
Test  mse: 0.03146834358646527
Iteration:=200
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
 current iteration:60
```

```
 current iteration:70
 current iteration:80
 current iteration:90
 current iteration:100
Train mse: 0.0011807227845635764
Test  mse: 0.022762015170109517
```

In [60]:

```
list1_org=[]
list2_pred=[]
sgd_predictions_same = np.zeros((len(ts1_same), len(ts1_same[0])),dtype=object)
ts1_same_pred=ts1_same.copy();
for u in range(len(ts1_same)):
    for i in range(len(ts1_same[0])):
        if ts1_same[u][i]!=0:
            list1_org.append(round(MF_SGD_same.predict(u,i),2))
            list2_pred.append(ts1_same[u][i])
            sgd_predictions_same[u][i]=str(ts1_same[u][i])+' | '+str(round(MF_SGD_same.predict(u,i)
2))
print("TEST DATA (50 RATINGS) MSE "+str(mean_squared_error(list2_pred,list1_org)))
```

```
TEST DATA (50 RATINGS) MSE 0.022809999999999997
```

In [61]:

```
sgd_predictions_same
```

Out[61]:

```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '1 | 1.1'],
       [0, 0, 0, '1 | 0.77', 0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 0.94',
        '1 | 0.34', 0, 0, 0, 0, 0],
       [0, 0, 0, '1 | 0.89', 0, 0, 0, 0, 0, '1 | 1.01', '1 | 0.94', 0, 0,
        0, 0, 0, 0, '1 | 1.15', 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 1.14',
        0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 0.96', 0, 0, '1 | 1.01',
        '1 | 0.97', 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '1 | 0.91', 0, 0, 0, 0, '1 | 1.0', '1 | 0.97',
        0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '1 | 0.95', 0, 0, 0, 0, 0, 0, 0, 0, '1 | 0.96',
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '1 | 0.87', 0, 0, 0, 0, 0, '1 | 1.02',
        '1 | 1.0', 0, 0, 0, 0, 0, '1 | 1.08'],
       [0, 0, 0, 0, 0, '1 | 0.74', 0, 0, 0, 0, 0, 0, '1 | 0.89', 0, 0,
        '1 | 1.01', 0, 0, 0, 0],
       [0, 0, '1 | 1.26', 0, 0, 0, 0, 0, 0, 0, '1 | 1.48', 0, 0, 0, 0, 0,
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '1 | 0.88'],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 1.11',
        0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, '1 | 0.94', 0, 0, 0, '1 | 1.04', 0, 0, 0, 0, '1 | 0.99',
        0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, '1 | 0.99', 0, 0, 0, 0, 0, '1 | 1.12',
        '1 | 1.0', 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '1 | 1.02', 0, 0, 0, '1 | 0.98', 0, 0,
        '1 | 0.96', 0, 0, '1 | 0.97', 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, '1 | 0.97', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 0.97',
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '1 | 1.16'],
       [0, 0, 0, 0, '1 | 0.99', 0, 0, 0, 0, 0, 0, 0, '1 | 0.98', 0, 0,
        '1 | 0.98', 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '1 | 0.73', 0, 0, 0, 0, 0, '1 | 0.9',
        '1 | 0.96', 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '1 | 0.98', 0, 0, 0,
        0],
```

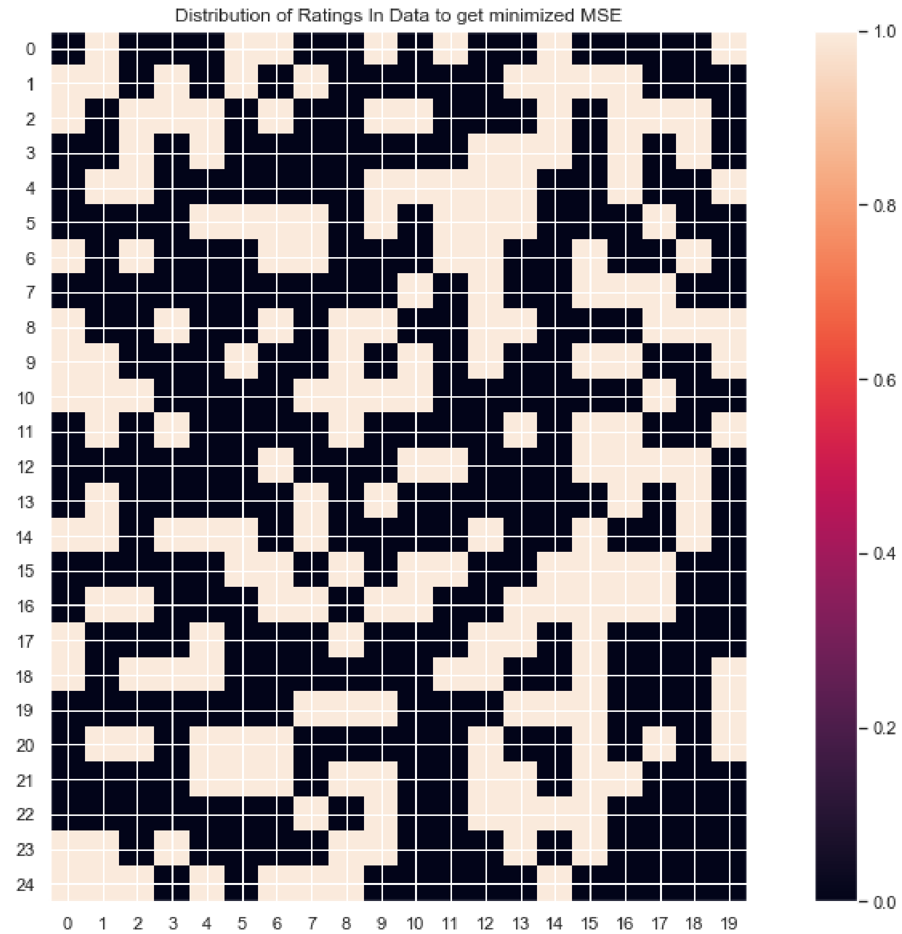
```
[0, 0, '1 | 0.98', 0, 0, 0, 0, 0, '1 | 0.99', 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0]], dtype=object)
```

Observations:

```
* Here if we consider all non-zero ratings as same ratings (1 here), we have observed that
mse has been drastically dropped to less than 1. Hence the data with same ratings can
minimize the MSE calculated in q#1 (mse from 18.55 to 0.69).
* But also observed that if we consider same rating always the predicted rating will be les
s than one as this may be due to empty ratings here are considered to be 0 ratings and
training model get dominated by 0 ratings and there is no much variance between data making
it to always predict to a value less than 0 but good thing here is mse is minimized.
* Practically this type of data doesn't exist in real world (i.e., giving same rating to
all items by all users) but to get most minimised mse we have considered this case.
```

In [62]:

```
%matplotlib inline
plt.figure(figsize=(15,10))
plt.imshow(data_same)
plt.xticks(range(len(data_same[0])))
plt.yticks(range(len(data_same)))
ax = plt.gca()
ax.set_xticklabels(range(len(data_same[0])))
ax.set_yticklabels(range(len(data_same)))
plt.title("Distribution of Ratings In Data to get minimized MSE")
plt.colorbar();
```



3a) Now populate the Ratings matrix in such a way that the MSE obtained in 1.d is maximized.

Assumptions:

```
* Considering data with more variance that is ratings are varied such a way that for same i
tem few users gives best rating and other giving worst rating making to have more variance
```

will yield maximise MSE
* To generate this type of data we need to consider ratings in such a way that their distribution is not uniform. Here I have considered 0.6 percent of data wil be empty ratings, 1,2,3,5,6 have 0.0 distribution and 4 rating will be in 0.2 and 7 rating will be in 0.15 percent of toatl data.

In [67]:

```
data_diff = np.random.RandomState(43).choice([0,1,2,3,4,5,6,7],500,p=[0.6,0.01,0.01,0.01,0.2,0.01,0.01,0.15])
len(data_diff)
np.count_nonzero(data_diff)
```

Out[67]:

```
201
```

In [68]:

```
data_diff.resize(25,20)
```

In [69]:

```
data_diff[24][3]=0
```

In [70]:

```
ts1_diff,tr1_diff=train_test_split(data_diff,44)
```

In [71]:

```
np.count_nonzero(ts1_diff)
```

Out[71]:

```
50
```

In [72]:

```
MF_ALS_diff = ExplicitMF(tr1_diff, 2, learning='als', verbose=True)
iter_array1 = [1, 5, 50, 100, 200]
MF_ALS_diff.calculate_learning_curve(iter_array1, ts1_diff, learning_rate=0.01)
```

```
Iteration:=1
Train mse: 15.82538019820787
Test  mse: 21.293548757793197
Iteration:=5
Train mse: 10.67067717619754
Test  mse: 22.90326060587929
Iteration:=50
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
Train mse: 10.534536409036066
Test  mse: 23.205178408209317
Iteration:=100
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
Train mse: 10.534536525494122
Test  mse: 23.2051786700374 6
Iteration:=200
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
```

```
 current iteration:50
 current iteration:60
 current iteration:70
 current iteration:80
 current iteration:90
 current iteration:100
Train mse: 10.534536525494142
Test  mse: 23.205178670037498
```

In [73]:

```python
list1_org=[]
list2_pred=[]
predictions_diff = np.zeros((len(ts1_diff), len(ts1_diff[0])),dtype=object)
ts1_diff_pred=ts1_diff.copy();
for u in range(len(ts1_diff)):
    for i in range(len(ts1_diff[0])):
        if ts1_diff[u][i]!=0:
            list1_org.append(round(MF_ALS_diff.predict(u,i),2))
            list2_pred.append(ts1_diff[u][i])
            predictions_diff[u][i]=str(ts1_diff[u][i])+' | '+str(round(MF_ALS_diff.predict(u,i),2))
print("TEST DATA (50 RATINGS) MSE "+str(mean_squared_error(list2_pred,list1_org)))
```

```
TEST DATA (50 RATINGS) MSE 23.201428
```

In [74]:

```python
predictions_diff
```

Out[74]:

```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '6 | 0.7'],
       [0, 0, 0, '7 | 0.74', 0, 0, 0, 0, 0, 0, 0, 0, 0, '4 | 1.45',
        '7 | 2.09', 0, 0, 0, 0, 0],
       [0, 0, 0, '4 | -0.02', 0, 0, 0, 0, 0, '7 | 0.27', '4 | 2.08', 0,
        0, 0, 0, 0, 0, '4 | 2.1', 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '7 | 0.55',
        0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, '2 | -1.29', 0, 0, '7 | 0.05',
        '4 | -1.45', 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | -0.41', 0, 0, 0, 0, '6 | -0.25',
        '7 | 1.56', 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | 1.07', 0, 0, 0, 0, 0, 0, 0, 0, '4 | 1.63',
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | -0.5', 0, 0, 0, 0, 0, '7 | 1.86',
        '7 | 3.22', 0, 0, 0, 0, 0, '4 | -0.12'],
       [0, 0, 0, 0, 0, '4 | 2.3', 0, 0, 0, 0, 0, 0, '4 | 0.67', 0, 0,
        '5 | 2.05', 0, 0, 0, 0],
       [0, 0, '7 | 1.28', 0, 0, 0, 0, 0, 0, 0, '4 | 0.29', 0, 0, 0, 0, 0,
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '4 | 0.19'],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '7 | 0.52',
        0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, '2 | 0.5', 0, 0, 0, '4 | 1.92', 0, 0, 0, 0, '4 | 1.04',
        0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, '4 | -0.58', 0, 0, 0, 0, 0, '7 | 0.13',
        '4 | 1.16', 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | 0.79', 0, 0, 0, '4 | 0.35', 0, 0,
        '4 | 2.78', 0, 0, '4 | 2.35', 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, '3 | 0.16', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '7 | 1.46',
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '7 | -0.49'],
       [0, 0, 0, 0, '7 | 0.91', 0, 0, 0, 0, 0, 0, 0, '6 | 0.1', 0, 0,
        '4 | 1.53', 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | 0.38', 0, 0, 0, 0, 0, '7 | 1.63',
        '4 | 2.56', 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '7 | 2.15', 0, 0, 0,
```

```
        0],
       [0, 0, '7 | 1.55', 0, 0, 0, 0, 0, '4 | 0.75', 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0]], dtype=object)
```

In [76]:

```
MF_SGD_diff = ExplicitMF(tr1_diff, 2, learning='sgd', verbose=True)
iter_array1 = [1, 5, 50, 100, 200]
MF_SGD_diff.calculate_learning_curve(iter_array1, ts1_diff, learning_rate=0.01)
```

```
Iteration:=1
Train mse: 2.9630623920980397
Test  mse: 2.559754127534314
Iteration:=5
Train mse: 2.4458255552391512
Test  mse: 2.555971864733458
Iteration:=50
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
Train mse: 0.8973826592227198
Test  mse: 4.104294121745248
Iteration:=100
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
Train mse: 0.5269432350893525
Test  mse: 5.718589400935397
Iteration:=200
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
 current iteration:60
 current iteration:70
 current iteration:80
 current iteration:90
 current iteration:100
Train mse: 0.3049938908415537
Test  mse: 9.132373663329593
```

In [77]:

```
list1_org=[]
list2_pred=[]
sgd_predictions_diff = np.zeros((len(ts1_diff), len(ts1_diff[0])),dtype=object)
ts1_diff_pred=ts1_diff.copy();
for u in range(len(ts1_diff)):
    for i in range(len(ts1_diff[0])):
        if ts1_diff[u][i]!=0:
            list1_org.append(round(MF_SGD_diff.predict(u,i),2))
            list2_pred.append(ts1_diff[u][i])
            sgd_predictions_diff[u][i]=str(ts1_diff[u][i])+' | '+str(round(MF_SGD_diff.predict(u,i)
2))
print("TEST DATA (50 RATINGS) MSE "+str(mean_squared_error(list2_pred,list1_org)))
```

```
TEST DATA (50 RATINGS) MSE 9.132591999999999
```

In [78]:

```
sgd_predictions_diff
```

Out[78]:

```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '6 | -0.15'],
       [0, 0, 0, '7 | 8.83', 0, 0, 0, 0, 0, 0, 0, 0, 0, '4 | 6.9',
        '7 | 6.22', 0, 0, 0, 0, 0],
       [0, 0, 0, '4 | 7.44', 0, 0, 0, 0, 0, '7 | 2.93', '4 | -0.74', 0,
```

```
       [0, 0, 0, '4 | 7.44', 0, 0, 0, 0, 0, '7 | 2.95', '4 | -0.74', 0,
        0, 0, 0, 0, 0, '4 | 4.53', 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '7 | 0.66',
        0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, '2 | 9.91', 0, 0, '7 | 9.71',
        '4 | 10.55', 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | 7.46', 0, 0, 0, 0, '6 | 3.17', '7 | 5.24',
        0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | 6.39', 0, 0, 0, 0, 0, 0, 0, 0, '4 | 5.56',
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | 8.24', 0, 0, 0, 0, 0, '7 | 7.05',
        '7 | 7.45', 0, 0, 0, 0, 0, '4 | 5.15'],
       [0, 0, 0, 0, 0, '4 | 4.44', 0, 0, 0, 0, 0, 0, '4 | 5.6', 0, 0,
        '5 | 5.5', 0, 0, 0, 0],
       [0, 0, '7 | 9.51', 0, 0, 0, 0, 0, 0, 0, '4 | -2.29', 0, 0, 0, 0,
        0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '4 | 1.28'],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '7 | 5.41',
        0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, '2 | 4.79', 0, 0, 0, '4 | 3.42', 0, 0, 0, 0, '4 | 9.26',
        0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, '4 | 4.91', 0, 0, 0, 0, 0, '7 | 6.73',
        '4 | 5.85', 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | 7.65', 0, 0, 0, '4 | 2.55', 0, 0,
        '4 | 6.64', 0, 0, '4 | 6.08', 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, '3 | 6.11', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '7 | 5.78',
        0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        '7 | 3.31'],
       [0, 0, 0, 0, '7 | 4.39', 0, 0, 0, 0, 0, 0, 0, '6 | 7.33', 0, 0,
        '4 | 6.4', 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, '7 | 6.22', 0, 0, 0, 0, 0, '7 | 6.62',
        '4 | 6.71', 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '7 | 3.05', 0, 0, 0,
        0],
       [0, 0, '7 | 4.84', 0, 0, 0, 0, 0, '4 | 1.79', 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0]], dtype=object)
```

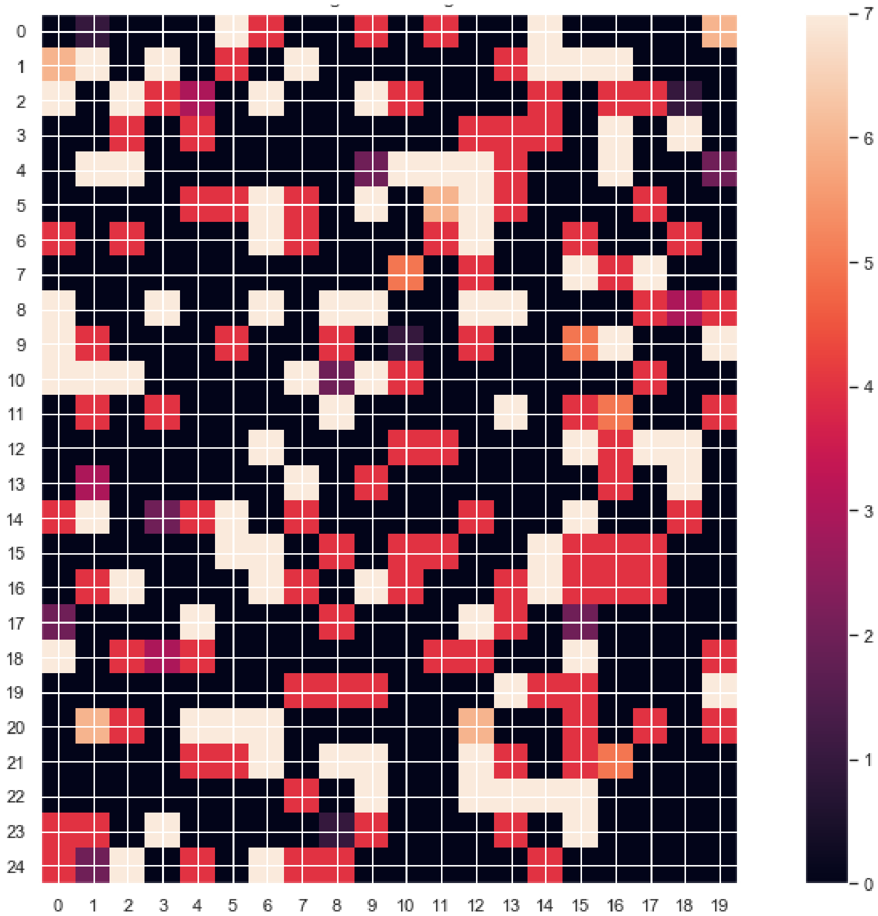
Observations:

```
* Here we have considered one of the random case of having more variance when compared to f
irst considered data and MSE has been maximised but there may be many other cases where we
can get more MSE.
* The smaller range in the data_diff i.e., range of ratings is more for two particular
ratings and less for other 5 ratings makes raises the chance of randomly guessing the corre
ct value at a given position in the matrix.
* The more variance indicating the more variation in user ratings given for a particular it
em given by users who are giving same rating for other items( but given different like 1 by
user1 and 7 by user2 for same item). If we can consider the data in this way then we can ge
t most maximised MSE.
* For experimenting purpose which will have higher MSE compared to first considered data is
taken here and shown how MSE can be maximised by randomly selecting data distribution
```

In [80]:

```
%matplotlib inline
plt.figure(figsize=(15,10))
plt.imshow(data_diff)
plt.xticks(range(len(data_diff[0])))
plt.yticks(range(len(data_diff)))
ax = plt.gca()
ax.set_xticklabels(range(len(data_diff[0])))
ax.set_yticklabels(range(len(data_diff)))
plt.title("Distribution of Ratings In Data to get maximised MSE")
plt.colorbar();
```

Distribution of Ratings In Data to get maximised MSE



4) Repeat #3 with the difference that now you use four latent factors. Use the same rating values as used in #3 above. Compare the predicted outcomes and the MSE values with those obtained in #3. Comment on any significant differences observed

In [86]:

```
MF_ALS_diff_4 = ExplicitMF(tr1_diff, 4, learning='als', verbose=True)
iter_array1 = [1, 5, 50, 100, 200]
MF_ALS_diff_4.calculate_learning_curve(iter_array1, ts1_diff, learning_rate=0.01)
```

```
Iteration:=1
Train mse: 14.121328440438328
Test  mse: 30.008752033818908
Iteration:=5
Train mse: 7.197768549810685
Test  mse: 25.898101793434044
Iteration:=50
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
Train mse: 6.563872040029183
Test  mse: 26.640509558479287
Iteration:=100
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
Train mse: 6.558333353486387
Test  mse: 26.68616918831395
Iteration:=200
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
 current iteration:60
 current iteration:70
 current iteration:80
 current iteration:90
```

```
 current iteration:90
 current iteration:100
Train mse: 6.557561177887836
Test  mse: 26.69260718833511
```

In [82]:

```python
list1_org=[]
list2_pred=[]
predictions_diff_4 = np.zeros((len(ts1_diff), len(ts1_diff[0])),dtype=object)
ts1_diff_pred=ts1_diff.copy();
for u in range(len(ts1_diff)):
    for i in range(len(ts1_diff[0])):
        if ts1_diff[u][i]!=0:
            list1_org.append(round(MF_ALS_diff_4.predict(u,i),2))
            list2_pred.append(ts1_diff[u][i])
            predictions_diff_4[u][i]=str(ts1_diff[u][i])+' | '+str(round(MF_ALS_diff_4.predict(u,i)
2))
print("TEST DATA (50 RATINGS) MSE "+str(mean_squared_error(list2_pred,list1_org)))
```

```
TEST DATA (50 RATINGS) MSE 26.69194
```

Observations:

```
    * Here I have observed that while increasing the latent features using ALS method, the
MSE has been increased for same data. But using SGD, there is improvement in MSE i.e., mse
has been decreased.
    * In the models (Like ALS) fitted on the small dataset the RMSE converges towards a
fixed point as rank (Latent features) increases. The training time increases exponentially
together with increased rank.
    * A lower regularization hyperparameter would bring the prediction accuracy to a lower
error for a lower rank which is observed.
    * Here clearly mse has been increased for our data distribution indicating that
accuracy is decreased with increase of rank/latent can be understood that it is due
overfitting.
```

In [83]:

```python
MF_SGD_diff_4 = ExplicitMF(tr1_diff, 4, learning='sgd', verbose=True)
iter_array1 = [1, 5, 50, 100, 200]
MF_SGD_diff_4.calculate_learning_curve(iter_array1, ts1_diff, learning_rate=0.01)
```

```
Iteration:=1
Train mse: 2.77255911429617
Test  mse: 2.5938338994250913
Iteration:=5
Train mse: 2.406851178803119
Test  mse: 2.590133308754732
Iteration:=50
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
Train mse: 0.4929000458553524
Test  mse: 3.9395923003121367
Iteration:=100
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
Train mse: 0.12744352389415836
Test  mse: 5.052623747293263
Iteration:=200
 current iteration:10
 current iteration:20
 current iteration:30
 current iteration:40
 current iteration:50
 current iteration:60
 current iteration:70
```

```
 current iteration:70
 current iteration:80
 current iteration:90
 current iteration:100
Train mse: 0.020419337491665565
Test  mse: 6.032602849472082
```

In [85]:

```
list1_org=[]
list2_pred=[]
sgd_predictions_diff_4 = np.zeros((len(ts1_diff), len(ts1_diff[0])),dtype=object)
ts1_diff_pred=ts1_diff.copy();
for u in range(len(ts1_diff)):
    for i in range(len(ts1_diff[0])):
        if ts1_diff[u][i]!=0:
            list1_org.append(round(MF_SGD_diff_4.predict(u,i),2))
            list2_pred.append(ts1_diff[u][i])
            sgd_predictions_diff_4[u][i]=str(ts1_diff[u][i])+' | '+str(round(MF_SGD_diff_4.predict(
u,i),2))
print("TEST DATA (50 RATINGS) MSE "+str(mean_squared_error(list2_pred,list1_org)))
```

```
TEST DATA (50 RATINGS) MSE 6.03285
```

Observation:

```
*  Here I have observed that SGD tends to overfit more than ALS and is more susceptible to
popularity bias as mse for 4 latents features using sgd is decreased compared with sgd of 2
latent features.
```

In [1]:

```
import pandas as pd
from sklearn.metrics import mean_squared_error
```

In [2]:

```
df=pd.read_excel('SVD-Data.xlsx',index_col=0)
```

In [3]:

```
df
```

Out[3]:

	data	program	complexity	concept	algorithm	processor	game	gene	drug	disease	...	candidate	optimal	result	training
doc1	6	12	3	4	8	2	2	0	0	0	...	1	2	3	10
doc2	12	5	5	0	15	2	0	0	0	0	...	0	6	4	12
doc3	10	8	4	6	12	4	9	0	0	0	...	8	8	5	16
doc4	16	12	7	1	15	6	6	0	0	0	...	4	12	6	9
doc5	18	4	4	0	10	4	8	0	0	0	...	0	6	8	0
doc6	1	0	4	0	0	0	0	8	12	16	...	8	0	0	0
doc7	4	0	0	0	0	0	0	12	15	8	...	8	0	0	0
doc8	1	0	2	0	0	0	0	16	4	10	...	4	2	12	0
doc9	0	0	0	0	0	0	0	14	6	8	...	5	0	8	0
doc10	2	1	4	1	0	0	0	15	9	12	...	6	0	4	0
doc11	4	0	5	2	0	0	0	10	7	11	...	4	1	6	0
doc12	5	6	0	0	0	0	2	0	0	0	...	15	1	9	8
doc13	0	5	0	0	0	0	0	0	0	0	...	8	0	7	4
doc14	3	4	1	0	0	0	0	0	10	0	...	6	0	8	0
doc15	2	10	0	0	0	0	2	0	8	0	...	8	0	6	0
doc16	0	5	1	1	0	0	0	0	6	0	...	4	0	5	0
doc17	0	0	2	2	0	0	3	0	4	3	...	7	0	9	0
doc18	2	1	2	0	0	0	15	0	0	0	...	0	6	10	12
doc19	0	0	3	0	0	0	12	0	2	0	...	2	4	6	10
doc20	1	4	0	3	0	0	9	0	4	0	...	0	2	9	12
doc21	1	4	3	1	1	0	12	0	2	0	...	0	6	9	9
doc22	0	0	0	5	0	0	8	0	0	0	...	0	8	10	14
doc23	0	0	0	0	1	0	14	0	1	1	...	0	0	8	11
doc24	10	0	0	10	3	0	12	0	0	0	...	0	0	0	0
doc25	12	1	2	7	3	0	19	0	0	0	...	0	0	0	0
doc26	11	2	1	8	4	0	18	0	0	0	...	0	4	4	0

26 rows × 22 columns

Note: As given dataset/ matrix is not square matrix, so we need to do SVD decomposition to get eigen vectors and eigen values. If we consider given matrix as A.

Then svd gives A=U.S.transpose(V).

```
* The eigenvectors of A.transpose(A)  make up the columns of U.
* The eigenvectors of transpose(A).A make up the columns of V.
* Also, the singular values in S are square roots of eigenvalues from A.transpose(A) or
```

transpose(A).A.
    * The singular values are the diagonal entries of the S matrix and are arranged in desc
ending order. The singular values are always real numbers. If the matrix A is a real matrix
, then U and V are also real.

5a) Show all the Eigen-values and Eigen-vectors obtained after the decomposition

In [4]:

```
import numpy as np
U, s, Vh = np.linalg.svd(df) ## considering transpose(v) variable as Vh.
```

In [5]:

```
print("The eigenvectors of A.transpose(A) are the columns of U matrix = \n")
print( np.round(U, decimals=2))
```

```
The eigenvectors of A.transpose(A) are the columns of U matrix =

[[-0.19  0.02  0.14 -0.26 -0.15 -0.09  0.29 -0.51  0.16 -0.1  -0.04  0.37
   0.08  0.07 -0.28 -0.16  0.05  0.17 -0.29 -0.   -0.04 -0.1   0.   -0.06
  -0.25  0.09]
 [-0.19 -0.02  0.2  -0.37 -0.17  0.01  0.12  0.02 -0.5   0.17 -0.02 -0.12
  -0.17  0.33  0.21  0.21  0.04  0.32  0.06 -0.15 -0.25  0.15  0.02 -0.02
   0.08 -0.04]
 [-0.24 -0.11  0.18 -0.27 -0.06 -0.45 -0.38  0.03  0.17  0.04 -0.04  0.15
  -0.13  0.05  0.25 -0.28 -0.09 -0.29  0.07  0.11 -0.19 -0.14  0.01  0.27
   0.16 -0.04]
 [-0.27 -0.06  0.24 -0.42  0.02  0.01  0.01  0.28  0.19 -0.03  0.14  0.01
   0.23 -0.19  0.    0.3   0.26 -0.12  0.03  0.04  0.47  0.13  0.04 -0.2
   0.07  0.11]
 [-0.21 -0.03  0.22 -0.25  0.2   0.52  0.11  0.14 -0.14 -0.11 -0.06 -0.26
  -0.1  -0.18 -0.15 -0.3  -0.25 -0.24 -0.06  0.04 -0.1  -0.21 -0.07  0.06
  -0.18 -0.15]
 [-0.11  0.33  0.18  0.17  0.06 -0.34  0.18  0.13 -0.28 -0.22  0.39  0.04
  -0.04 -0.22 -0.31 -0.15  0.23 -0.08  0.23 -0.12 -0.21 -0.06  0.04 -0.12
   0.13 -0.06]
 [-0.1   0.34  0.2   0.12  0.13 -0.31  0.22  0.09  0.    0.04 -0.56 -0.04
  -0.2  -0.32  0.17  0.13 -0.16  0.14 -0.04 -0.08  0.19  0.01 -0.17  0.08
  -0.15  0.01]
 [-0.16  0.34  0.19  0.13 -0.15  0.43 -0.22 -0.1   0.13 -0.   -0.19  0.31
  -0.1  -0.01 -0.12  0.29  0.1   0.08  0.29  0.18 -0.08 -0.17  0.19  0.14
   0.23 -0.1 ]
 [-0.11  0.31  0.15  0.15 -0.1   0.19 -0.14 -0.22  0.11  0.02 -0.12 -0.24
   0.09 -0.    0.3  -0.28  0.28 -0.18 -0.24 -0.09 -0.16  0.28 -0.12 -0.33
   0.18  0.19]
 [-0.11  0.32  0.17  0.16  0.03 -0.09 -0.1   0.06  0.12 -0.    0.22 -0.13
   0.28  0.42  0.21  0.18 -0.13 -0.07 -0.07 -0.08  0.04 -0.3   0.09 -0.1
  -0.45 -0.24]
 [-0.11  0.26  0.18  0.1   0.03  0.04 -0.11  0.06 -0.05  0.15  0.3  -0.01
   0.05  0.16 -0.22 -0.15 -0.36  0.18 -0.17  0.14  0.26  0.34 -0.09  0.38
   0.13  0.29]
 [-0.26  0.09 -0.33 -0.14  0.04 -0.15 -0.45 -0.06 -0.09 -0.3  -0.22 -0.26
   0.05  0.01 -0.33  0.12 -0.19  0.12  0.1   0.06 -0.1   0.04  0.05 -0.26
  -0.12  0.24]
 [-0.2   0.12 -0.32 -0.09 -0.01  0.07 -0.11 -0.27 -0.19 -0.18  0.04  0.04
   0.28 -0.14  0.18 -0.16  0.1   0.11  0.16 -0.22  0.28  0.11 -0.23  0.3
   0.02 -0.44]
 [-0.19  0.14 -0.3  -0.03  0.2  -0.02  0.18  0.37  0.02  0.36 -0.32  0.09
   0.27  0.3  -0.17 -0.32  0.26  0.07  0.03  0.13 -0.04 -0.07  0.08 -0.01
   0.07 -0.03]
 [-0.2   0.14 -0.29 -0.1   0.12 -0.06  0.25 -0.11  0.42  0.01  0.2  -0.41
  -0.27 -0.01 -0.06  0.26  0.12  0.01 -0.22  0.03 -0.18 -0.    0.06  0.28
   0.19 -0.14]
 [-0.18  0.14 -0.29 -0.05  0.08  0.1   0.3  -0.14 -0.07  0.09  0.18  0.14
   0.13 -0.12  0.38  0.06 -0.27 -0.15  0.28 -0.01 -0.07 -0.19  0.08 -0.01
   0.02  0.51]
 [-0.18  0.13 -0.29  0.01  0.16  0.09 -0.22  0.12 -0.18  0.14  0.19  0.42
  -0.49 -0.04  0.08  0.05  0.05 -0.1  -0.37 -0.03  0.11  0.03 -0.05 -0.23
  -0.15 -0.08]
 [-0.28 -0.22 -0.02  0.25 -0.16  0.08  0.01  0.3   0.03 -0.2  -0.11  0.18
   0.19  0.01 -0.08  0.18 -0.03 -0.11 -0.26 -0.56 -0.21  0.01 -0.04  0.25
```

```
  0.06  0.16]
 [-0.23 -0.14 -0.03  0.22 -0.15 -0.04  0.13  0.12 -0.16 -0.25  0.    0.03
  0.18 -0.2   0.24 -0.   -0.05  0.11 -0.25  0.57 -0.13  0.2   0.34  0.02
 -0.06 -0.16]
 [-0.24 -0.1  -0.04  0.15 -0.21 -0.05  0.21 -0.14  0.09  0.17 -0.08 -0.01
 -0.13  0.13 -0.16 -0.04 -0.38 -0.32  0.21 -0.15  0.19  0.31  0.24 -0.28
  0.13 -0.29]
 [-0.23 -0.14 -0.01  0.12 -0.15  0.05  0.07  0.2   0.32  0.01  0.16  0.08
 -0.06 -0.    0.04 -0.04 -0.11  0.29  0.25  0.16 -0.2   0.05 -0.64 -0.23
 -0.08 -0.08]
 [-0.22 -0.16  0.01  0.18 -0.31 -0.05 -0.17 -0.11 -0.13  0.55  0.1  -0.25
  0.08 -0.37 -0.11 -0.02  0.05  0.16 -0.12 -0.04  0.1  -0.39  0.04 -0.05
  0.01  0.02]
 [-0.23 -0.17 -0.02  0.25 -0.18  0.01  0.1  -0.05 -0.12 -0.23 -0.05 -0.18
 -0.35  0.32  0.02 -0.1   0.35 -0.13  0.16  0.14  0.37 -0.14 -0.07  0.18
 -0.17  0.26]
 [-0.13 -0.19  0.11  0.16  0.39 -0.07 -0.01 -0.31 -0.23  0.22 -0.06  0.04
  0.2   0.05 -0.15  0.36  0.1  -0.36  0.01  0.23 -0.16  0.14 -0.33  0.08
 -0.05 -0.03]
 [-0.16 -0.22  0.14  0.17  0.46 -0.02  0.   -0.13 -0.01 -0.21  0.    0.01
 -0.    0.16  0.11 -0.05 -0.13  0.3  -0.12 -0.05  0.18 -0.31  0.02 -0.24
  0.5  -0.05]
 [-0.17 -0.21  0.15  0.13  0.39  0.06 -0.13 -0.08  0.2   0.13  0.08  0.
 -0.1  -0.12  0.07 -0.14  0.16  0.27  0.27 -0.22 -0.11  0.29  0.35  0.09
 -0.37  0.07]]
```

In [6]:

```
print("The singular values in s are square roots of eigenvalues from A.transpose(A) or transpose(A
).A = \n")
print( np.round(s, decimals=2))
```

```
The singular values in s are square roots of eigenvalues from A.transpose(A) or transpose(A).A =

[97.11 58.25 51.3  41.7  31.59 18.23 15.2  13.69 10.86 10.45 10.01  8.88
  8.21  7.37  6.42  5.77  4.2   3.31  2.59  2.36  1.88  1.21]
```

In [7]:

```
eigen_values=s**2
print("The eigenvalues from A.transpose(A) or transpose(A).A = \n")
print( np.round(eigen_values, decimals=2))
```

```
The eigenvalues from A.transpose(A) or transpose(A).A =

[9.43124e+03 3.39297e+03 2.63214e+03 1.73927e+03 9.98230e+02 3.32360e+02
 2.31010e+02 1.87280e+02 1.17990e+02 1.09260e+02 1.00190e+02 7.88800e+01
 6.73700e+01 5.43500e+01 4.12200e+01 3.33100e+01 1.76400e+01 1.09900e+01
 6.71000e+00 5.58000e+00 3.54000e+00 1.46000e+00]
```

In [8]:

```
S= np.diag(s)
tmp=np.zeros((4,22))
S=np.concatenate((S,tmp))
print("Matrix S is given by= \n")
print( np.round(S, decimals=2))
```

```
Matrix S is given by=

[[97.11  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.   58.25  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.   51.3   0.    0.    0.    0.    0.    0.    0.    0.    0.
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.   41.7   0.    0.    0.    0.    0.    0.    0.    0.
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.   31.59  0.    0.    0.    0.    0.    0.    0.
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.   18.23  0.    0.    0.    0.    0.    0.
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
```

```
 [ 0.    0.    0.    0.    0.    0.   15.2   0.    0.    0.    0.    0.  
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.   13.69  0.    0.    0.    0.  
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.   10.86  0.    0.    0.  
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.   10.45  0.    0.  
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.   10.01  0.  
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    8.88
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  
   8.21  0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  
   0.    7.37  0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  
   0.    0.    6.42  0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  
   0.    0.    0.    5.77  0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  
   0.    0.    0.    0.    4.2   0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  
   0.    0.    0.    0.    0.    3.31  0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  
   0.    0.    0.    0.    0.    0.    2.59  0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  
   0.    0.    0.    0.    0.    0.    0.    2.36  0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  
   0.    0.    0.    0.    0.    0.    0.    0.    1.88  0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  
   0.    0.    0.    0.    0.    0.    0.    0.    0.    1.21]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]]
```

In [9]:

```
print("The eigenvectors of transpose(A).A are the rows of Vh matrix here =  \n")
print( np.round(Vh, decimals=2))
```

```
The eigenvectors of transpose(A).A are the rows of Vh matrix here = 

[[-0.25 -0.19 -0.1  -0.1  -0.16 -0.04 -0.33 -0.09 -0.14 -0.09 -0.13 -0.12
  -0.18 -0.16 -0.33 -0.3  -0.14 -0.35 -0.34 -0.29 -0.2  -0.21]
 [-0.09  0.01  0.04 -0.1  -0.09 -0.02 -0.38  0.41  0.34  0.36  0.14  0.12
   0.27 -0.1   0.08 -0.2   0.12 -0.13  0.    0.1   0.38 -0.26]
 [ 0.31 -0.02  0.12  0.09  0.26  0.07  0.16  0.26  0.02  0.21 -0.37 -0.34
  -0.12  0.15 -0.07  0.07 -0.37 -0.21 -0.22 -0.1   0.36  0.07]
 [-0.33 -0.34 -0.09  0.06 -0.43 -0.14  0.33  0.25  0.18  0.23 -0.1  -0.09
  -0.07 -0.14  0.08 -0.08 -0.08  0.2  -0.11  0.03 -0.07  0.43]
 [ 0.48  0.    0.01  0.24  0.05  0.    0.29 -0.03  0.14  0.02  0.21  0.16
   0.13 -0.14 -0.2  -0.56  0.15  0.14 -0.21 -0.24 -0.07  0.02]
 [ 0.13 -0.17 -0.   -0.19 -0.03  0.01  0.02  0.11 -0.35 -0.14 -0.03  0.01
  -0.44  0.03  0.47 -0.49  0.03  0.02  0.16  0.19  0.17 -0.09]
 [-0.04  0.25  0.02 -0.19  0.04 -0.01 -0.06 -0.27  0.53 -0.11 -0.09 -0.09
  -0.45 -0.16 -0.32 -0.15 -0.    0.1   0.2   0.18  0.23  0.16]
 [ 0.06 -0.27  0.27 -0.49  0.05  0.1  -0.   -0.09  0.33  0.11  0.07 -0.14
   0.03  0.39  0.13 -0.04 -0.12  0.23  0.06 -0.31 -0.29 -0.16]
 [-0.21  0.65 -0.08  0.05 -0.23  0.05  0.28  0.24  0.14 -0.2  -0.12 -0.05
   0.03  0.16  0.19 -0.17 -0.19 -0.05  0.01 -0.13 -0.17 -0.27]
 [ 0.03 -0.1  -0.07  0.53  0.09 -0.03 -0.32  0.04  0.37 -0.11  0.18  0.03
  -0.33  0.29  0.33  0.05  0.05 -0.17 -0.12 -0.06 -0.18  0.14]
 [-0.17  0.15  0.44  0.15  0.06  0.03  0.07 -0.2  -0.1   0.53 -0.21  0.23
  -0.13  0.17 -0.05 -0.18  0.11 -0.14 -0.05  0.31 -0.28 -0.02]
 [-0.36  0.01  0.24  0.28  0.06  0.01  0.03 -0.07 -0.15  0.1   0.02  0.02
  -0.22 -0.02  0.    0.05  0.17  0.31  0.12 -0.57  0.38 -0.16]
 [ 0.06  0.21  0.14  0.06 -0.14  0.03 -0.36  0.19 -0.21 -0.06  0.09 -0.38
  -0.01  0.2  -0.18 -0.05  0.11  0.54 -0.33  0.23 -0.05  0.07]
 [ 0.1   0.08  0.18  0.01  0.26 -0.12 -0.04  0.31 -0.    0.13  0.15 -0.13
   0.25  0.62  0.09  0.17  0.14  0.11  0.11  0.02  0.4   0.16]
```

```
 -0.25 -0.62  0.09  0.17 -0.14  0.11  0.11 -0.02 -0.4  -0.16]
 [-0.25 -0.22  0.03 -0.08  0.4   0.04  0.25  0.41  0.12 -0.34 -0.16  0.14
  -0.08  0.12 -0.23  0.04  0.39  0.02 -0.17  0.16 -0.07 -0.17]
 [ 0.08  0.08  0.01 -0.03  0.04 -0.07 -0.28  0.39 -0.14 -0.03 -0.22  0.3
   0.02  0.18 -0.27 -0.13 -0.09  0.02  0.47 -0.25 -0.16  0.39]
 [-0.38  0.16 -0.21 -0.17  0.53  0.1   0.04 -0.02 -0.12  0.14  0.43 -0.05
   0.07  0.06  0.05 -0.23 -0.23 -0.02 -0.14 -0.02  0.04  0.34]
 [ 0.05  0.09  0.33 -0.17 -0.18 -0.57  0.19  0.11 -0.09 -0.08  0.45 -0.05
  -0.14  0.19 -0.16  0.11  0.08 -0.33 -0.04 -0.04  0.11  0.08]
 [ 0.03  0.05 -0.36  0.02  0.09 -0.11  0.06 -0.01 -0.03  0.29 -0.07 -0.56
   0.02  0.09 -0.03 -0.08  0.52 -0.16  0.31 -0.09 -0.15 -0.03]
 [-0.16 -0.13  0.51  0.21  0.01  0.23 -0.01 -0.02  0.04 -0.35  0.05 -0.37
   0.34 -0.15  0.04 -0.22 -0.02 -0.22  0.26  0.08 -0.01  0.16]
 [ 0.1   0.17  0.11 -0.26 -0.2   0.59  0.01  0.12 -0.04  0.04  0.08  0.07
  -0.19 -0.14  0.08  0.18  0.37 -0.24 -0.2  -0.21 -0.01  0.3 ]
 [-0.    0.18  0.14 -0.19  0.19 -0.43 -0.16 -0.14  0.07 -0.13 -0.4   0.02
   0.21 -0.17  0.39 -0.05  0.24  0.04 -0.29 -0.17  0.02  0.25]]
```

5b) How many, and which Eigen values (and vectors) would you select if you should preserve 85% of the information?

In [10]:

```
eigen_values=s**2
percent_contrib=eigen_values/sum(eigen_values)
print("The percent of the each eigen value contributes to valuable information = \n")
print( np.round(percent_contrib, decimals=2))
```

```
The percent of the each eigen value contributes to valuable information =

[0.48 0.17 0.13 0.09 0.05 0.02 0.01 0.01 0.01 0.01 0.01 0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.  ]
```

In [11]:

```
sum(percent_contrib[:4])
```

Out[11]:

```
0.8776413228945918
```

Note: Here first 4 eigen values contribute to about 87.76%. So we will consider first four eigen values and corresponding eigen vectors in U and Vh to preserve 85% of the information

In [12]:

```
new_eigen_values=eigen_values[:4]
print("The new eigenvalues to preserve about 85% information are = \n")
print( np.round(new_eigen_values, decimals=2))
```

```
The new eigenvalues to preserve about 85% information are =

[9431.24 3392.97 2632.14 1739.27]
```

After considering first 4 eigen values, the new eigen vectors (new doc-doc similarity matrix, term-term similarity matrix and concept/ hidden features matrix)

In [13]:

```
new_s=s[:4]
print("The new singular values in s to preserve about 85% information are = \n")
print( np.round(new_s, decimals=2))
new_S= np.diag(new_s)
print("\nNew Matrix new_S is given by= \n")
print( np.round(new_S, decimals=2))
```

```
The new singular values in s to preserve about 85% information are =
```

```
[97.11 58.25 51.3  41.7 ]

New Matrix new_S is given by=

[[97.11  0.    0.    0.  ]
 [ 0.   58.25  0.    0.  ]
 [ 0.    0.   51.3   0.  ]
 [ 0.    0.    0.   41.7 ]]
```

In [14]:

```
new_U=U[:,:4]
print("The new U matrix to preserve about 85% of information is = \n")
print( np.round(new_U, decimals=2))
```

```
The new U matrix to preserve about 85% of information is =

[[-0.19  0.02  0.14 -0.26]
 [-0.19 -0.02  0.2  -0.37]
 [-0.24 -0.11  0.18 -0.27]
 [-0.27 -0.06  0.24 -0.42]
 [-0.21 -0.03  0.22 -0.25]
 [-0.11  0.33  0.18  0.17]
 [-0.1   0.34  0.2   0.12]
 [-0.16  0.34  0.19  0.13]
 [-0.11  0.31  0.15  0.15]
 [-0.11  0.32  0.17  0.16]
 [-0.11  0.26  0.18  0.1 ]
 [-0.26  0.09 -0.33 -0.14]
 [-0.2   0.12 -0.32 -0.09]
 [-0.19  0.14 -0.3  -0.03]
 [-0.2   0.14 -0.29 -0.1 ]
 [-0.18  0.14 -0.29 -0.05]
 [-0.18  0.13 -0.29  0.01]
 [-0.28 -0.22 -0.02  0.25]
 [-0.23 -0.14 -0.03  0.22]
 [-0.24 -0.1  -0.04  0.15]
 [-0.23 -0.14 -0.01  0.12]
 [-0.22 -0.16  0.01  0.18]
 [-0.23 -0.17 -0.02  0.25]
 [-0.13 -0.19  0.11  0.16]
 [-0.16 -0.22  0.14  0.17]
 [-0.17 -0.21  0.15  0.13]]
```

In [15]:

```
new_Vh=Vh[:4,:]
print("The new Vh matrix to preserve about 85% of information is = \n")
print( np.round(new_Vh, decimals=2))
```

```
The new Vh matrix to preserve about 85% of information is =

[[-0.25 -0.19 -0.1  -0.1  -0.16 -0.04 -0.33 -0.09 -0.14 -0.09 -0.13 -0.12
  -0.18 -0.16 -0.33 -0.3  -0.14 -0.35 -0.34 -0.29 -0.2  -0.21]
 [-0.09  0.01  0.04 -0.1  -0.09 -0.02 -0.38  0.41  0.34  0.36  0.14  0.12
   0.27 -0.1   0.08 -0.2   0.12 -0.13  0.    0.1   0.38 -0.26]
 [ 0.31 -0.02  0.12  0.09  0.26  0.07  0.16  0.26  0.02  0.21 -0.37 -0.34
  -0.12  0.15 -0.07  0.07 -0.37 -0.21 -0.22 -0.1   0.36  0.07]
 [-0.33 -0.34 -0.09  0.06 -0.43 -0.14  0.33  0.25  0.18  0.23 -0.1  -0.09
  -0.07 -0.14  0.08 -0.08 -0.08  0.2  -0.11  0.03 -0.07  0.43]]
```

In [16]:

```
A_approx = np.matrix(new_U) * np.diag(new_s) * np.matrix(new_Vh)
print("A calculated using only the first four components:\n")
print(pd.DataFrame(A_approx, index=df.index, columns=df.columns))
print("\nError from actual value:\n")
print(df - A_approx)
```

```
A calculated using only the first four components:

           data     program  complexity   concept  algorithm  processor  \
```

```
doc1   10.300120   7.022448    3.813746  1.648384   9.450226   2.853672
doc2   13.076268   8.582204    4.561587  1.937866  12.442253   3.763198
doc3   12.958480   8.039034    4.366933  3.032609  11.517628   3.366585
doc4   16.405203  10.648158    5.668035  2.909362  15.243391   4.561246
doc5   12.234016   7.185908    4.435423  2.552957  10.866573   3.208450
doc6    1.273464  -0.417309    2.324172  0.284109  -0.677866  -0.189951
doc7    1.941640   0.083289    2.520716  0.142339   0.206869   0.101122
doc8    3.076034   1.030567    3.027475  0.700518   0.809186   0.249627
doc9    1.414800   0.020756    2.234253  0.338633  -0.457975  -0.125770
doc10   1.428594  -0.206353    2.309270  0.390279  -0.543345  -0.157602
doc11   2.565974   0.487866    2.404090  0.548642   0.855827   0.259885
doc12   2.362146   7.236292    1.256543 -0.040207   1.758328   0.640977
doc13   0.359501   5.411364    0.621626 -0.516743  -0.013337   0.118409
doc14  -0.543230   4.376802    0.446796 -0.514415  -1.104304  -0.226864
doc15   0.715160   5.526160    0.843207 -0.573060   0.333394   0.234617
doc16  -0.295363   4.535291    0.501611 -0.594773  -0.730971  -0.103188
doc17  -1.187263   3.575878    0.213426 -0.412654  -1.858795  -0.478660
doc18   4.056492   1.496390    1.279971  4.341577   0.714630  -0.165078
doc19   2.800846   1.094697    1.072465  3.372349  -0.037946  -0.306828
doc20   3.558184   2.280137    1.385339  2.981046   1.037545   0.076213
doc21   4.262023   2.337203    1.458746  3.144487   1.856214   0.310119
doc22   3.746627   1.329337    1.246672  3.452722   1.100047   0.035824
doc23   2.503527   0.460891    0.859026  3.622213  -0.443503  -0.465254
doc24   3.860253  -0.035158    1.087562  3.282838   1.678683   0.209310
doc25   5.084177   0.360496    1.449677  3.899431   2.546137   0.423345
doc26   5.902377   1.083090    1.760523  3.844087   3.444693   0.714930

            game       gene      drug    disease  ...  candidate   optimal  \
doc1    3.187141   1.147217  0.944053   0.862010  ...   3.453027  5.450831
doc2    3.269074  -0.087635 -0.483232  -0.305883  ...   2.911816  6.852879
doc3    7.914111  -0.870532 -0.715058  -0.904753  ...   2.376344  7.345221
doc4    6.149106  -0.434719 -0.621048  -0.625079  ...   3.570824  8.846655
doc5    5.834920   1.533301  0.616918   1.145846  ...   2.706234  6.619606
doc6   -0.349529  13.047064  9.413519  11.340862  ...   5.506069  0.019878
doc7   -1.197036  12.790137  9.061465  11.081701  ...   5.471436  0.261492
doc8    0.869710  13.302294  9.926140  11.571774  ...   6.566350  1.070123
doc9    0.040011  11.887834  8.821873  10.352626  ...   5.473241  0.206598
doc10   0.125102  12.509503  9.139144  10.884427  ...   5.470126  0.181978
doc11   0.411913  10.591517  7.493000   9.175148  ...   4.585172  0.841608
doc12   1.630357  -1.373276  3.891444  -0.825508  ...   8.627948  1.886264
doc13   0.070709  -0.549506  4.003762  -0.129100  ...   7.704019  0.637395
doc14   0.185205   0.671113  4.705636   0.937448  ...   7.545090  0.125542
doc15  -0.401097   0.178823  4.388069   0.482212  ...   7.929699  0.720219
doc16  -0.260059   0.482424  4.452769   0.753988  ...   7.420925  0.200381
doc17   0.700141   0.874433  4.681924   1.118369  ...   7.024784 -0.168858
doc18  16.843338  -0.495801  1.189328  -0.182852  ...   1.020234  3.959134
doc19  13.255578   0.785225  2.074909   0.910705  ...   1.645591  2.895439
doc20  11.573313   0.647752  2.210432   0.776685  ...   2.539859  3.128174
doc21  11.846837  -0.176914  1.182460   0.013818  ...   1.635260  3.504067
doc22  13.074189   0.065513  1.052379   0.223968  ...   0.812270  3.333602
doc23  14.274993   0.366581  1.552415   0.545372  ...   0.823655  2.850151
doc24  11.610740  -0.133451 -0.612802  -0.091370  ...  -1.681035  3.099675
doc25  13.633923  -0.312624 -0.853470  -0.254725  ...  -1.893135  3.920761
doc26  13.152024  -0.269393 -0.802060  -0.235684  ...  -1.493385  4.280319

          result   training   campaign        win        run      best  \
doc1    4.652360   6.739441   0.880309   2.437138   5.762861  4.196929
doc2    4.126902   7.963891  -0.011752   1.444770   5.817767  3.715825
doc3    5.772223   9.998419   0.183787   4.862840   7.283962  4.876566
doc4    6.055297  10.972770   0.238122   3.464103   8.145103  5.324335
doc5    5.067154   8.254079  -0.602778   2.907943   5.660961  4.290761
doc6    4.842404  -0.777998  -0.189492   0.387930   0.667979  4.228740
doc7    4.408933  -0.844391  -0.385219  -0.445992   0.455010  3.901626
doc8    6.418701   0.893170   0.570378   1.819958   2.520896  5.646614
doc9    5.030344  -0.239313   0.355300   1.055428   1.375648  4.428611
doc10   5.042708  -0.369075   0.036895   0.876829   1.085541  4.401187
doc11   4.332169   0.338080  -0.414887   0.477902   1.033873  3.750722
doc12   9.414850   5.961711  10.958255  10.550615  13.111630  9.264392
doc13   7.781777   3.820269   9.847181   8.654653  10.748731  7.776007
doc14   7.650588   3.107109   9.310794   8.405682   9.911479  7.575751
doc15   7.753822   3.646434   9.604757   8.133182  10.513270  7.764303
doc16   7.320944   2.998520   9.134116   7.905675   9.679297  7.305207
doc17   7.374234   2.795290   8.799160   8.376263   9.273793  7.234810
doc18   8.744053   9.832426   1.851072  13.264966   8.316062  6.851482
doc19   7.692608   7.640600   2.033149  11.052027   7.063902  6.144721
doc20   7.798847   7.618890   2.837414  10.508854   7.727918  6.442708
doc21   7.031390   7.837698   1.933318   9.805060   7.063979  5.685098
```

```
doc22  6.776323   7.751359   1.048334   9.946151   6.277384  5.283655
doc23  7.362155   7.729375   1.401567  11.209625   6.547552  5.722807
doc24  3.599661   6.056448  -2.149955   6.073779   2.446036  2.289146
doc25  4.300740   7.408212  -2.512594   7.096048   3.085770  2.767608
doc26  4.457787   7.680381  -2.328862   6.865671   3.454051  2.973423

       expression      sport
doc1     7.446118  -0.758256
doc2     8.104615  -1.763957
doc3     6.533272   2.277354
doc4     9.548705  -0.453435
doc5     8.417171   0.975983
doc6    12.265189   0.571664
doc7    12.616995  -0.429765
doc8    13.573200   1.005491
doc9    11.348171   0.727917
doc10   11.843906   0.871891
doc11   10.826523   0.542325
doc12    1.474125   0.081443
doc13    0.989356  -0.459413
doc14    1.359736   0.103266
doc15    1.916960  -0.918144
doc16    1.419260  -0.389234
doc17    0.948081   0.822499
doc18   -0.504757  13.264749
doc19    0.437238  10.677361
doc20    1.228857   8.887846
doc21    0.819113   8.826596
doc22    0.352808  10.143712
doc23   -0.389751  11.658569
doc24    0.115813   8.906005
doc25    0.371320  10.246424
doc26    1.067460   9.530854

[26 rows x 22 columns]

Error from actual value:

           data    program  complexity   concept  algorithm  processor  \
doc1  -4.300120  4.977552   -0.813746  2.351616  -1.450226  -0.853672
doc2  -1.076268 -3.582204    0.438413 -1.937866   2.557747  -1.763198
doc3  -2.958480 -0.039034   -0.366933  2.967391   0.482372   0.633415
doc4  -0.405203  1.351842    1.331965 -1.909362  -0.243391   1.438754
doc5   5.765984 -3.185908   -0.435423 -2.552957  -0.866573   0.791550
doc6  -0.273464  0.417309    1.675828 -0.284109   0.677866   0.189951
doc7   2.058360 -0.083289   -2.520716 -0.142339  -0.206869  -0.101122
doc8  -2.076034 -1.030567   -1.027475 -0.700518  -0.809186  -0.249627
doc9  -1.414800 -0.020756   -2.234253 -0.338633   0.457975   0.125770
doc10  0.571406  1.206353    1.690730  0.609721   0.543345   0.157602
doc11  1.434026 -0.487866    2.595910  1.451358  -0.855827  -0.259885
doc12  2.637854 -1.236292   -1.256543  0.040207  -1.758328  -0.640977
doc13 -0.359501 -0.411364   -0.621626  0.516743   0.013337  -0.118409
doc14  3.543230 -0.376802    0.553204  0.514415   1.104304   0.226864
doc15  1.284840  4.473840   -0.843207  0.573060  -0.333394  -0.234617
doc16  0.295363  0.464709    0.498389  1.594773   0.730971   0.103188
doc17  1.187263 -3.575878    1.786574  2.412654   1.858795   0.478660
doc18 -2.056492 -0.496390    0.720029 -4.341577  -0.714630   0.165078
doc19 -2.800846 -1.094697    1.927535 -3.372349   0.037946   0.306828
doc20 -2.558184  1.719863   -1.385339  0.018954  -1.037545  -0.076213
doc21 -3.262023  1.662797    1.541254 -2.144487  -0.856214  -0.310119
doc22 -3.746627 -1.329337   -1.246672  1.547278  -1.100047  -0.035824
doc23 -2.503527 -0.460891   -0.859026 -3.622213   1.443503   0.465254
doc24  6.139747  0.035158   -1.087562  6.717162   1.321317  -0.209310
doc25  6.915823  0.639504    0.550323  3.100569   0.453863  -0.423345
doc26  5.097623  0.916910   -0.760523  4.155913   0.555307  -0.714930

           game      gene      drug   disease  ...  candidate   optimal  \
doc1  -1.187141 -1.147217 -0.944053 -0.862010  ...  -2.453027 -3.450831
doc2  -3.269074  0.087635  0.483232  0.305883  ...  -2.911816 -0.852879
doc3   1.085889  0.870532  0.715058  0.904753  ...   5.623656  0.654779
doc4  -0.149106  0.434719  0.621048  0.625079  ...   0.429176  3.153345
doc5   2.165080 -1.533301 -0.616918 -1.145846  ...  -2.706234 -0.619606
doc6   0.349529 -5.047064  2.586481  4.659138  ...   2.493931 -0.019878
doc7   1.197036 -0.790137  5.938535 -3.081701  ...   2.528564 -0.261492
doc8  -0.869710  2.697706 -5.926140 -1.571774  ...  -2.566350  0.929877
doc9  -0.040011  2.112166 -2.821873 -2.352626  ...  -0.473241 -0.206598
doc10 -0.125102  2.490497 -0.139144  1.115573  ...   0.529874 -0.181978
```

```
doc11 -0.411913 -0.591517 -0.493000  1.824852  ...  -0.585172  0.158392
doc12  0.369643  1.373276 -3.891444  0.825508  ...   6.372052 -0.886264
doc13 -0.070709  0.549506 -4.003762  0.129100  ...   0.295981 -0.637395
doc14 -0.185205 -0.671113  5.294364 -0.937448  ...  -1.545090 -0.125542
doc15  2.401097 -0.178823  3.611931 -0.482212  ...   0.070301 -0.720219
doc16  0.260059 -0.482424  1.547231 -0.753988  ...  -3.420925 -0.200381
doc17  2.299859 -0.874433 -0.681924  1.881631  ...  -0.024784  0.168858
doc18 -1.843338  0.495801 -1.189328  0.182852  ...  -1.020234  2.040866
doc19 -1.255578 -0.785225 -0.074909 -0.910705  ...   0.354409  1.104561
doc20 -2.573313 -0.647752  1.789568 -0.776685  ...  -2.539859 -1.128174
doc21  0.153163  0.176914  0.817540 -0.013818  ...  -1.635260  2.495933
doc22 -5.074189 -0.065513 -1.052379 -0.223968  ...  -0.812270  4.666398
doc23 -0.274993 -0.366581 -0.552415  0.454628  ...  -0.823655 -2.850151
doc24  0.389260  0.133451  0.612802  0.091370  ...   1.681035 -3.099675
doc25  5.366077  0.312624  0.853470  0.254725  ...   1.893135 -3.920761
doc26  4.847976  0.269393  0.802060  0.235684  ...   1.493385 -0.280319

         result  training  campaign       win       run      best  expression  \
doc1  -1.652360  3.260559 -0.880309 -0.437138  1.237139  1.803071    4.553882
doc2  -0.126902  4.036109  0.011752 -1.444770  2.182233  2.284175   -0.104615
doc3  -0.772223  6.001581 -0.183787 -0.862840 -2.283962 -2.876566   -2.533272
doc4  -0.055297 -1.972770 -0.238122  1.535897 -0.145103 -1.324335   -1.548705
doc5   2.932846 -8.254079  0.602778  1.092057  0.339039  1.709239    1.582829
doc6  -4.842404  0.777998  0.189492  0.612070 -0.667979 -0.228740   -0.265189
doc7  -4.408933  0.844391  0.385219  0.445992 -0.455010 -3.901626    1.383005
doc8   5.581299 -0.893170 -0.570378 -0.819958  3.479104 -0.646614    2.426800
doc9   2.969656  0.239313 -0.355300 -1.055428 -1.375648  3.571389    0.651829
doc10 -1.042708  0.369075 -0.036895  1.123171 -1.085541  0.598813   -3.843906
doc11  1.667831 -0.338080  0.414887 -0.477902 -1.033873  0.249278   -1.826523
doc12 -0.414850  2.038289 -0.958255 -0.550615 -1.111630 -1.264392   -1.474125
doc13 -0.781777  0.179731  2.152819  0.345347 -1.748731  2.223993    2.010644
doc14  0.349412 -3.107109 -0.310794  3.594318 -1.911479 -3.575751   -1.359736
doc15 -1.753822 -3.646434 -1.604757 -2.133182  0.486730  1.235697   -1.916960
doc16 -2.320944 -2.998520  2.865884  1.094325  0.320703  1.694793    1.580740
doc17  1.625766 -2.795290  1.200840 -0.376263 -0.273793 -4.234810    0.051919
doc18  1.255947  2.167574 -1.851072  2.735034  1.683938 -0.851482    0.504757
doc19 -1.692608  2.359400 -0.033149  0.947973  0.936098  1.855279    1.562762
doc20  1.201153  4.381110 -0.837414 -1.508854  2.272082  1.557292    0.771143
doc21  1.968610  1.162302 -0.933318 -0.805060  1.936021  0.314902   -0.819113
doc22  3.223677  6.248641 -1.048334 -3.946151 -0.277384  3.716345   -0.352808
doc23  0.637845  3.270625 -1.401567 -2.209625  2.452448  2.277193    0.389751
doc24 -3.599661 -6.056448  2.149955  1.926221 -2.446036 -2.289146   -0.115813
doc25 -4.300740 -7.408212  2.512594  1.903952 -3.085770 -2.767608   -0.371320
doc26 -0.457787 -7.680381  2.328862  0.134329 -3.454051 -2.973423   -1.067460

          sport
doc1   0.758256
doc2   1.763957
doc3  -2.277354
doc4   0.453435
doc5  -0.975983
doc6   1.428336
doc7   1.429765
doc8  -1.005491
doc9  -0.727917
doc10 -0.871891
doc11 -0.542325
doc12 -0.081443
doc13  0.459413
doc14 -0.103266
doc15  0.918144
doc16  0.389234
doc17 -0.822499
doc18 -1.264749
doc19  0.322639
doc20  0.112154
doc21 -1.826596
doc22  1.856288
doc23  0.341431
doc24  3.093995
doc25 -0.246424
doc26 -0.530854

[26 rows x 22 columns]
```

5e) Compute the MSE between the original matrix and the predicted matrix using the truncated number of Eigen vectors.

In [18]:

```
df_list=df.values.tolist()
A_approx_list=pd.DataFrame(A_approx, index=df.index, columns=df.columns).values.tolist()
print("MSE between the original matrix and predcited matrix using the truncated number of eigen ve
ctors: "+str(mean_squared_error(A_approx_list, df_list)))
```

```
MSE between the original matrix and predcited matrix using the truncated number of eigen vectors:
4.191212518402567
```

5d) How many concept vectors are chosen by you in step-b above? Write the intuitive interpretation of each of these vectors. What is the intuitive interpretation of the Eigen-values associated with each of these vectors?

Note:

```
* Here I am taking plots for all columns of U and rows of Vh and explaining about few eigen
vectors(considered from first 4) with concept strength in the following observations
* Also from abov mean value we can say that as mse is very less, first four eigen values
and corresponding eigen vector are much enough to extract useful information. It indicates
that with these 4 hidden features/concepts we can summarize the docs and words.
```

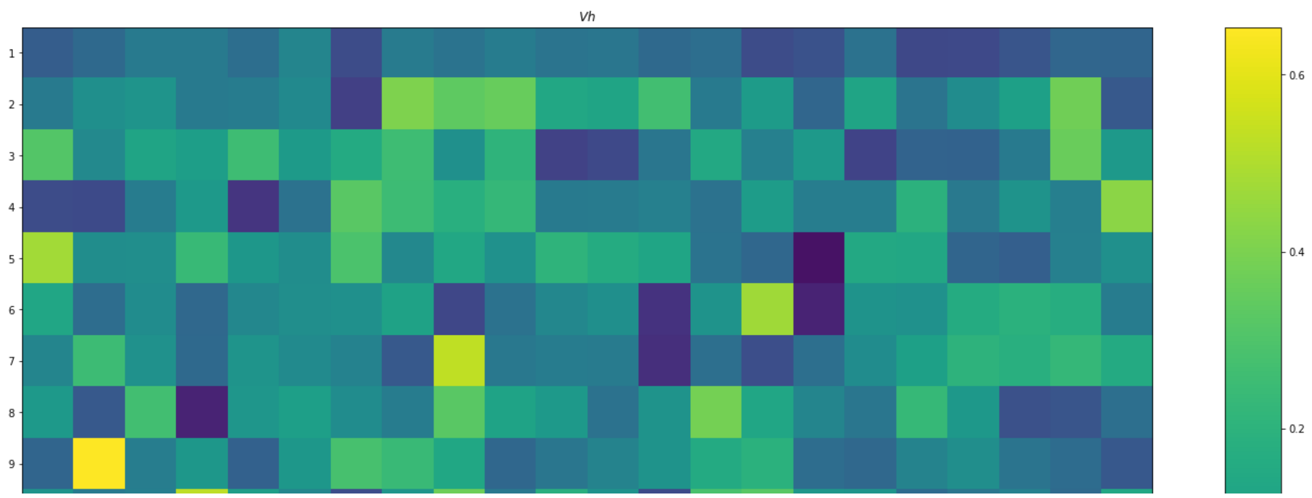
To help visualize the similarity between words, Vh can be displayed as an image. For example, notice how the similar words (gene and disease, election and vote, sport and game) have similar color values in the first 3 and 4th rows:

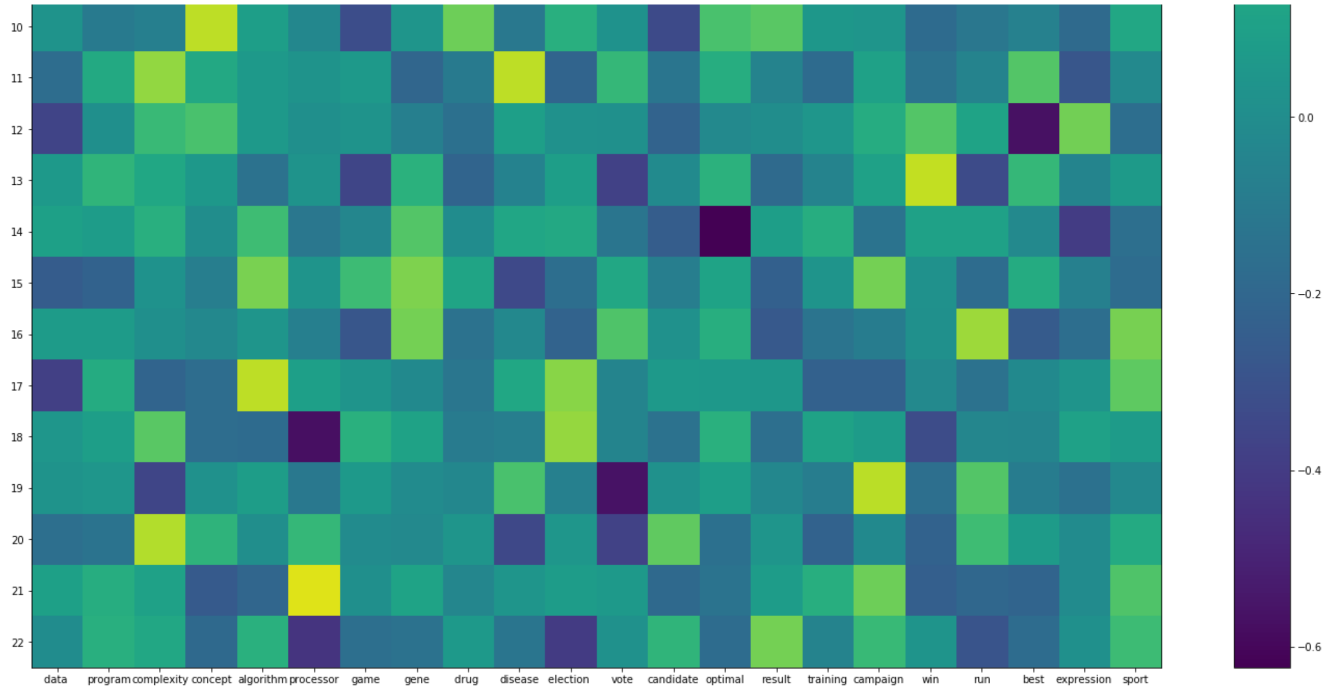
Observations:

```
* I am just considering (for now) 3 and 4th rows of Vh which are eigen vectors and observed
that pair of words like (gene,disease),(election,vote)(sport,game) etc are closely related.
So here SVD is trying to get concept vector which learn hidden features/ latent features th
at explains the concpets that are closely related. Similarly 1 and 2 rows have words which
are closely related.
```

In [19]:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(25,20))
plt.imshow(Vh, interpolation='none')
plt.xticks(range(len(df.columns)))
plt.yticks(range(len(Vh)))
#plt.ylim([len(Vh) - 1.5, -.5])
ax = plt.gca()
ax.set_xticklabels(df.columns)
ax.set_yticklabels(range(1, len(Vh) + 1))
plt.title("$Vh$")
plt.colorbar();
```





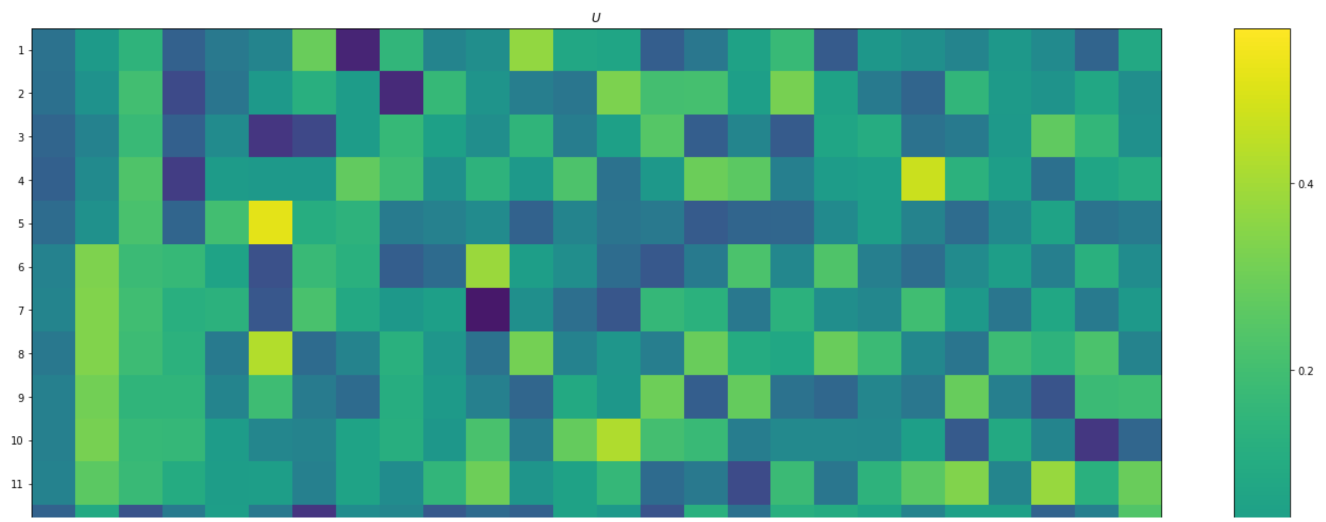
To help visualize the similarity between words, U can be displayed as an image.

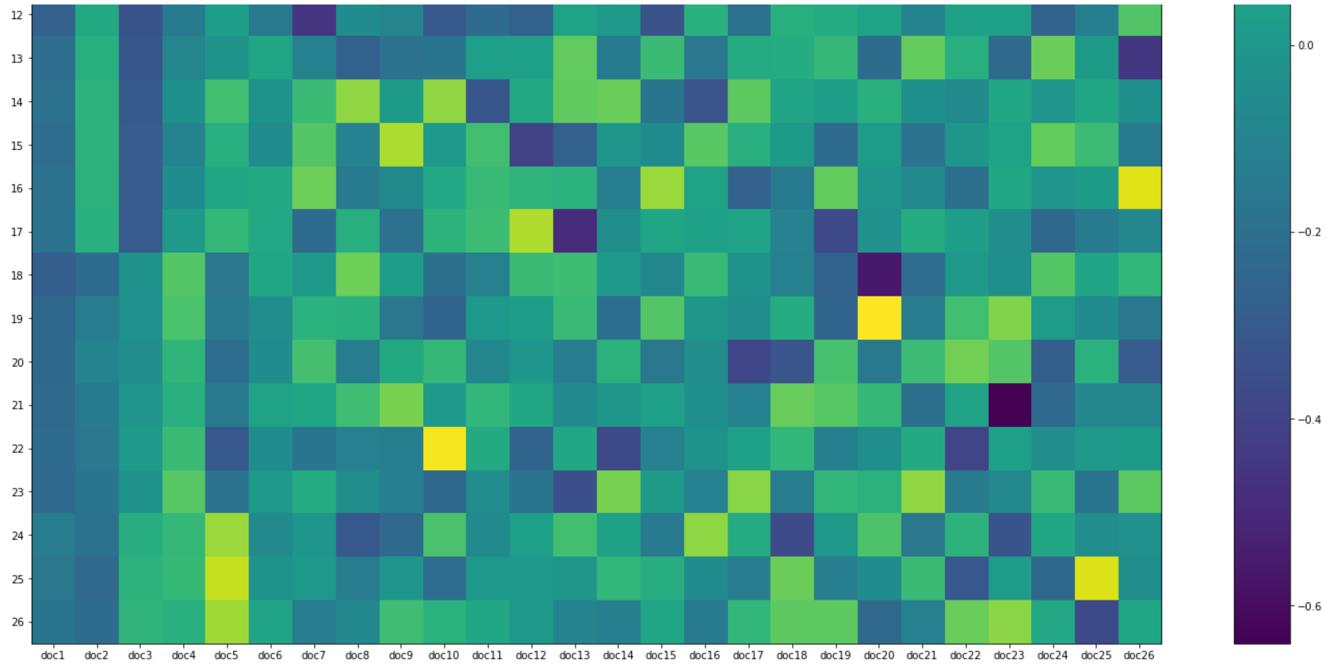
Observations:

```
*Recall from:
        ai=u1*σ1*(V1,i)+u2*σ2*(V2,i)+u3*σ3*(V3,i)...
    Thus the words differ very much in how much the values in  u[k]  contribute to their
final word count.
*From first 4 columns we can see patterns such that no fluctuating changes that resembles s
ome kind of hidden features for which first 4 docs can be summarised separetly using the co
ncepts/hidden features.
```

In [21]:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(25,20))
plt.imshow(U, interpolation='none')
plt.xticks(range(len(df.index)))
plt.yticks(range(len(U)))
#plt.ylim([len(Vh) - 1.5, -.5])
ax = plt.gca()
ax.set_xticklabels(df.index)
ax.set_yticklabels(range(1, len(U) + 1))
plt.title("$U$")
plt.colorbar();
```





5c) Take one of the dropped Eigen-vectors and interpret it in terms of documents and words to show its relevance/irrelevance to the original data table.

In [22]:

```
dropped_s=s[-1]
dropped_U=U[:,25:26]
#dropped_U.resize(26,1)
dropped_Vh=Vh[21:22,:]
#dropped_Vh.resize(1,22)

A_approx_dropped = np.matrix(dropped_U) * np.matrix(dropped_s) * np.matrix(dropped_Vh)
print("A calculated using only the first four components:\n")
print(pd.DataFrame(A_approx_dropped, index=df.index, columns=df.columns))
print("\nError from actual value:\n")
print(df - A_approx_dropped)
```

```
A calculated using only the first four components:

           data   program  complexity   concept  algorithm  processor  \
doc1  -0.000271  0.019103    0.014709 -0.019401   0.019777  -0.045252
doc2   0.000136 -0.009623   -0.007410  0.009773  -0.009962   0.022795
doc3   0.000114 -0.008006   -0.006165  0.008131  -0.008288   0.018965
doc4  -0.000334  0.023556    0.018138 -0.023923   0.024386  -0.055799
doc5   0.000455 -0.032068   -0.024692  0.032568  -0.033198   0.075963
doc6   0.000174 -0.012299   -0.009470  0.012491  -0.012733   0.029134
doc7  -0.000041  0.002910    0.002241 -0.002956   0.003013  -0.006894
doc8   0.000315 -0.022196   -0.017091  0.022542  -0.022978   0.052579
doc9  -0.000592  0.041728    0.032130 -0.042378   0.043198  -0.098845
doc10  0.000757 -0.053353   -0.041081  0.054185  -0.055233   0.126382
doc11 -0.000900  0.063471    0.048872 -0.064461   0.065708  -0.150351
doc12 -0.000746  0.052595    0.040497 -0.053414   0.054448  -0.124586
doc13  0.001384 -0.097605   -0.075155  0.099126  -0.101044   0.231206
doc14  0.000106 -0.007487   -0.005765  0.007603  -0.007750   0.017734
doc15  0.000437 -0.030825   -0.023735  0.031305  -0.031911   0.073017
doc16 -0.001605  0.113147    0.087122 -0.114911   0.117134  -0.268023
doc17  0.000257 -0.018094   -0.013932  0.018376  -0.018731   0.042861
doc18 -0.000489  0.034487    0.026554 -0.035024   0.035702  -0.081692
doc19  0.000498 -0.035142   -0.027059  0.035690  -0.036380   0.083244
doc20  0.000918 -0.064708   -0.049824  0.065717  -0.066988   0.153280
doc21  0.000253 -0.017836   -0.013734  0.018115  -0.018465   0.042251
doc22 -0.000048  0.003362    0.002589 -0.003415   0.003481  -0.007964
doc23 -0.000822  0.057975    0.044640 -0.058879   0.060018  -0.137332
doc24  0.000096 -0.006746   -0.005195  0.006851  -0.006984   0.015981
doc25  0.000141 -0.009932   -0.007648  0.010087  -0.010282   0.023528
doc26 -0.000230  0.016207    0.012479 -0.016460   0.016778  -0.038391

           game      gene      drug   disease  ...  candidate   optimal  \
doc1  -0.016291 -0.015039  0.006842 -0.013168  ...   0.022083 -0.018062
doc2   0.008206  0.007576 -0.003447  0.006633  ...  -0.011124  0.009098
```

```
doc3   0.006827  0.006303 -0.002867  0.005518  ...  -0.009255  0.007570
doc4  -0.020087 -0.018544  0.008437 -0.016237  ...   0.027230 -0.022271
doc5   0.027346  0.025245 -0.011485  0.022104  ...  -0.037070  0.030320
doc6   0.010488  0.009682 -0.004405  0.008478  ...  -0.014218  0.011629
doc7  -0.002482 -0.002291  0.001042 -0.002006  ...   0.003364 -0.002752
doc8   0.018928  0.017474 -0.007950  0.015300  ...  -0.025659  0.020986
doc9  -0.035584 -0.032850  0.014945 -0.028762  ...   0.048237 -0.039453
doc10  0.045497  0.042001 -0.019109  0.036775  ...  -0.061676  0.050444
doc11 -0.054126 -0.049967  0.022733 -0.043750  ...   0.073373 -0.060011
doc12 -0.044850 -0.041405  0.018837 -0.036253  ...   0.060799 -0.049727
doc13  0.083233  0.076838 -0.034958  0.067278  ...  -0.112830  0.092283
doc14  0.006384  0.005894 -0.002681  0.005160  ...  -0.008654  0.007078
doc15  0.026286  0.024266 -0.011040  0.021247  ...  -0.035633  0.029144
doc16 -0.096487 -0.089074  0.040525 -0.077991  ...   0.130797 -0.106978
doc17  0.015430  0.014244 -0.006481  0.012472  ...  -0.020916  0.017107
doc18 -0.029409 -0.027149  0.012352 -0.023771  ...   0.039866 -0.032606
doc19  0.029968  0.027665 -0.012586  0.024223  ...  -0.040624  0.033226
doc20  0.055180  0.050941 -0.023176  0.044602  ...  -0.074802  0.061180
doc21  0.015210  0.014042 -0.006388  0.012294  ...  -0.020619  0.016864
doc22 -0.002867 -0.002647  0.001204 -0.002318  ...   0.003887 -0.003179
doc23 -0.049439 -0.045641  0.020764 -0.039962  ...   0.067019 -0.054814
doc24  0.005753  0.005311 -0.002416  0.004650  ...  -0.007799  0.006378
doc25  0.008470  0.007819 -0.003557  0.006846  ...  -0.011482  0.009391
doc26 -0.013821 -0.012759  0.005805 -0.011171  ...   0.018735 -0.015323

         result  training  campaign       win       run      best  expression  \
doc1   0.040309 -0.005504  0.024724  0.004125 -0.030759 -0.017897    0.001754
doc2  -0.020305  0.002773 -0.012454 -0.002078  0.015494  0.009015   -0.000884
doc3  -0.016893  0.002307 -0.010362 -0.001729  0.012891  0.007500   -0.000735
doc4   0.049704 -0.006787  0.030486  0.005087 -0.037928 -0.022068    0.002163
doc5  -0.067664  0.009240 -0.041503 -0.006925  0.051633  0.030043   -0.002945
doc6  -0.025952  0.003544 -0.015918 -0.002656  0.019803  0.011522   -0.001129
doc7   0.006141 -0.000839  0.003766  0.000628 -0.004686 -0.002726    0.000267
doc8  -0.046835  0.006395 -0.028727 -0.004793  0.035739  0.020795   -0.002038
doc9   0.088047 -0.012023  0.054005  0.009011 -0.067187 -0.039093    0.003832
doc10 -0.112576  0.015373 -0.069051 -0.011521  0.085905  0.049984   -0.004899
doc11  0.133927 -0.018288  0.082146  0.013706 -0.102197 -0.059463    0.005828
doc12  0.110976 -0.015154  0.068069  0.011357 -0.084684 -0.049273    0.004830
doc13 -0.205950  0.028123 -0.126322 -0.021077  0.157156  0.091441   -0.008963
doc14 -0.015797  0.002157 -0.009689 -0.001617  0.012054  0.007014   -0.000687
doc15 -0.065041  0.008881 -0.039894 -0.006656  0.049631  0.028878   -0.002830
doc16  0.238744 -0.032601  0.146437  0.024433 -0.182181 -0.106002    0.010390
doc17 -0.038179  0.005213 -0.023418 -0.003907  0.029134  0.016951   -0.001661
doc18  0.072768 -0.009937  0.044633  0.007447 -0.055528 -0.032309    0.003167
doc19 -0.074151  0.010125 -0.045481 -0.007588  0.056583  0.032923   -0.003227
doc20 -0.136536  0.018644 -0.083747 -0.013973  0.104188  0.060622   -0.005942
doc21 -0.037636  0.005139 -0.023084 -0.003852  0.028719  0.016710   -0.001638
doc22  0.007094 -0.000969  0.004351  0.000726 -0.005414 -0.003150    0.000309
doc23  0.122330 -0.016704  0.075033  0.012519 -0.093348 -0.054314    0.005324
doc24 -0.014235  0.001944 -0.008731 -0.001457  0.010862  0.006320   -0.000619
doc25 -0.020958  0.002862 -0.012855 -0.002145  0.015992  0.009305   -0.000912
doc26  0.034197 -0.004670  0.020975  0.003500 -0.026095 -0.015184    0.001488

          sport
doc1   0.026481
doc2  -0.013339
doc3  -0.011098
doc4   0.032652
doc5  -0.044452
doc6  -0.017049
doc7   0.004034
doc8  -0.030768
doc9   0.057842
doc10 -0.073956
doc11  0.087982
doc12  0.072905
doc13 -0.135297
doc14 -0.010378
doc15 -0.042728
doc16  0.156841
doc17 -0.025081
doc18  0.047804
doc19 -0.048713
doc20 -0.089696
doc21 -0.024724
doc22  0.004661
doc23  0.080364
```

```
doc24 -0.009351
doc25 -0.013768
doc26  0.022466

[26 rows x 22 columns]

Error from actual value:

            data    program  complexity   concept  algorithm  processor  \
doc1    6.000271  11.980897    2.985291  4.019401   7.980223   2.045252
doc2   11.999864   5.009623    5.007410 -0.009773  15.009962   1.977205
doc3    9.999886   8.008006    4.006165  5.991869  12.008288   3.981035
doc4   16.000334  11.976444    6.981862  1.023923  14.975614   6.055799
doc5   17.999545   4.032068    4.024692 -0.032568  10.033198   3.924037
doc6    0.999826   0.012299    4.009470 -0.012491   0.012733  -0.029134
doc7    4.000041  -0.002910   -0.002241  0.002956  -0.003013   0.006894
doc8    0.999685   0.022196    2.017091 -0.022542   0.022978  -0.052579
doc9    0.000592  -0.041728   -0.032130  0.042378  -0.043198   0.098845
doc10   1.999243   1.053353    4.041081  0.945815   0.055233  -0.126382
doc11   4.000900  -0.063471    4.951128  2.064461  -0.065708   0.150351
doc12   5.000746   5.947405   -0.040497  0.053414  -0.054448   0.124586
doc13  -0.001384   5.097605    0.075155 -0.099126   0.101044  -0.231206
doc14   2.999894   4.007487    1.005765 -0.007603   0.007750  -0.017734
doc15   1.999563  10.030825    0.023735 -0.031305   0.031911  -0.073017
doc16   0.001605   4.886853    0.912878  1.114911  -0.117134   0.268023
doc17  -0.000257   0.018094    2.013932  1.981624   0.018731  -0.042861
doc18   2.000489   0.965513    1.973446  0.035024  -0.035702   0.081692
doc19  -0.000498   0.035142    3.027059 -0.035690   0.036380  -0.083244
doc20   0.999082   4.064708    0.049824  2.934283   0.066988  -0.153280
doc21   0.999747   4.017836    3.013734  0.981885   1.018465  -0.042251
doc22   0.000048  -0.003362   -0.002589  5.003415  -0.003481   0.007964
doc23   0.000822  -0.057975   -0.044640  0.058879   0.939982   0.137332
doc24   9.999904   0.006746    0.005195  9.993149   3.006984  -0.015981
doc25  11.999859   1.009932    2.007648  6.989913   3.010282  -0.023528
doc26  11.000230   1.983793    0.987521  8.016460   3.983222   0.038391

            game       gene       drug    disease  ...  candidate    optimal  \
doc1    2.016291   0.015039  -0.006842   0.013168  ...   0.977917   2.018062
doc2   -0.008206  -0.007576   0.003447  -0.006633  ...   0.011124   5.990902
doc3    8.993173  -0.006303   0.002867  -0.005518  ...   8.009255   7.992430
doc4    6.020087   0.018544  -0.008437   0.016237  ...   3.972770  12.022271
doc5    7.972654  -0.025245   0.011485  -0.022104  ...   0.037070   5.969680
doc6   -0.010488   7.990318  12.004405  15.991522  ...   8.014218  -0.011629
doc7    0.002482  12.002291  14.998958   8.002006  ...   7.996636   0.002752
doc8   -0.018928  15.982526   4.007950   9.984700  ...   4.025659   1.979014
doc9    0.035584  14.032850   5.985055   8.028762  ...   4.951763   0.039453
doc10  -0.045497  14.957999   9.019109  11.963225  ...   6.061676  -0.050444
doc11   0.054126  10.049967   6.977267  11.043750  ...   3.926627   1.060011
doc12   2.044850   0.041405  -0.018837   0.036253  ...  14.939201   1.049727
doc13  -0.083233  -0.076838   0.034958  -0.067278  ...   8.112830  -0.092283
doc14  -0.006384  -0.005894  10.002681  -0.005160  ...   6.008654  -0.007078
doc15   1.973714  -0.024266   8.011040  -0.021247  ...   8.035633  -0.029144
doc16   0.096487   0.089074   5.959475   0.077991  ...   3.869203   0.106978
doc17   2.984570  -0.014244   4.006481   2.987528  ...   7.020916  -0.017107
doc18  15.029409   0.027149  -0.012352   0.023771  ...  -0.039866   6.032606
doc19  11.970032  -0.027665   2.012586  -0.024223  ...   2.040624   3.966774
doc20   8.944820  -0.050941   4.023176  -0.044602  ...   0.074802   1.938820
doc21  11.984790  -0.014042   2.006388  -0.012294  ...   0.020619   5.983136
doc22   8.002867   0.002647  -0.001204   0.002318  ...  -0.003887   8.003179
doc23  14.049439   0.045641   0.979236   1.039962  ...  -0.067019   0.054814
doc24  11.994247  -0.005311   0.002416  -0.004650  ...   0.007799  -0.006378
doc25  18.991530  -0.007819   0.003557  -0.006846  ...   0.011482  -0.009391
doc26  18.013821   0.012759  -0.005805   0.011171  ...  -0.018735   4.015323

          result   training   campaign        win        run      best  \
doc1    2.959691  10.005504  -0.024724   1.995875   7.030759  6.017897
doc2    4.020305  11.997227   0.012454   0.002078   7.984506  5.990985
doc3    5.016893  15.997693   0.010362   4.001729   4.987109  1.992500
doc4    5.950296   9.006787  -0.030486   4.994913   8.037928  4.022068
doc5    8.067664  -0.009240   0.041503   4.006925   5.948367  5.969957
doc6    0.025952  -0.003544   0.015918   1.002656  -0.019803  3.988478
doc7   -0.006141   0.000839  -0.003766  -0.000628   0.004686  0.002726
doc8   12.046835  -0.006395   0.028727   1.004793   5.964261  4.979205
doc9    7.911953   0.012023  -0.054005  -0.009011   0.067187  8.039093
doc10   4.112576  -0.015373   0.069051   2.011521  -0.085905  4.950016
doc11   5.866073   0.018288  -0.082146  -0.013706   0.102197  4.059463
doc12   8.889024   8.015154   9.931931   9.988643  12.084684  8.049273
```

```
doc13   7.205950   3.971877  12.126322   9.021077   8.842844  9.908559
doc14   8.015797  -0.002157   9.009689  12.001617   7.987946  3.992986
doc15   6.065041  -0.008881   8.039894   6.006656  10.950369  8.971122
doc16   4.761256   0.032601  11.853563   8.975567  10.182181  9.106002
doc17   9.038179  -0.005213  10.023418   8.003907   8.970866  2.983049
doc18   9.927232  12.009937  -0.044633  15.992553  10.055528  6.032309
doc19   6.074151   9.989875   2.045481  12.007588   7.943417  7.967077
doc20   9.136536  11.981356   2.083747   9.013973   9.895812  7.939378
doc21   9.037636   8.994861   1.023084   9.003852   8.971281  5.983290
doc22   9.992906  14.000969  -0.004351   5.999274   6.005414  9.003150
doc23   7.877670  11.016704  -0.075033   8.987481   9.093348  8.054314
doc24   0.014235  -0.001944   0.008731   8.001457  -0.010862 -0.006320
doc25   0.020958  -0.002862   0.012855   9.002145  -0.015992 -0.009305
doc26   3.965803   0.004670  -0.020975   6.996500   0.026095  0.015184

       expression     sport
doc1    11.998246  -0.026481
doc2     8.000884   0.013339
doc3     4.000735   0.011098
doc4     7.997837  -0.032652
doc5    10.002945   0.044452
doc6    12.001129   2.017049
doc7    13.999733   0.995966
doc8    16.002038   0.030768
doc9    11.996168  -0.057842
doc10    8.004899   0.073956
doc11    8.994172  -0.087982
doc12   -0.004830  -0.072905
doc13    3.008963   0.135297
doc14    0.000687   0.010378
doc15    0.002830   0.042728
doc16    2.989610  -0.156841
doc17    1.001661   0.025081
doc18   -0.003167  11.952196
doc19    2.003227  11.048713
doc20    2.005942   9.089696
doc21    0.001638   7.024724
doc22   -0.000309  11.995339
doc23   -0.005324  11.919636
doc24    0.000619  12.009351
doc25    0.000912  10.013768
doc26   -0.001488   8.977534

[26 rows x 22 columns]
```

In [23]:

```
df_list=df.values.tolist()
A_approx_list_drop=pd.DataFrame(A_approx_dropped, index=df.index, columns=df.columns).values.tolist
()
mean_squared_error(A_approx_list_drop, df_list)
```

Out[23]:

```
34.25604360428098
```

Observation:

```
* Here I am considering last eigen value and corresponding eigen vector in Vh and
multiplying to get new predicted matrix using this last concept/eigen value and observed th
at mse has been increased drastically indicating that does not useful/ contributing to usef
ul information.
* Also if we consider eigen vector or last row in Vh and see its effect on doc1 to doc 17,
which can bee seen from error dataframe printed above for the last word sport. (Almost gett
ing zero error for doc1 to doc17)
```

5f) Four interpretations of SVD

Interpretation 1 :-

```
if A is the document-to-term matrix,
Q: what is AT A?
A: term-to-term ([m x m]) similarity matrix
Q: A AT ?
A: document-to-document ([n x n]) similarity matrix
```

Term-to-term Matrix

In [24]:

```python
print( np.round(Vh, decimals=2))
```

```
[[-0.25 -0.19 -0.1  -0.1  -0.16 -0.04 -0.33 -0.09 -0.14 -0.09 -0.13 -0.12
  -0.18 -0.16 -0.33 -0.3  -0.14 -0.35 -0.34 -0.29 -0.2  -0.21]
 [-0.09  0.01  0.04 -0.1  -0.09 -0.02 -0.38  0.41  0.34  0.36  0.14  0.12
   0.27 -0.1   0.08 -0.2   0.12 -0.13  0.    0.1   0.38 -0.26]
 [ 0.31 -0.02  0.12  0.09  0.26  0.07  0.16  0.26  0.02  0.21 -0.37 -0.34
  -0.12  0.15 -0.07  0.07 -0.37 -0.21 -0.22 -0.1   0.36  0.07]
 [-0.33 -0.34 -0.09  0.06 -0.43 -0.14  0.33  0.25  0.18  0.23 -0.1  -0.09
  -0.07 -0.14  0.08 -0.08 -0.08  0.2  -0.11  0.03 -0.07  0.43]
 [ 0.48  0.    0.01  0.24  0.05  0.    0.29 -0.03  0.14  0.02  0.21  0.16
   0.13 -0.14 -0.2  -0.56  0.15  0.14 -0.21 -0.24 -0.07  0.02]
 [ 0.13 -0.17 -0.   -0.19 -0.03  0.01  0.02  0.11 -0.35 -0.14 -0.03  0.01
  -0.44  0.03  0.47 -0.49  0.03  0.02  0.16  0.19  0.17 -0.09]
 [-0.04  0.25  0.02 -0.19  0.04 -0.01 -0.06 -0.27  0.53 -0.11 -0.09 -0.09
  -0.45 -0.16 -0.32 -0.15 -0.    0.1   0.2   0.18  0.23  0.16]
 [ 0.06 -0.27  0.27 -0.49  0.05  0.1  -0.   -0.09  0.33  0.11  0.07 -0.14
   0.03  0.39  0.13 -0.04 -0.12  0.23  0.06 -0.31 -0.29 -0.16]
 [-0.21  0.65 -0.08  0.05 -0.23  0.05  0.28  0.24  0.14 -0.2  -0.12 -0.05
   0.03  0.16  0.19 -0.17 -0.19 -0.05  0.01 -0.13 -0.17 -0.27]
 [ 0.03 -0.1  -0.07  0.53  0.09 -0.03 -0.32  0.04  0.37 -0.11  0.18  0.03
  -0.33  0.29  0.33  0.05  0.05 -0.17 -0.12 -0.06 -0.18  0.14]
 [-0.17  0.15  0.44  0.15  0.06  0.03  0.07 -0.2  -0.1   0.53 -0.21  0.23
  -0.13  0.17 -0.05 -0.18  0.11 -0.14 -0.05  0.31 -0.28 -0.02]
 [-0.36  0.01  0.24  0.28  0.06  0.01  0.03 -0.07 -0.15  0.1   0.02  0.02
  -0.22 -0.02  0.    0.05  0.17  0.31  0.12 -0.57  0.38 -0.16]
 [ 0.06  0.21  0.14  0.06 -0.14  0.03 -0.36  0.19 -0.21 -0.06  0.09 -0.38
  -0.01  0.2  -0.18 -0.05  0.11  0.54 -0.33  0.23 -0.05  0.07]
 [ 0.1   0.08  0.18  0.01  0.26 -0.12 -0.04  0.31 -0.    0.13  0.15 -0.13
  -0.25 -0.62  0.09  0.17 -0.14  0.11  0.11 -0.02 -0.4  -0.16]
 [-0.25 -0.22  0.03 -0.08  0.4   0.04  0.25  0.41  0.12 -0.34 -0.16  0.14
  -0.08  0.12 -0.23  0.04  0.39  0.02 -0.17  0.16 -0.07 -0.17]
 [ 0.08  0.08  0.01 -0.03  0.04 -0.07 -0.28  0.39 -0.14 -0.03 -0.22  0.3
   0.02  0.18 -0.27 -0.13 -0.09  0.02  0.47 -0.25 -0.16  0.39]
 [-0.38  0.16 -0.21 -0.17  0.53  0.1   0.04 -0.02 -0.12  0.14  0.43 -0.05
   0.07  0.06  0.05 -0.23 -0.23 -0.02 -0.14 -0.02  0.04  0.34]
 [ 0.05  0.09  0.33 -0.17 -0.18 -0.57  0.19  0.11 -0.09 -0.08  0.45 -0.05
  -0.14  0.19 -0.16  0.11  0.08 -0.33 -0.04 -0.04  0.11  0.08]
 [ 0.03  0.05 -0.36  0.02  0.09 -0.11  0.06 -0.01 -0.03  0.29 -0.07 -0.56
   0.02  0.09 -0.03 -0.08  0.52 -0.16  0.31 -0.09 -0.15 -0.03]
 [-0.16 -0.13  0.51  0.21  0.01  0.23 -0.01 -0.02  0.04 -0.35  0.05 -0.37
   0.34 -0.15  0.04 -0.22 -0.02 -0.22  0.26  0.08 -0.01  0.16]
 [ 0.1   0.17  0.11 -0.26 -0.2   0.59  0.01  0.12 -0.04  0.04  0.08  0.07
  -0.19 -0.14  0.08  0.18  0.37 -0.24 -0.2  -0.21 -0.01  0.3 ]
 [-0.    0.18  0.14 -0.19  0.19 -0.43 -0.16 -0.14  0.07 -0.13 -0.4   0.02
   0.21 -0.17  0.39 -0.05  0.24  0.04 -0.29 -0.17  0.02  0.25]]
```

Document-to-document

In [25]:

```python
print( np.round(U, decimals=2))
```

```
[[-0.19  0.02  0.14 -0.26 -0.15 -0.09  0.29 -0.51  0.16 -0.1  -0.04  0.37
   0.08  0.07 -0.28 -0.16  0.05  0.17 -0.29 -0.   -0.04 -0.1   0.   -0.06
  -0.25  0.09]
 [-0.19 -0.02  0.2  -0.37 -0.17  0.01  0.12  0.02 -0.5   0.17 -0.02 -0.12
  -0.17  0.33  0.21  0.21  0.04  0.32  0.06 -0.15 -0.25  0.15  0.02 -0.02
   0.08 -0.04]
```

```
 [-0.24 -0.11  0.18 -0.27 -0.06 -0.45 -0.38  0.03  0.17  0.04 -0.04  0.15
  -0.13  0.05  0.25 -0.28 -0.09 -0.29  0.07  0.11 -0.19 -0.14  0.01  0.27
   0.16 -0.04]
 [-0.27 -0.06  0.24 -0.42  0.02  0.01  0.01  0.28  0.19 -0.03  0.14  0.01
   0.23 -0.19  0.    0.3   0.26 -0.12  0.03  0.04  0.47  0.13  0.04 -0.2
   0.07  0.11]
 [-0.21 -0.03  0.22 -0.25  0.2   0.52  0.11  0.14 -0.14 -0.11 -0.06 -0.26
  -0.1  -0.18 -0.15 -0.3  -0.25 -0.24 -0.06  0.04 -0.1  -0.21 -0.07  0.06
  -0.18 -0.15]
 [-0.11  0.33  0.18  0.17  0.06 -0.34  0.18  0.13 -0.28 -0.22  0.39  0.04
  -0.04 -0.22 -0.31 -0.15  0.23 -0.08  0.23 -0.12 -0.21 -0.06  0.04 -0.12
   0.13 -0.06]
 [-0.1   0.34  0.2   0.12  0.13 -0.31  0.22  0.09  0.    0.04 -0.56 -0.04
  -0.2  -0.32  0.17  0.13 -0.16  0.14 -0.04 -0.08  0.19  0.01 -0.17  0.08
  -0.15  0.01]
 [-0.16  0.34  0.19  0.13 -0.15  0.43 -0.22 -0.1   0.13 -0.   -0.19  0.31
  -0.1  -0.01 -0.12  0.29  0.1   0.08  0.29  0.18 -0.08 -0.17  0.19  0.14
   0.23 -0.1 ]
 [-0.11  0.31  0.15  0.15 -0.1   0.19 -0.14 -0.22  0.11  0.02 -0.12 -0.24
   0.09 -0.    0.3  -0.28  0.28 -0.18 -0.24 -0.09 -0.16  0.28 -0.12 -0.33
   0.18  0.19]
 [-0.11  0.32  0.17  0.16  0.03 -0.09 -0.1   0.06  0.12 -0.    0.22 -0.13
   0.28  0.42  0.21  0.18 -0.13 -0.07 -0.07 -0.08  0.04 -0.3   0.09 -0.1
  -0.45 -0.24]
 [-0.11  0.26  0.18  0.1   0.03  0.04 -0.11  0.06 -0.05  0.15  0.3  -0.01
   0.05  0.16 -0.22 -0.15 -0.36  0.18 -0.17  0.14  0.26  0.34 -0.09  0.38
   0.13  0.29]
 [-0.26  0.09 -0.33 -0.14  0.04 -0.15 -0.45 -0.06 -0.09 -0.3  -0.22 -0.26
   0.05  0.01 -0.33  0.12 -0.19  0.12  0.1   0.06 -0.1   0.04  0.05 -0.26
  -0.12  0.24]
 [-0.2   0.12 -0.32 -0.09 -0.01  0.07 -0.11 -0.27 -0.19 -0.18  0.04  0.04
   0.28 -0.14  0.18 -0.16  0.1   0.11  0.16 -0.22  0.28  0.11 -0.23  0.3
   0.02 -0.44]
 [-0.19  0.14 -0.3  -0.03  0.2  -0.02  0.18  0.37  0.02  0.36 -0.32  0.09
   0.27  0.3  -0.17 -0.32  0.26  0.07  0.03  0.13 -0.04 -0.07  0.08 -0.01
   0.07 -0.03]
 [-0.2   0.14 -0.29 -0.1   0.12 -0.06  0.25 -0.11  0.42  0.01  0.2  -0.41
  -0.27 -0.01 -0.06  0.26  0.12  0.01 -0.22  0.03 -0.18 -0.    0.06  0.28
   0.19 -0.14]
 [-0.18  0.14 -0.29 -0.05  0.08  0.1   0.3  -0.14 -0.07  0.09  0.18  0.14
   0.13 -0.12  0.38  0.06 -0.27 -0.15  0.28 -0.01 -0.07 -0.19  0.08 -0.01
   0.02  0.51]
 [-0.18  0.13 -0.29  0.01  0.16  0.09 -0.22  0.12 -0.18  0.14  0.19  0.42
  -0.49 -0.04  0.08  0.05  0.05 -0.1  -0.37 -0.03  0.11  0.03 -0.05 -0.23
  -0.15 -0.08]
 [-0.28 -0.22 -0.02  0.25 -0.16  0.08  0.01  0.3   0.03 -0.2  -0.11  0.18
   0.19  0.01 -0.08  0.18 -0.03 -0.11 -0.26 -0.56 -0.21  0.01 -0.04  0.25
   0.06  0.16]
 [-0.23 -0.14 -0.03  0.22 -0.15 -0.04  0.13  0.12 -0.16 -0.25  0.    0.03
   0.18 -0.2   0.24 -0.   -0.05  0.11 -0.25  0.57 -0.13  0.2   0.34  0.02
  -0.06 -0.16]
 [-0.24 -0.1  -0.04  0.15 -0.21 -0.05  0.21 -0.14  0.09  0.17 -0.08 -0.01
  -0.13  0.13 -0.16 -0.04 -0.38 -0.32  0.21 -0.15  0.19  0.31  0.24 -0.28
   0.13 -0.29]
 [-0.23 -0.14 -0.01  0.12 -0.15  0.05  0.07  0.2   0.32  0.01  0.16  0.08
  -0.06 -0.    0.04 -0.04 -0.11  0.29  0.25  0.16 -0.2   0.05 -0.64 -0.23
  -0.08 -0.08]
 [-0.22 -0.16  0.01  0.18 -0.31 -0.05 -0.17 -0.11 -0.13  0.55  0.1  -0.25
   0.08 -0.37 -0.11 -0.02  0.05  0.16 -0.12 -0.04  0.1  -0.39  0.04 -0.05
   0.01  0.02]
 [-0.23 -0.17 -0.02  0.25 -0.18  0.01  0.1  -0.05 -0.12 -0.23 -0.05 -0.18
  -0.35  0.32  0.02 -0.1   0.35 -0.13  0.16  0.14  0.37 -0.14 -0.07  0.18
  -0.17  0.26]
 [-0.13 -0.19  0.11  0.16  0.39 -0.07 -0.01 -0.31 -0.23  0.22 -0.06  0.04
   0.2   0.05 -0.15  0.36  0.1  -0.36  0.01  0.23 -0.16  0.14 -0.33  0.08
  -0.05 -0.03]
 [-0.16 -0.22  0.14  0.17  0.46 -0.02  0.   -0.13 -0.01 -0.21  0.    0.01
  -0.    0.16  0.11 -0.05 -0.13  0.3  -0.12 -0.05  0.18 -0.31  0.02 -0.24
   0.5  -0.05]
 [-0.17 -0.21  0.15  0.13  0.39  0.06 -0.13 -0.08  0.2   0.13  0.08  0.
  -0.1  -0.12  0.07 -0.14  0.16  0.27  0.27 -0.22 -0.11  0.29  0.35  0.09
  -0.37  0.07]]
```

Interpretation 2 :-

* Best axis to project on: ('best' = min sum of squares of projection errors)

* Best axis to project on: ('best' = min sum of squares of projection errors)
* Consider v1(first row) from Vh matrix to explain the first eigen vector and coordinates of points by using U*S and considering first singular value to explain the variance('spread') on the v1 axis
* Dimension reduction by ignoring the smallest eigen values.(keeping 80-90% of information)

In [26]:

```
v1=Vh[1,:]
v1 #first eigen vector considering row of Vh)
```

Out[26]:

```
array([-9.47272720e-02,  1.03618349e-02,  3.68472441e-02, -1.00234277e-01,
       -8.81490721e-02, -1.60222130e-02, -3.81757069e-01,  4.09018722e-01,
        3.37754296e-01,  3.57049494e-01,  1.40297719e-01,  1.20156593e-01,
        2.67384426e-01, -1.01342986e-01,  7.85448110e-02, -2.02454569e-01,
        1.19962059e-01, -1.30983930e-01,  1.07241581e-04,  1.01273578e-01,
        3.76676800e-01, -2.64873974e-01])
```

In [27]:

```
print("variance ('spread') on the eigen vector v1 :"+str(s[1]))
```

```
variance ('spread') on the eigen vector v1 :58.249211961514824
```

In [28]:

```
U.shape
```

Out[28]:

```
(26, 26)
```

In [29]:

```
S.shape
```

Out[29]:

```
(26, 22)
```

In [30]:

```
print("The coordinates of the points in the projection axis")
np.matrix(U) * np.matrix(S)
```

```
The coordinates of the points in the projection axis
```

Out[30]:

```
matrix([[-1.82287456e+01,  9.07164966e-01,  7.28671604e+00,
         -1.10017015e+01, -4.88323830e+00, -1.71504456e+00,
          4.43019635e+00, -7.04776734e+00,  1.69324363e+00,
         -1.02041454e+00, -4.47750871e-01,  3.27766386e+00,
          6.78154348e-01,  4.92969591e-01, -1.81091797e+00,
         -9.31692393e-01,  2.29494791e-01,  5.65666369e-01,
         -7.59735227e-01, -2.67445503e-03, -7.26128389e-02,
         -1.17342428e-01],
        [-1.89053881e+01, -1.35638207e+00,  1.02310274e+01,
         -1.54832986e+01, -5.43208363e+00,  2.55257723e-01,
          1.85383882e+00,  3.37063992e-01, -5.41754423e+00,
          1.74216790e+00, -1.69708037e-01, -1.10096807e+00,
         -1.36644197e+00,  2.43209836e+00,  1.32480967e+00,
          1.22679205e+00,  1.71058091e-01,  1.06287008e+00,
          1.44511067e-01, -3.49543359e-01, -4.61088434e-01,
          1.84827381e-01],
        [-2.37593842e+01, -6.20844874e+00,  9.02222067e+00,
         -1.12417708e+01, -1.78034629e+00, -8.16075573e+00,
```

```
 -5.76180196e+00,  3.59075251e-01,  1.79884324e+00,
  4.52285170e-01, -4.28623350e-01,  1.33162928e+00,
 -1.05558140e+00,  3.32779720e-01,  1.59168475e+00,
 -1.60573264e+00, -3.74609415e-01, -9.74575440e-01,
  1.79632882e-01,  2.56534938e-01, -3.48626042e-01,
 -1.73653957e-01],
[-2.62657223e+01, -3.67546306e+00,  1.20819072e+01,
 -1.76660190e+01,  6.72040797e-01,  1.65388213e-01,
  1.70475691e-01,  3.76409051e+00,  2.08357652e+00,
 -3.47686709e-01,  1.39518308e+00,  1.14893433e-01,
  1.86143929e+00, -1.37203693e+00,  1.82317424e-02,
  1.70360554e+00,  1.09848779e+00, -4.00298570e-01,
  7.19241112e-02,  9.82204326e-02,  8.86747215e-01,
  1.60166398e-01],
[-2.07766326e+01, -1.64447294e+00,  1.13357745e+01,
 -1.03813609e+01,  6.30356420e+00,  9.45511475e+00,
  1.72447118e+00,  1.93243573e+00, -1.50403658e+00,
 -1.15898961e+00, -5.87650708e-01, -2.32823887e+00,
 -8.03224287e-01, -1.31163976e+00, -9.83804928e-01,
 -1.70917266e+00, -1.04398677e+00, -8.10635816e-01,
 -1.67988975e-01,  8.92440336e-02, -1.91804417e-01,
 -2.57776556e-01],
[-1.02202613e+01,  1.94942144e+01,  9.25740266e+00,
  6.99014206e+00,  1.81172600e+00, -6.25990486e+00,
  2.72040031e+00,  1.71205102e+00, -3.02286564e+00,
 -2.26451832e+00,  3.85654747e+00,  3.32851865e-01,
 -3.47571377e-01, -1.58643876e+00, -1.99406204e+00,
 -8.52064458e-01,  9.46961339e-01, -2.77110540e-01,
  5.99804918e-01, -2.81810538e-01, -3.91280777e-01,
 -6.78953928e-02],
[-9.48447202e+00,  1.97400003e+01,  1.00185025e+01,
  5.06025602e+00,  4.19780439e+00, -5.70946565e+00,
  3.31018653e+00,  1.18814782e+00,  2.67733545e-02,
  4.16480442e-01, -5.63900820e+00, -3.40942572e-01,
 -1.64466851e+00, -2.34600602e+00,  1.06046091e+00,
  7.58873840e-01, -6.54427096e-01,  4.48665321e-01,
 -1.09757133e-01, -1.94123164e-01,  3.66849592e-01,
  1.21376539e-02],
[-1.53924294e+01,  1.96570246e+01,  9.52144565e+00,
  5.58446185e+00, -4.65167352e+00,  7.79661779e+00,
 -3.34172783e+00, -1.39433345e+00,  1.37482841e+00,
 -5.18061626e-02, -1.85316740e+00,  2.77407880e+00,
 -8.59698697e-01, -5.16775174e-02, -7.94315901e-01,
  1.67889446e+00,  4.33753128e-01,  2.59034781e-01,
  7.55481107e-01,  4.30993129e-01, -1.58142540e-01,
 -2.07415758e-01],
[-1.10610285e+01,  1.79047747e+01,  7.74396940e+00,
  6.20495426e+00, -3.01295513e+00,  3.52859597e+00,
 -2.07861620e+00, -3.01789838e+00,  1.19590061e+00,
  1.66638075e-01, -1.15683518e+00, -2.14448682e+00,
  7.40414150e-01, -1.33158212e-02,  1.92071365e+00,
 -1.62293989e+00,  1.17468408e+00, -6.09873571e-01,
 -6.09504433e-01, -2.06519933e-01, -3.07709255e-01,
  3.43170261e-01],
[-1.09904715e+01,  1.85040256e+01,  8.65060160e+00,
  6.80192998e+00,  8.98934158e-01, -1.62223102e+00,
 -1.55869529e+00,  8.21989231e-01,  1.28082073e+00,
 -1.51951223e-03,  2.18080186e+00, -1.18344112e+00,
  2.29849592e+00,  3.11929127e+00,  1.32538541e+00,
  1.02328395e+00, -5.42573637e-01, -2.36032939e-01,
 -1.83019075e-01, -1.86488967e-01,  7.85257956e-02,
 -3.58062241e-01],
[-1.03506479e+01,  1.52215194e+01,  9.12688328e+00,
  4.26154521e+00,  1.02631690e+00,  6.65914840e-01,
 -1.72285327e+00,  7.93424940e-01, -5.66613654e-01,
  1.61307442e+00,  2.97872590e+00, -9.06719089e-02,
  4.38206900e-01,  1.20497093e+00, -1.41018929e+00,
 -8.87836684e-01, -1.52940049e+00,  5.98538143e-01,
 -4.34297282e-01,  3.26566551e-01,  4.80306472e-01,
  4.07342306e-01],
[-2.54014302e+01,  5.40810603e+00, -1.71164994e+01,
 -6.03796723e+00,  1.12566815e+00, -2.74506899e+00,
 -6.88614832e+00, -7.55734456e-01, -9.28355201e-01,
 -3.17223005e+00, -2.17844638e+00, -2.34531819e+00,
  4.29812430e-01,  1.06631132e-01, -2.12002353e+00,
  7.18980846e-01, -7.93598521e-01,  4.02793259e-01,
  2.59008697e-01,  1.37413671e-01, -1.89508600e-01,
  5.41840245e-02],
```

```
5.41242345e-02],
[-1.97011997e+01,  6.81968720e+00, -1.62853538e+01,
 -3.82853526e+00, -4.34044835e-01,  1.29348966e+00,
 -1.67770970e+00, -3.65181001e+00, -2.06920411e+00,
 -1.84249519e+00,  4.06423612e-01,  3.45685667e-01,
  2.26961642e+00, -1.04601609e+00,  1.16792023e+00,
 -9.28483297e-01,  4.24676419e-01,  3.53362512e-01,
  4.20112284e-01, -5.08996399e-01,  5.28922509e-01,
  1.38496154e-01],
[-1.85891933e+01,  8.00323201e+00, -1.53938189e+01,
 -1.40539729e+00,  6.38567891e+00, -3.60333455e-01,
  2.76042406e+00,  4.99956734e+00,  1.82738745e-01,
  3.80127008e+00, -3.16545619e+00,  8.24554428e-01,
  2.22489116e+00,  2.17575278e+00, -1.08681305e+00,
 -1.86572814e+00,  1.11309358e+00,  2.17562126e-01,
  9.00903890e-02,  2.97005286e-01, -6.64130814e-02,
 -8.22995768e-02],
[-1.96406884e+01,  8.13386505e+00, -1.51270988e+01,
 -4.23061763e+00,  3.82169603e+00, -1.08648506e+00,
  3.72889454e+00, -1.46679432e+00,  4.55221480e+00,
  1.22336946e-01,  1.98870506e+00, -3.60551695e+00,
 -2.18563161e+00, -5.01045726e-02, -3.54384622e-01,
  1.48920629e+00,  5.11347189e-01,  4.93893459e-02,
 -5.63889245e-01,  6.01053276e-02, -3.37282356e-01,
 -5.67324684e-04],
[-1.78954288e+01,  7.98425512e+00, -1.49874694e+01,
 -2.28659838e+00,  2.41535275e+00,  1.80478048e+00,
  4.59119933e+00, -1.91150208e+00, -7.38665497e-01,
  8.94197947e-01,  1.75725288e+00,  1.26828691e+00,
  1.08445305e+00, -9.19175902e-01,  2.46348088e+00,
  3.24663404e-01, -1.11611451e+00, -4.84441622e-01,
  7.32230600e-01, -1.27822621e-02, -1.40562533e-01,
 -2.34871592e-01],
[-1.75360304e+01,  7.40678284e+00, -1.49789460e+01,
  3.33119870e-01,  5.14146673e+00,  1.59430793e+00,
 -3.31976059e+00,  1.58646056e+00, -1.99640775e+00,
  1.45840593e+00,  1.90933435e+00,  3.76240710e+00,
 -4.01906844e+00, -3.15568648e-01,  4.81601735e-01,
  2.83946979e-01,  2.20578925e-01, -3.46264558e-01,
 -9.69261913e-01, -6.97665867e-02,  1.99550337e-01,
  4.02773138e-02],
[-2.68426792e+01, -1.27493356e+01, -1.25130492e+00,
  1.02493435e+01, -5.17223558e+00,  1.45276197e+00,
  9.48753596e-02,  4.11034396e+00,  3.54731794e-01,
 -2.05891055e+00, -1.14640433e+00,  1.60377403e+00,
  1.56320571e+00,  1.00786330e-01, -5.34729866e-01,
  1.01694405e+00, -1.14686618e-01, -3.68683551e-01,
 -6.84450245e-01, -1.32126439e+00, -3.93930457e-01,
  1.03789349e-02],
[-2.26675704e+01, -7.90001101e+00, -1.53750078e+00,
  9.27151354e+00, -4.76437537e+00, -7.77200003e-01,
  1.99004533e+00,  1.69935031e+00, -1.75981942e+00,
 -2.64137857e+00,  1.40635282e-02,  2.90606543e-01,
  1.43763610e+00, -1.46309294e+00,  1.56142720e+00,
 -1.19127192e-02, -1.93934850e-01,  3.56293059e-01,
 -6.46466902e-01,  1.33690880e+00, -2.47620118e-01,
  2.44079673e-01],
[-2.31842060e+01, -6.00204248e+00, -2.22422707e+00,
  6.15169957e+00, -6.59682925e+00, -9.80941135e-01,
  3.21772133e+00, -1.85132280e+00,  1.00880795e+00,
  1.72726223e+00, -8.43310695e-01, -6.44710532e-02,
 -1.06319509e+00,  9.89379192e-01, -1.02798670e+00,
 -2.59348965e-01, -1.61611283e+00, -1.07223098e+00,
  5.55948267e-01, -3.58827352e-01,  3.55606158e-01,
  3.74742847e-01],
[-2.19156631e+01, -8.06434800e+00, -6.87024033e-01,
  5.11960491e+00, -4.79965560e+00,  9.81637871e-01,
  1.05034184e+00,  2.70328434e+00,  3.51384579e+00,
  7.47031780e-02,  1.58620123e+00,  7.06051471e-01,
 -5.07370965e-01, -1.88569595e-02,  2.76383584e-01,
 -2.31036010e-01, -4.44322667e-01,  9.51208758e-01,
  6.54868772e-01,  3.87218239e-01, -3.69413446e-01,
  6.41530970e-02],
[-2.10921819e+01, -9.46717329e+00,  4.21534010e-01,
  7.53270479e+00, -9.73622584e+00, -9.59035789e-01,
 -2.57774262e+00, -1.54613000e+00, -1.36390465e+00,
  5.72836245e+00,  9.82293165e-01, -2.23364015e+00,
```

```
 6.67344189e-01, -2.74345412e+00, -7.27435309e-01,
-1.21981167e-01,  2.11763464e-01,  5.27549764e-01,
-3.09431572e-01, -9.46393423e-02,  1.83351939e-01,
-4.67134648e-01],
[-2.18920257e+01, -9.88277527e+00, -1.05786355e+00,
 1.06242801e+01, -5.71495955e+00,  2.27900612e-01,
 1.55931874e+00, -6.31898900e-01, -1.34406059e+00,
-2.42539813e+00, -5.02522456e-01, -1.56381326e+00,
-2.85505075e+00,  2.34339463e+00,  1.14220085e-01,
-5.98530895e-01,  1.47052489e+00, -4.41216059e-01,
 4.06447220e-01,  3.22965888e-01,  6.89452093e-01,
-1.70078934e-01],
[-1.30851953e+01, -1.10215067e+01,  5.76574456e+00,
 6.73189234e+00,  1.21663903e+01, -1.33915137e+00,
-1.05021869e-01, -4.22393976e+00, -2.49717657e+00,
 2.27853605e+00, -6.24110434e-01,  3.39933181e-01,
 1.63691973e+00,  3.64400290e-01, -9.62999820e-01,
 2.09390846e+00,  4.23339766e-01, -1.19231044e+00,
 1.98041811e-02,  5.43908182e-01, -3.02893574e-01,
 1.70878206e-01],
[-1.59229628e+01, -1.30997375e+01,  7.09842613e+00,
 7.00329158e+00,  1.45821594e+01, -3.44923184e-01,
 4.98166893e-02, -1.79743039e+00, -1.14938940e-01,
-2.19300563e+00,  6.21241469e-03,  1.19432630e-01,
-2.48167956e-02,  1.16766278e+00,  7.13356497e-01,
-3.12904319e-01, -5.42614293e-01,  9.79566493e-01,
-3.17117342e-01, -1.23950889e-01,  3.40494936e-01,
-3.70581437e-01],
[-1.67440900e+01, -1.24024728e+01,  7.51091962e+00,
 5.32078649e+00,  1.24424063e+01,  1.17662706e+00,
-2.03106280e+00, -1.07843149e+00,  2.13904997e+00,
 1.35395878e+00,  8.05249395e-01,  3.21879444e-02,
-8.13680949e-01, -9.18111740e-01,  4.62100872e-01,
-8.21165769e-01,  6.72335123e-01,  8.80494970e-01,
 6.94147122e-01, -5.23533351e-01, -2.13284448e-01,
 3.51425469e-01]])
```

Note:- Considering only first four eigen vectors to keep 80-90% of information and resultant matrices will be as mentioned above when first 4 eigen vectors are considered

Interpretation 3 :-

```
* Finds non-zero ‘blobs’ in  a data matrix
```

Observation:

```
* As data contains docs without any zero blocks and documents are spread with terms and con
cepts cannot differentiate the zero blocks
```

Interpretation 4 :-

```
    * Fixed point operation - transpose(A)* A * v(i) = (eigen_value(i))^2 * v(i)
    * Convergence -  transpose(A)* A * v'=(constant)v1 [v1--> strongest right vector)
```

Here first row of Vh matrix can be considered as strongest right vector and next 3 can be considered as useful right vectors if we fix the rank or no of hidden features to 4 and first column of U matrix can considered as strongest left vector and next 3 columns can be considered as useful left vectors which satisfies A.v(i)=eigne_value(i).u(i)

Observations:

```
* Found dim. reduction: keep the first four strongest eigenvalues (~87%)
* Picked up linear correlations.
* Unable to find non-zero ‘blobs’
```

5g) Consider using only the top two Eigen-values and Eigen vectors. Show and interpret how doc-1, doc-8, and doc-16 are

summarized in this case?

In [31]:

```
top2_U=U[:,:2]
top2_Vh=Vh[:2,:]
top2_s=s[:2]
A_approx_top2 = np.matrix(top2_U) * np.diag(top2_s) * np.matrix(top2_Vh)
print("A calculated using only the first two components:\n")
print(pd.DataFrame(A_approx_top2, index=df.index, columns=df.columns))
```

```
A calculated using only the first two components:

           data   program  complexity   concept  algorithm  processor  \
doc1   4.401943  3.450646    1.939520  1.647805   2.846464   0.777928
doc2   4.782951  3.554929    1.926868  1.939231   3.154621   0.843611
doc3   6.437617  4.420996    2.255642  2.888569   4.361583   1.132371
doc4   6.814729  4.920392    2.611052  2.873742   4.540668   1.200746
doc5   5.270937  3.905200    2.111920  2.146594   3.480424   0.929576
doc6   0.669572  2.131390    1.786993 -0.979137  -0.077644   0.131968
doc7   0.465140  1.995033    1.719111 -1.073956  -0.217433   0.096042
doc8   1.927526  3.109486    2.333821 -0.502113   0.738341   0.354210
doc9   1.027130  2.273642    1.816341 -0.739625   0.197440   0.193985
doc10  0.952994  2.266531    1.831044 -0.806421   0.133290   0.181316
doc11  1.106413  2.111732    1.643189 -0.538430   0.319923   0.206094
doc12  5.741480  4.851353    2.855382  1.880817   3.601207   1.017633
doc13  4.204381  3.789882    2.311348  1.195616   2.561667   0.747209
doc14  3.818494  3.592220    2.238681  0.970917   2.278818   0.679903
doc15  4.064995  3.792076    2.353445  1.058119   2.436109   0.723522
doc16  3.649488  3.461054    2.165438  0.906645   2.169114   0.650047
doc17  3.615707  3.387222    2.106579  0.930246   2.162320   0.643675
doc18  7.816318  4.935289    2.337034  3.838287   5.433146   1.371211
doc19  6.329052  4.197355    2.079147  2.953979   4.335414   1.112009
doc20  6.276457  4.314552    2.203104  2.813017   4.251050   1.104059
doc21  6.159501  4.053706    1.994468  2.898732   4.229189   1.081954
doc22  6.089648  3.883712    1.856671  2.960796   4.220646   1.068631
doc23  6.325936  4.030401    1.924993  3.078746   4.385687   1.110062
doc24  4.265583  2.356037    0.962145  2.352853   3.072223   0.745445
doc25  5.161100  2.870220    1.182300  2.831841   3.710990   0.902110
doc26  5.297210  3.032459    1.293854  2.840274   3.781350   0.926635

            game      gene      drug   disease  ...  candidate   optimal  \
doc1    5.643962  2.043064  2.796904  1.874120  ...   3.609867  2.829796
doc2    6.730443  1.179296  2.124828  1.123465  ...   3.129623  3.167643
doc3   10.177860 -0.360062  1.149199 -0.196168  ...   2.728910  4.437372
doc4   10.034501  0.905869  2.347155  0.921379  ...   3.869174  4.582391
doc5    7.455347  1.233099  2.283183  1.179737  ...   3.398257  3.496765
doc6   -4.083501  8.910944  7.980599  7.829555  ...   7.100387 -0.337483
doc7   -4.419125  8.943985  7.963087  7.854740  ...   7.030188 -0.480325
doc8   -2.445992  9.451949  8.742237  8.327540  ...   8.099349  0.475020
doc9   -3.200431  8.337952  7.558629  7.333547  ...   6.830707 -0.041645
doc10  -3.452385  8.576585  7.751389  7.541509  ...   6.977903 -0.113684
doc11  -2.409522  7.175291  6.555292  6.315080  ...   5.982021  0.116423
doc12   6.282763  4.541942  5.297085  4.091161  ...   6.138323  3.523305
doc13   3.870688  4.596456  4.995058  4.110404  ...   5.462783  2.466610
doc14   3.053437  4.978550  5.242877  4.438420  ...   5.573829  2.168432
doc15   3.349106  5.128429  5.430660  4.574484  ...   5.802995  2.323729
doc16   2.832699  4.907153  5.141682  4.372645  ...   5.440599  2.059158
doc17   2.935049  4.637991  4.897535  4.135895  ...   5.219802  2.060075
doc18  13.688114 -2.752594 -0.638758 -2.269377  ...   1.549541  5.594440
doc19  10.464837 -1.152088  0.428698 -0.892990  ...   2.074927  4.433803
doc20   9.910050 -0.328395  1.140330 -0.171385  ...   2.677850  4.324264
doc21  10.280485 -1.288273  0.270463 -1.015610  ...   1.892090  4.329940
doc22  10.545413 -1.937588 -0.315856 -1.586519  ...   1.364879  4.340118
doc23  10.966914 -2.034212 -0.346949 -1.666889  ...   1.401505  4.510436
doc24   8.507558 -3.307774 -1.934794 -2.822427  ...  -0.529816  3.214267
doc25  10.233476 -3.897517 -2.249015 -3.323127  ...  -0.561296  3.879723
doc26  10.237127 -3.537006 -1.901325 -3.004339  ...  -0.223176  3.940671

         result  training  campaign        win       run      best  \
doc1   6.055065  5.359726  2.657387   6.188463  6.206104  5.295130
doc2   6.099392  6.023759  2.480448   6.719074  6.436225  5.259035
doc3   7.311671  8.482189  2.577022   9.034137  8.088256  6.153186
doc4   8.333361  8.731555  3.231295   9.569569  8.941815  7.125126
doc5   6.691023  6.651133  2.707507   7.404275  7.073263  5.763992
doc6   4.886097 -0.838698  3.767461   0.982860  3.481596  4.891546
```

```
doc7    4.663870   -1.112213  3.694075   0.696078  3.231121  4.706412
doc8    6.596713    0.701204  4.510113   2.751145  5.242486  6.384391
doc9    5.037247   -0.261230  3.694336   1.481963  3.767666  4.970568
doc10   5.061154   -0.404007  3.756359   1.379058  3.743709  5.011116
doc11   4.593300    0.065980  3.273129   1.587630  3.525528  4.496052
doc12   8.763114    6.629713  4.200142   8.080716  8.648539  7.798346
doc13   7.002814    4.610484  3.572529   5.923498  6.708038  6.314214
doc14   6.730746    4.032708  3.559040   5.383709  6.329580  6.116661
doc15   7.086173    4.326021  3.721720   5.730424  6.687577  6.430033
doc16   6.501518    3.825575  3.459768   5.146147  6.093385  5.916710
doc17   6.338184    3.833193  3.340246   5.097432  5.970965  5.755640
doc18   7.810049   10.744056  2.223439  10.957732  9.137268  6.370870
doc19   6.820407    8.492632  2.221452   8.877929  7.716365  5.670225
doc20   7.139074    8.265489  2.521367   8.808085  7.892458  6.009909
doc21   6.560676    8.297246  2.096614   8.639288  7.460359  5.438956
doc22   6.180173    8.330833  1.813197   8.538105  7.179853  5.061830
doc23   6.410089    8.658207  1.875166   8.869294  7.452117  5.248050
doc24   3.429696    6.210580  0.507279   5.971219  4.453695  2.618883
doc25   4.197994    7.494296  0.654718   7.225323  5.419594  3.218433
doc26   4.522306    7.602837  0.853165   7.418109  5.699223  3.523432

       expression     sport
doc1     4.053590  3.521355
doc2     3.338748  4.260540
doc3     2.499496  6.547385
doc4     3.963974  6.393665
doc5     3.611268  4.722993
doc6     9.424149 -3.054482
doc7     9.366904 -3.271420
doc8    10.538674 -2.030290
doc9     8.996648 -2.459982
doc10    9.208004 -2.633268
doc11    7.841274 -1.895850
doc12    7.209549  3.809310
doc13    6.580533  2.259134
doc14    6.799911  1.716172
doc15    7.063231  1.898555
doc16    6.651493  1.578035
doc17    6.360789  1.656828
doc18    0.663541  8.916157
doc19    1.640000  6.770132
doc20    2.460122  6.374021
doc21    1.424989  6.658498
doc22    0.728893  6.860139
doc23    0.735216  7.135275
doc24   -1.487034  5.619539
doc25   -1.692007  6.755603
doc26   -1.262159  6.740362

[26 rows x 22 columns]
```

In [45]:

```
A_approx_top2_df=pd.DataFrame(A_approx_top2, index=df.index, columns=df.columns)
print("**********doc8 using 2 eigen vectors**********")
print(A_approx_top2_df.loc['doc8'])
print("**********doc8 from original data**********")
print(df.loc['doc8'])
```

```
**********doc8 using 2 eigen vectors**********
data           1.927526
program        3.109486
complexity     2.333821
concept       -0.502113
algorithm      0.738341
processor      0.354210
game          -2.445992
gene           9.451949
drug           8.742237
disease        8.327540
election       4.825426
vote           4.244648
candidate      8.099349
optimal        0.475020
result         6.596713
training       0.701204
```

```
campaign       4.510113
win            2.751145
run            5.242486
best           6.384391
expression    10.538674
sport         -2.030290
Name: doc8, dtype: float64
**********doc8 from original data**********
data           1
program        0
complexity     2
concept        0
algorithm      0
processor      0
game           0
gene          16
drug           4
disease       10
election       0
vote           0
candidate      4
optimal        2
result        12
training       0
campaign       0
win            1
run            6
best           5
expression    16
sport          0
Name: doc8, dtype: int64
```

Observations about doc8:

```
* From original data doc8 contains more words about medical terms (gene, disease,
drug,result etc.)
* With top two eigen vectors also we are able to say doc8 contains more words about medical
terms(may be latent feature to identify as medical terms) as observed above
```

In [46]:

```python
A_approx_top2_df=pd.DataFrame(A_approx_top2, index=df.index, columns=df.columns)
print("**********doc1 using 2 eigen vectors**********")
print(A_approx_top2_df.loc['doc1'])
print("**********doc1 from original data**********")
print(df.loc['doc1'])
```

```
**********doc1 using 2 eigen vectors**********
data          4.401943
program       3.450646
complexity    1.939520
concept       1.647805
algorithm     2.846464
processor     0.777928
game          5.643962
gene          2.043064
drug          2.796904
disease       1.874120
election      2.575852
vote          2.338653
candidate     3.609867
optimal       2.829796
result        6.055065
training      5.359726
campaign      2.657387
win           6.188463
run           6.206104
best          5.295130
expression    4.053590
sport         3.521355
Name: doc1, dtype: float64
```

```
**********doc1 from original data**********
data            6
program        12
complexity      3
concept         4
algorithm       8
processor       2
game            2
gene            0
drug            0
disease         0
election        0
vote            0
candidate       1
optimal         2
result          3
training       10
campaign        0
win             2
run             7
best            6
expression     12
sport           0
Name: doc1, dtype: int64
```

Observations about doc1:

```
* From original data doc1 contains more words about computer science/study (data, program,
algorithm etc.)
* However, with top two eigen vectors we are not able to say doc1 contains particular set o
f words that are related to single concept and failing in this case so may be we need
another eigen may be 3rd or 4th
```

In [48]:

```python
A_approx_top2_df=pd.DataFrame(A_approx_top2, index=df.index, columns=df.columns)
print("**********doc16 using 2 eigen vectors**********")
print(A_approx_top2_df.loc['doc16'])
print("**********doc16 from original data**********")
print(df.loc['doc16'])
```

```
**********doc16 using 2 eigen vectors**********
data          3.649488
program       3.461054
complexity    2.165438
concept       0.906645
algorithm     2.169114
processor     0.650047
game          2.832699
gene          4.907153
drug          5.141682
disease       4.372645
election      3.523979
vote          3.148243
candidate     5.440599
optimal       2.059158
result        6.501518
training      3.825575
campaign      3.459768
win           5.146147
run           6.093385
best          5.916710
expression    6.651493
sport         1.578035
Name: doc16, dtype: float64
**********doc16 from original data**********
data            0
program         5
complexity      1
concept         1
algorithm       0
processor       0
```

```
game            0
gene            0
drug            6
disease         0
election        8
vote            9
candidate       4
optimal         0
result          5
training        0
campaign       12
win             9
run            10
best            9
expression      3
sport           0
Name: doc16, dtype: int64
```

Observations about doc16:

```
* From original data doc16 contains more words about election/politics (vote, compaign,
election, candidate etc.)
* However, with top two eigen vectors we are not able to say doc16 contains particular set
of words that are related to single concept and failing in this case so may be we need
another eigen may be 3rd or 4th
```

In [50]:

```
A_approx_top2_df=pd.DataFrame(A_approx_top2, index=df.index, columns=df.columns)
print("**********doc18 using 2 eigen vectors**********")
print(A_approx_top2_df.loc['doc18'])
print("**********doc18 from original data**********")
print(df.loc['doc18'])
```

```
**********doc18 using 2 eigen vectors**********
data            7.816318
program         4.935289
complexity      2.337034
concept         3.838287
algorithm       5.433146
processor       1.371211
game           13.688114
gene           -2.752594
drug           -0.638758
disease        -2.269377
election        1.816944
vote            1.751349
candidate       1.549541
optimal         5.594440
result          7.810049
training       10.744056
campaign        2.223439
win            10.957732
run             9.137268
best            6.370870
expression      0.663541
sport           8.916157
Name: doc18, dtype: float64
**********doc18 from original data**********
data            2
program         1
complexity      2
concept         0
algorithm       0
processor       0
game           15
gene            0
drug            0
disease         0
election        0
vote            0
candidate       0
optimal         6
```

```
result         10
training       12
campaign        0
win            16
run            10
best            6
expression      0
sport          12
Name: doc18, dtype: int64
```

Observations about doc18:

```
* From original data doc18 contains more words about sports/games
(sport,win,game,run,training etc.)
* With top two eigen vectors also we are able to say doc18 contains more words about sports
terms(may be latent feature to identify as sports terms) as observed above.
```

Final Conclusion about two eigen vectors which we have considered

```
* Here we are considereing two eigen values and corresponding eigen vectors and from abouve
four docs we can just say that two eigen vectors considered may be relevant hidden
features/ latent features which are able to cluster the sports terms and medical terms.
* If we consider 3rd and 4th eigen vector then other hidden features like identifying elect
ion terms and computer science terms can be easily summarized for the docs which is purely
my observation from the results.
```

5h) Compute the MSE if we use only the top two Eigen-vectors and compare it to the value obtained in #5d.

In [32]:

```
df_list=df.values.tolist()
A_approx_list_top2=pd.DataFrame(A_approx_top2, index=df.index, columns=df.columns).values.tolist()
mean_squared_error(A_approx_list_top2, df_list)
```

Out[32]:

```
11.833550835132428
```

Observation:

```
* Mse is increased from ~4 to ~11 indicating that even though we are resolving the
complexity but we are not getting good accuracy i.e., we are missing most of the
information which is actually cannot be ignored when we consider only top 2.
```