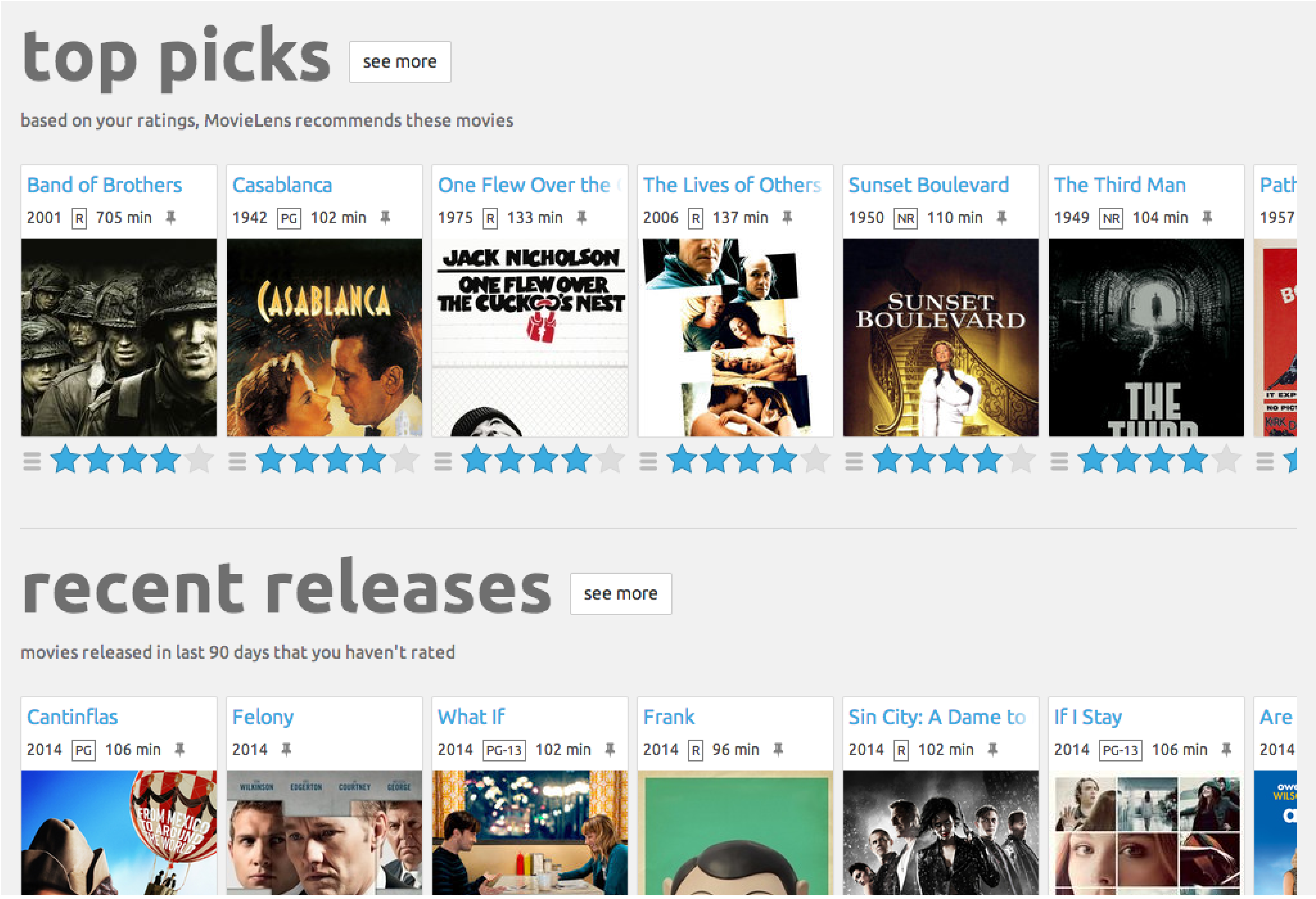
Use Case: Ein Recommender System für Filme

- Collaborativer Item-Item Filter
- Szenario:
  - Eingabe: ein Film (den der User gerade gesehen hat)
  - Ausgabe: Liste von Filmen die als nächstes geschaut werden könnten

"User die diesen Film gesehen haben, sahen auch ..."

Daten: freie Datenbank mit Filmbewertungen: http://www.movielens.org/

- 943 User, 1682 Filme und 100000 Bewertungen
- Download als CSV Datei



Hochschule Offenburg
offenburg.university

Movielens Datenstrukturen

- Relationale Datendank mit drei Tabellen [users, ratings, movies]

	user_id	age	sex	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213

	user_id	item_id	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

	movie id	movie title	release date	video release date	IMDb URL	unknown	Action	Adventure	Animation	Childrens	...	Fantasy	Film-Noir
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	0	0	0	1	1	...	0	0
1	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...	0	1	1	0	0	...	0	0
2	3	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...	0	0	0	0	0	...	0	0
3	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...	0	1	0	0	0	...	0	0
4	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)	0	0	0	0	0	...	0	0

Datenimport:

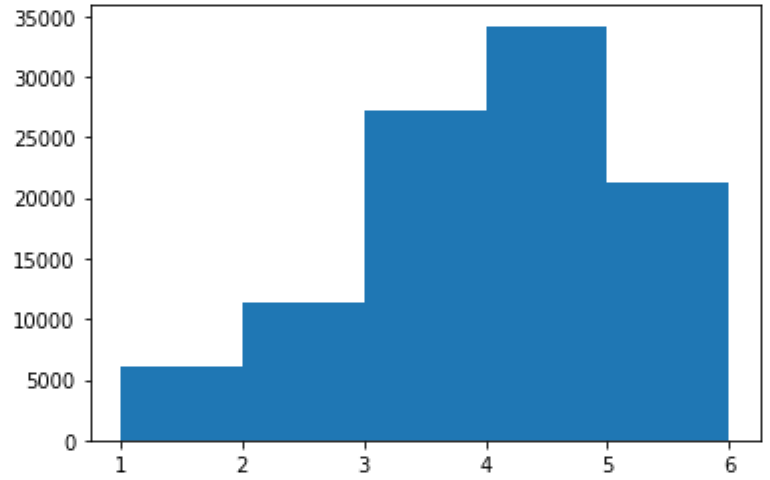
In [2]:
```
#read data to DataFrames
import pandas as pd
u_cols = ['user_id', 'age', 'sex', 'occupation', 'zip_code']
users = pd.read_csv(path+'/DATA/movielens100k/u.user', sep='|', names=u_cols, encoding = "ISO-8859-1")

r_cols = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_csv(path+'/DATA/movielens100k/u.data', sep='\t', names=r_cols, encoding = "ISO-8859-1")

m_cols=['movie_id', 'title', 'release date', 'video release date', 'IMDb_URL', 'unknown', 'Action', 'Adventure', 'Animation', 'C
movies = pd.read_csv(path+'/DATA/movielens100k/u.item', sep='|', names=m_cols ,encoding = "ISO-8859-1" )
```

Dann schauen wir uns die Daten doch mal an ...

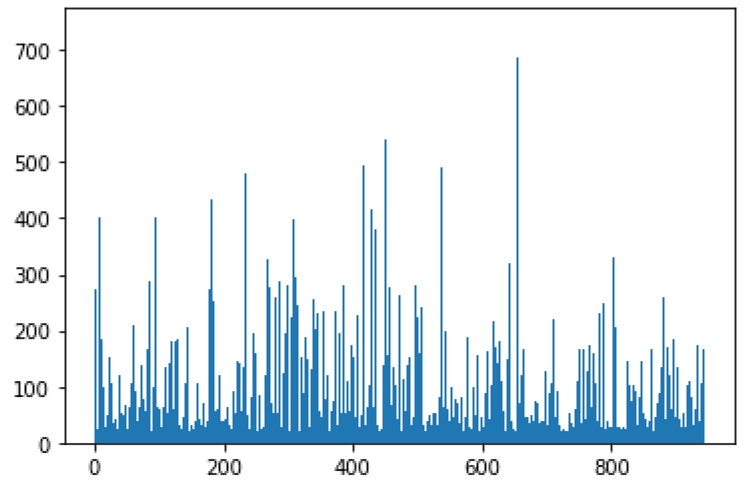
In [3]:
```
import numpy as np
print (np.shape(users), np.shape(ratings), np.shape(movies))
print (ratings[1:10])
print ("hello")
```

```
(943, 5) (100000, 4) (1682, 24)
   user_id  movie_id  rating  timestamp
1      186       302       3  891717742
2       22       377       1  878887116
3      244        51       2  880606923
4      166       346       1  886397596
5      298       474       4  884182806
6      115       265       2  881171488
7      253       465       5  891628467
8      305       451       3  886324817
9        6        86       3  883603013
hello
```

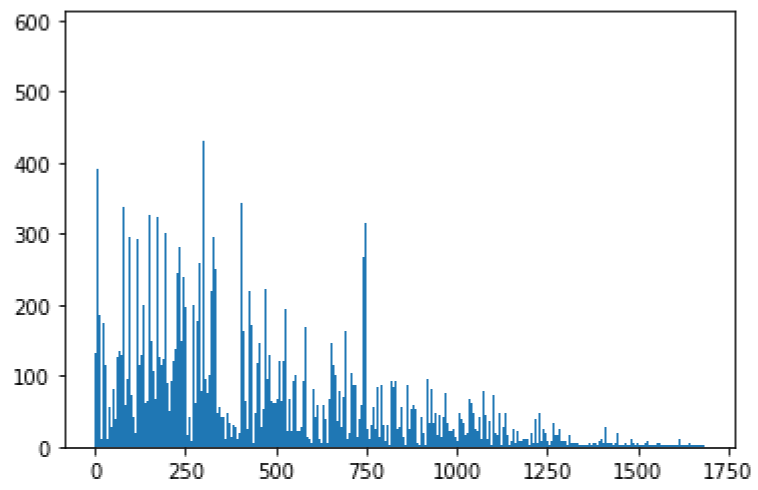
In [4]:
```python
#schauen wir uns mal die die Verteilung der ratings an
import matplotlib.pyplot as plt
%matplotlib inline
res=plt.hist(ratings['rating'],[1,2,3,4,5,6])
```



```
In [5]: #wie viele Bewertungen geben user ab?
        res=plt.hist(ratings['user_id'],943)
```



In [6]:
```
#wie oft werden Filme bewertet?
res=plt.hist(ratings['movie_id'],1682)
```



Formalisierung: ein Item-Item Collaborative Filter

Formalisierung: ein Item-Item Collaborative Filter

- User: $U := \{u_1, \ldots, u_n\}, |U| = n$
- Filme (Produkte): $P := \{p_1, \ldots, p_m\}, |P| = m$
- Kontext: Matrix R der Größe $n \times m$
  mit Bewertungen r_{ij}, mit $i \in 1 \ldots n, j \in 1 \ldots m$

Formalisierung: ein Item-Item Collaborative Filter

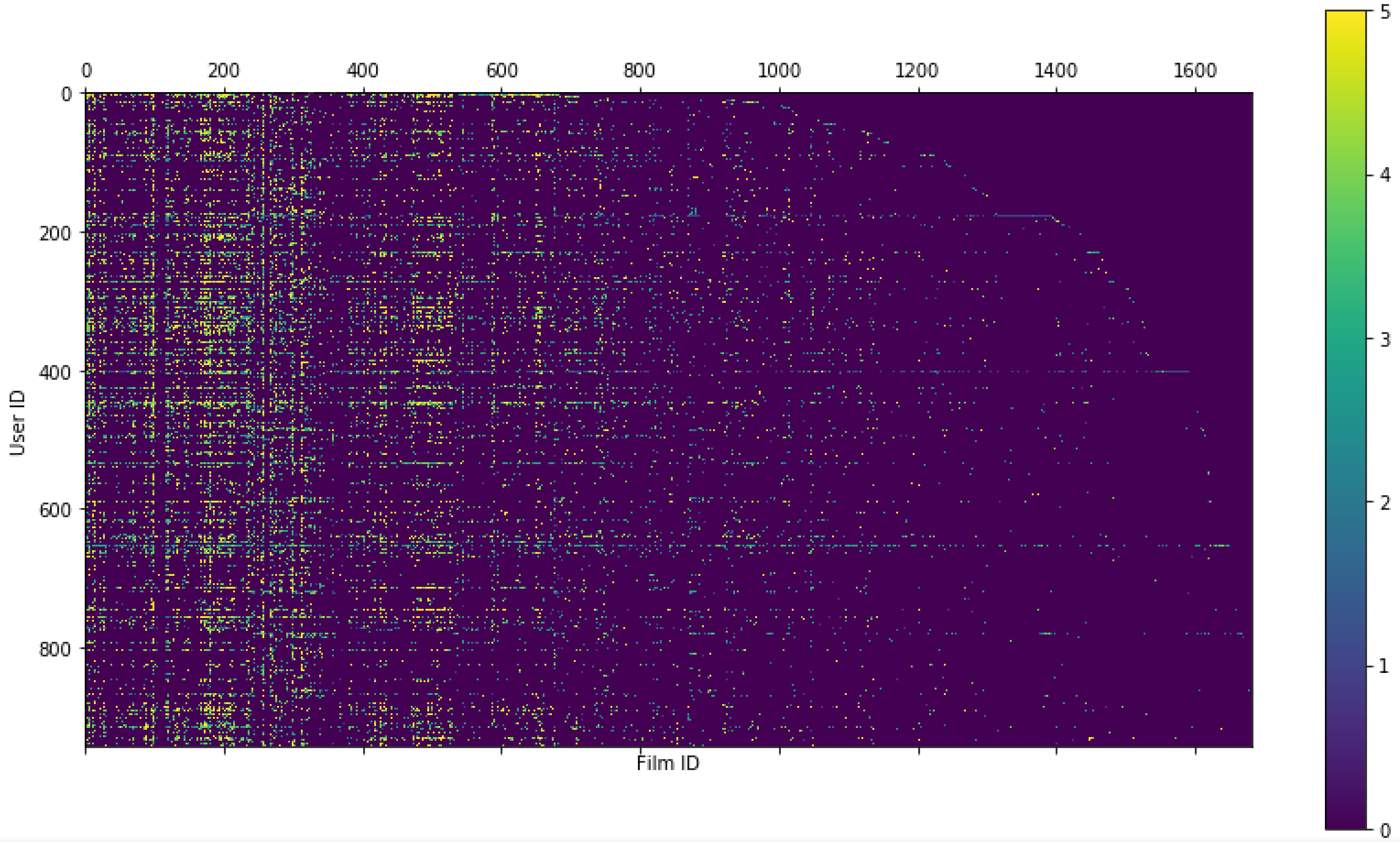
- User: $U := \{u_1, \ldots, u_n\}, |U| = n$
- Filme (Produkte): $P := \{p_1, \ldots, p_m\}, |P| = m$
- Kontext: Matrix R der Größe $n \times m$
  mit Bewertungen r_{ij}, mit $i \in 1 \ldots n, j \in 1 \ldots m$

In [7]:
```
#gerate matrix (this can be done more efficiently!)
R=np.zeros((np.shape(users)[0],np.shape(movies)[0]))
for i in range(np.shape(ratings)[0]):
    R[ratings['user_id'][i]-1, ratings['movie_id'][i]-1]=ratings['rating'][i]
```

Plot R

In [8]:
```
plt.rcParams['figure.figsize'] = (10.0, 8.0)
#show matrix
plt.matshow(R)
plt.xlabel('Film ID')
plt.ylabel('User ID')
plt.colorbar()
```

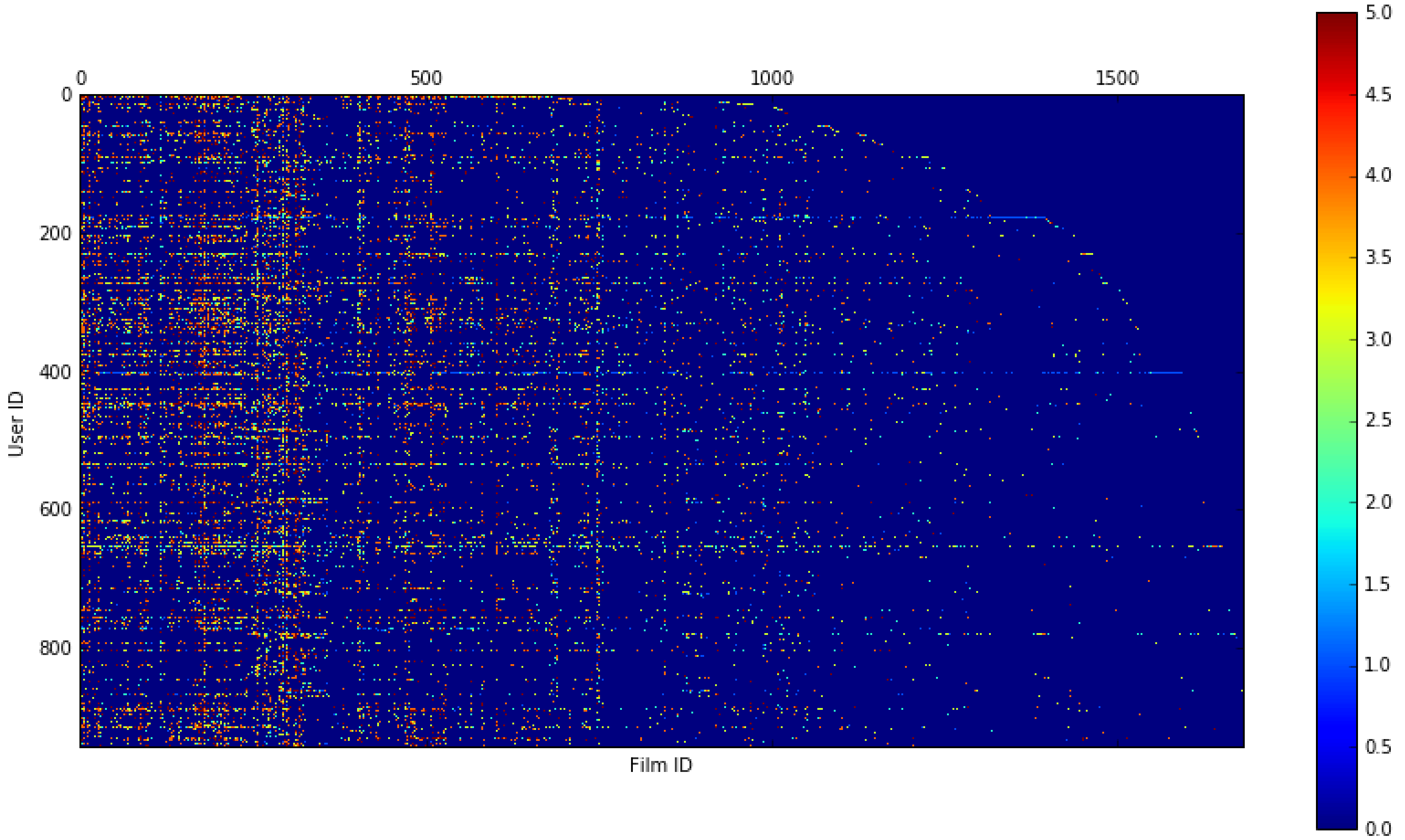
Out[8]: `<matplotlib.colorbar.Colorbar at 0x7f52d5e06e10>`



Distanzmaße

In R sind alle * Bewertungen des i-ten Films im i-ten Spaltenvektor abgelegt * Bewertungen des j-ten Users im j-ten Zeilenvektor abgelegt

Filmbewertungen := Spaltenvektoren in der Kontext Matrix R



Items: Kosinus-Distanz:

Für Filmbewertungen := Spaltenvektoren $\vec{a}, \vec{b}$ in R gilt die Distanz:

$$d_{cos}(\vec{a}, \vec{b}) := \frac{<\vec{a},\vec{b}>}{|\vec{a}||\vec{b}|}$$

- mit den Skalarprodukt $< \vec{a}, \vec{b} >$
- und der Vektornorm $|\vec{a}|$
- mit dem Wertebereich zwischen 0 (keine Ähnlichkeit) und 1 (Identisch)

Items: Kosinus-Distanz:

Für Filmbewertungen := Spaltenvektoren $\vec{a}, \vec{b}$ in R gilt die Distanz:

$$d_{cos}(\vec{a}, \vec{b}) := \frac{<\vec{a},\vec{b}>}{|\vec{a}||\vec{b}|}$$

- mit den Skalarprodukt $< \vec{a}, \vec{b} >$
- und der Vektornorm $|\vec{a}|$
- mit dem Wertebereich zwischen 0 (keine Ähnlichkeit) und 1 (Identisch)

In [9]:

```
def CosineDist(a,b):
    res =  a.dot(b)
    norm = np.linalg.norm(a)*np.linalg.norm(b)
    if norm > 0: #norm ist null wenn keine Berwerung existiert -> Fallunterscheidung
        return res/norm
    else:
        return res
```

Erstelle Kosinus-Distanz Matrix für alle Filme (offline)

In [30]:
```
#implementation with for-loops is not efficient!
#D=np.zeros((np.shape(movies)[0],np.shape(movies)[0]))
#for i in range(0,np.shape(movies)[0]):
#    for j in range(0,np.shape(movies)[0]):
#        if i!=j:
#            D[i,j]=CosineDist(R[:,i],R[:,j])

import scipy.spatial
D=scipy.spatial.distance.squareform(scipy.spatial.distance.pdist(R.T, metric='cosine'))
D=np.abs(np.nan_to_num( D-1)) #dist to similarity
np.fill_diagonal(D,0) #set self-dist to zero
```

In [31]:
```
np.save("movie_dist",D)
```

Erstelle Kosinus-Distanz Matrix für alle Filme (offline)

In [30]:
```python
#implementation with for-loops is not efficient!
#D=np.zeros((np.shape(movies)[0],np.shape(movies)[0]))
#for i in range(0,np.shape(movies)[0]):
#    for j in range(0,np.shape(movies)[0]):
#        if i!=j:
#            D[i,j]=CosineDist(R[:,i],R[:,j])

import scipy.spatial
D=scipy.spatial.distance.squareform(scipy.spatial.distance.pdist(R.T, metric='cosine'))
D=np.abs(np.nan_to_num( D-1)) #dist to similarity
np.fill_diagonal(D,0) #set self-dist to zero
```

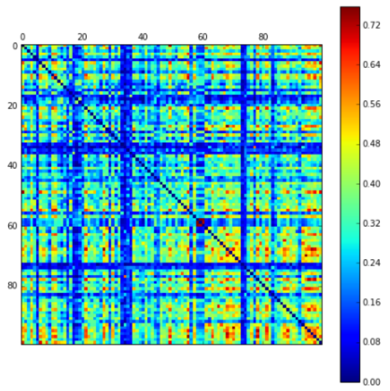
In [31]:
```python
np.save("movie_dist",D)
```

In [13]:
```python
plt.rcParams['figure.figsize'] = (10.0, 8.0)
plt.matshow(D)
plt.colorbar()
```

Out[13]:
```
<matplotlib.colorbar.Colorbar at 0x7f52cffced90>
```

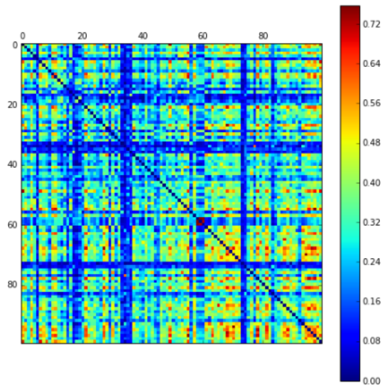
Anfragen an die Distanz-Matrix

- Suche die Top-5 ähnlichsten Filme
- argmax auf Spalten / Zeilen von D



Anfragen an die Distanz-Matrix

- Suche die Top-5 ähnlichsten Filme
- argmax auf Spalten / Zeilen von D



In [14]:
```
def getTopN(movie_id,D, N=5):
    return D[movie_id,:].argsort()[-N:]
```

Hilfsfunktionen

In [15]:
```
def getIDbyName(name):
    if np.size(movies.movie_id[movies.title.str.contains(name)]) > 0:
        m = int(movies.movie_id[movies.title.str.contains(name)][:1]), str(movies.title[movies.title.str.contains(name)][:1])
        return m[0]-1
    else:
        return -1

def getNameByID(IDs):
    res=movies.iloc[IDs]
    return res.title
```

Implementierung Collaborative Item-Item Filter

In [16]:
```
def CII(title, D):
    if getIDbyName(title) > 0:
        print ("recommending movies for: '" + str(getNameByID(getIDbyName(title)))+"'")
        return getNameByID(getTopN(getIDbyName(title),D))[::-1]
    else:
        print ("no movie title containing " + str(title) + "found...")
```

Implementierung Collaborative Item-Item Filter

In [16]:
```
def CII(title, D):
    if getIDbyName(title) > 0:
        print ("recommending movies for: '" + str(getNameByID(getIDbyName(title)))+"'")
        return getNameByID(getTopN(getIDbyName(title),D))[::-1]
    else:
        print ("no movie title containing " + str(title) + "found...")
```

In [17]:
```
CII("Star",D)
```

```
recommending movies for: 'Star Wars (1977)'
```

Out[17]:
```
180            Return of the Jedi (1983)
173       Raiders of the Lost Ark (1981)
171      Empire Strikes Back, The (1980)
0                      Toy Story (1995)
126               Godfather, The (1972)
Name: title, dtype: object
```

In [18]:
```
#wie oft kommt ein Film (per ID) in der DB for
id=180
print(movies.title[id])
np.sum(ratings.movie_id==id)
```

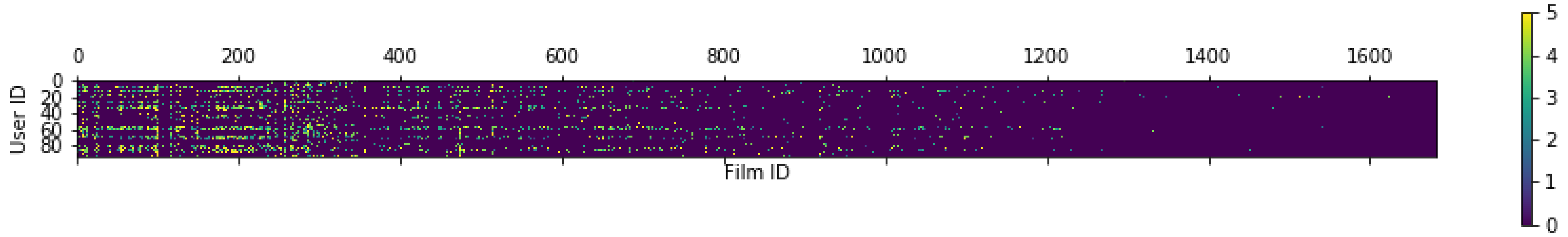
```
Return of the Jedi (1983)
```

Out[18]:
```
221
```

Quallitätsicherung - wie gut ist unser System objekitv?

In [19]:
```
#split into train and test data
from sklearn.model_selection import train_test_split
R_train, R_test = train_test_split(R, test_size=0.1)
plt.matshow(R_test)
plt.xlabel('Film ID')
plt.ylabel('User ID')
plt.colorbar()
```

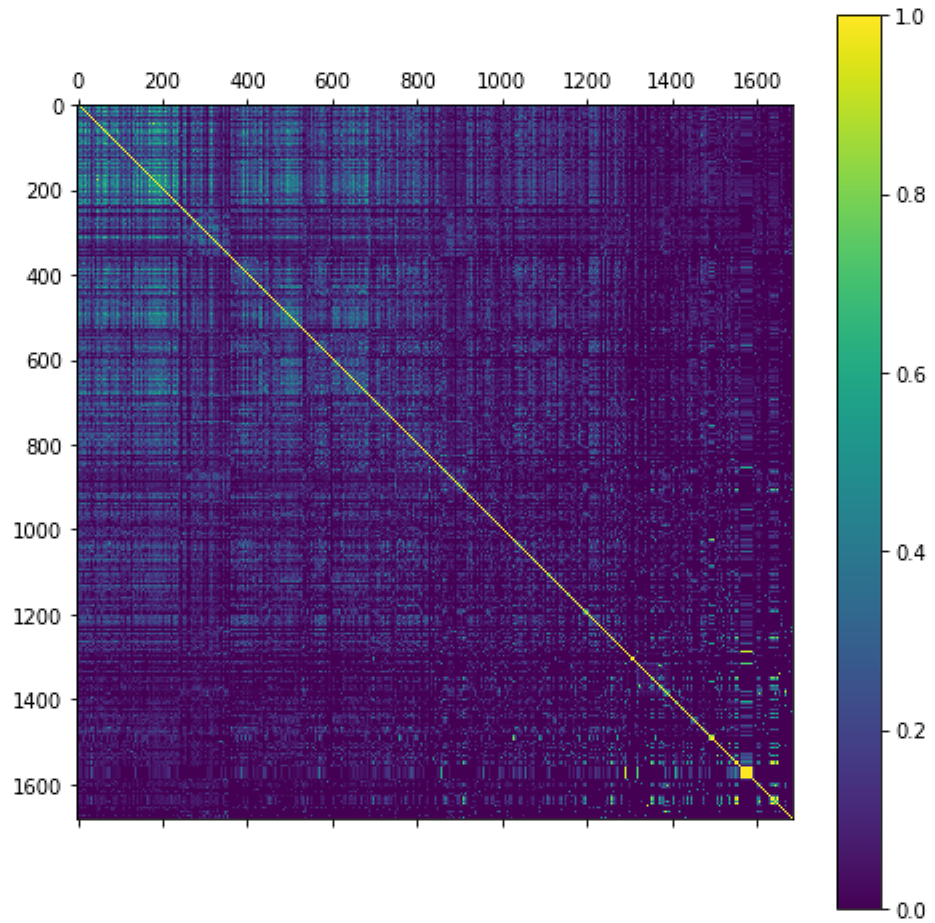
Out[19]:
```
<matplotlib.colorbar.Colorbar at 0x7f52cd68c6d0>
```



In [20]:
```
D_train=scipy.spatial.distance.squareform(scipy.spatial.distance.pdist(R_train.T, metric='cosine'))
D_train=np.abs(np.nan_to_num( D_train-1))
```

In [21]:
```
plt.rcParams['figure.figsize'] = (10.0, 8.0)
plt.matshow(D_train)
plt.colorbar()
```

Out[21]: <matplotlib.colorbar.Colorbar at 0x7f52cd623b50>



In [22]:
```
#Anfrage auf dem Trainingsdatensatz
CII("Star",D_train)
```

```
recommending movies for: 'Star Wars (1977)'
```

Out[22]:
```
49                     Star Wars (1977)
180           Return of the Jedi (1983)
173      Raiders of the Lost Ark (1981)
171     Empire Strikes Back, The (1980)
0                      Toy Story (1995)
Name: title, dtype: object
```

In [23]:
```
#get top 5 ids from random test user 23
np.argsort(R_test[23])[-5:]
```

Out[23]:
```
array([ 22, 126,  98, 356,  47])
```

In [24]:
```
def Score_byID(ID, D, Test):
    #print ("Hit Scores for: ", getNameByID(ID))
    res_id = getTopN(ID,D)[::-1]
    res_title = getNameByID(getTopN(ID,D))[::-1]
    res_score = Test[res_id]
    return res_id, res_title, res_score, np.mean(res_score)
```

In [24]:
```
def Score_byID(ID, D, Test):
    #print ("Hit Scores for: ", getNameByID(ID))
    res_id = getTopN(ID,D)[::-1]
    res_title = getNameByID(getTopN(ID,D))[::-1]
    res_score = Test[res_id]
    return res_id, res_title, res_score, np.mean(res_score)
```

In [25]:
```
Score_byID(326,D_train, R_test[23])
```

Out[25]:
```
(array([326, 301, 332, 285, 299]),
 326                Cop Land (1997)
 301       L.A. Confidential (1997)
 332               Game, The (1997)
 285     English Patient, The (1996)
 299            Air Force One (1997)
 Name: title, dtype: object,
 array([0., 0., 0., 0., 0.]),
 0.0)
```

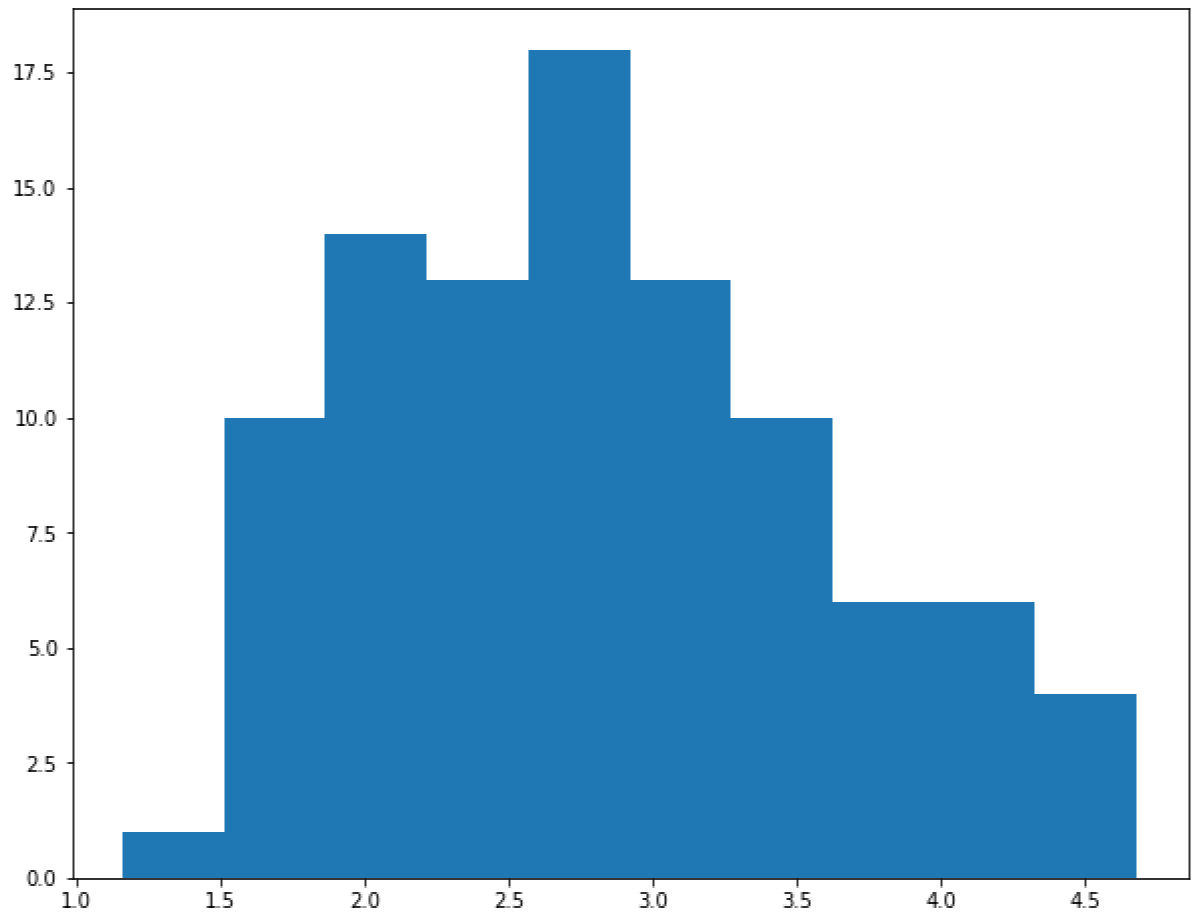
In [26]:
```python
#compute scores for all test users
def test_Score(D_train, R_test):
    userScores=[]
    for i in range(R_test.shape[0]):
        userScore=0
        userTop = np.argsort(R_test[i])[-5:]
        for e in userTop:
            res_id, res_title, res_score, av_score = Score_byID(e,D_train, R_test[i])
            userScore+=av_score
        userScores.append(userScore/(5))
    return userScores
```

In [26]:
```
#compute scores for all test users
def test_Score(D_train, R_test):
    userScores=[]
    for i in range(R_test.shape[0]):
        userScore=0
        userTop = np.argsort(R_test[i])[-5:]
        for e in userTop:
            res_id, res_title, res_score, av_score = Score_byID(e,D_train, R_test[i])
            userScore+=av_score
        userScores.append(userScore/(5))
    return userScores
```

In [27]:
```
test_res=test_Score(D_train,R_test)
```

In [28]:
```
plt.hist(test_res)
```

Out[28]:
```
(array([ 1., 10., 14., 13., 18., 13., 10.,  6.,  6.,  4.]),
 array([1.16 , 1.512, 1.864, 2.216, 2.568, 2.92 , 3.272, 3.624, 3.976,
        4.328, 4.68 ]),
 <a list of 10 Patch objects>)
```



```
In [29]: np.mean(test_res)
Out[29]: 2.8046315789473684

In [ ]:

In [ ]:
```