A Recommender System refers to a system that is capable of predicting the preference of a set of items for a user, and recommend the top items based on their previous preference.
A situation Where a user interacts with a really large catalog of items. These items could be:

- Product of Amazone
- Movies at Netflix
- News items on Google News

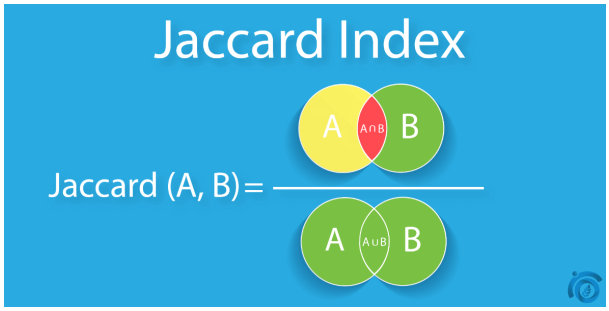
There are million of items. User is interacting with these large catalog of items. There are two ways in which user can interact:

- User know what he is looking for
- Someone recommend user

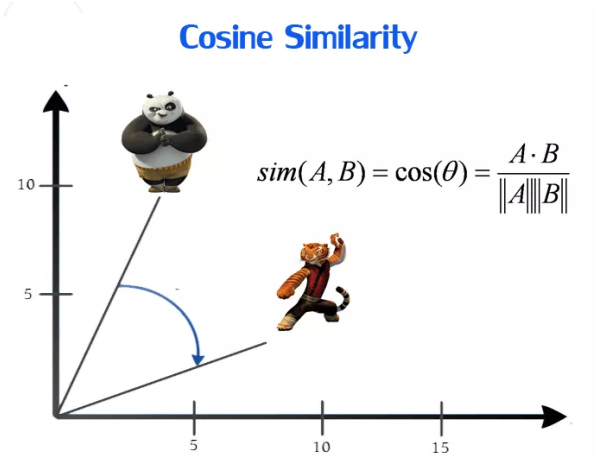
Why we are using recommendation system? we need this because we are moving from an area of scarcity to an area of abundance. There are alot of product available in market but relailer sell only who are most popular one.

Collaborative recommendation is probably the most familiar, most widely implemented one. Collaborative recommender systems recognize commonalities between users on the basis of their ratings, and generate new recommendations based on inter-user comparisons like:

- Jaccard similarity



- Cosine similarity



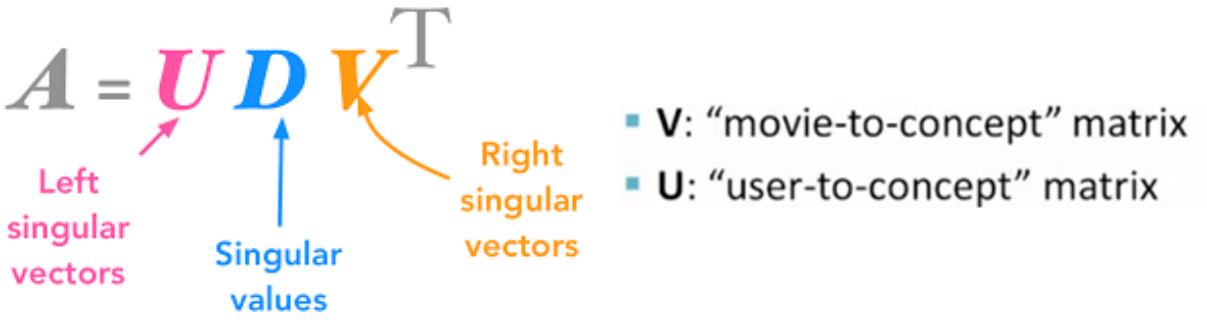
Benifits of using Collaborative System:

- No feature selection needed
- Work for any kind of item

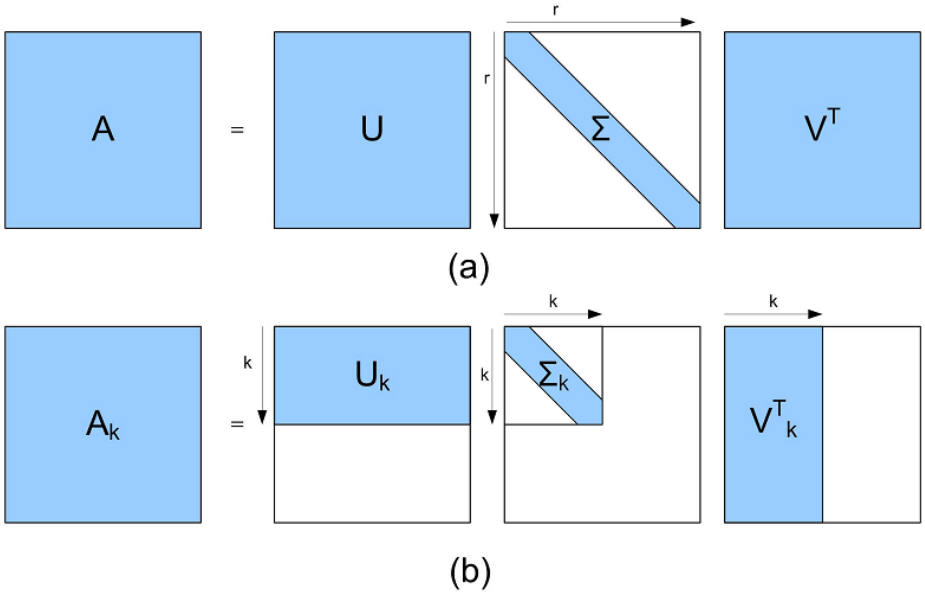
Drawback of using Collaborative System:

- Need enough users in system to find a match
- Can not recommend new items
- Popularity bias

To overcome this we use Singular Value Decomposition (SVD) algorithm SVD give 'best' axis to project which give minimum reconstruction error. it picks up linear correlations.Benefit of SVD is to get similarity between users although they have no rating in common.



D is strength of each concept
D: x1>=x2>=.....>=xk
We start reducing dimention from k to r



A ~= Ak

When we stop decomposition? Thumb rule is to keep 80-90% of energy.

ENERGY =

Similarity betweem SVD and Eigen-decomposition:

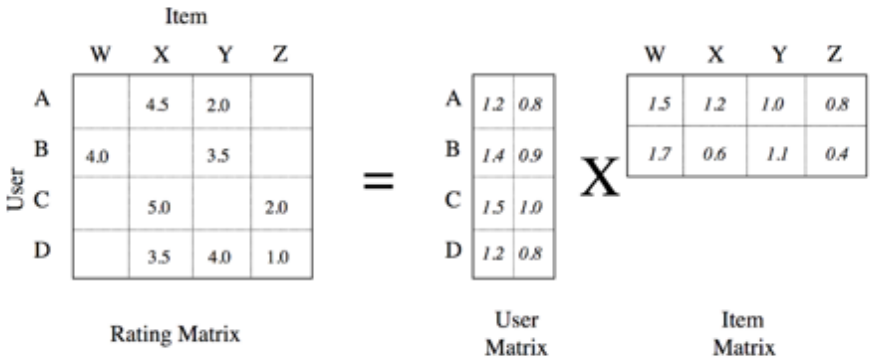
SVD: A = U *D *trans(V)
ED: R = X *P *trans(X)

- R is symmetric
- U, V, X are orthonormal(e.g., trans(U)*U = I)
- P, D are diagonal

[A *trans(A)] = [U *D *trans(V) *trans(U *D *trans(V))]
[U *D *trans(V) *trans(U *D *trans(V))] = [U *D *trans(V) *V *trans(D) *trans(U)]
[U *D *trans(V) *V *trans(D) *trans(U)] = [U *D *trans(D) *trans(U)] = [X *P *trans(X)] = R

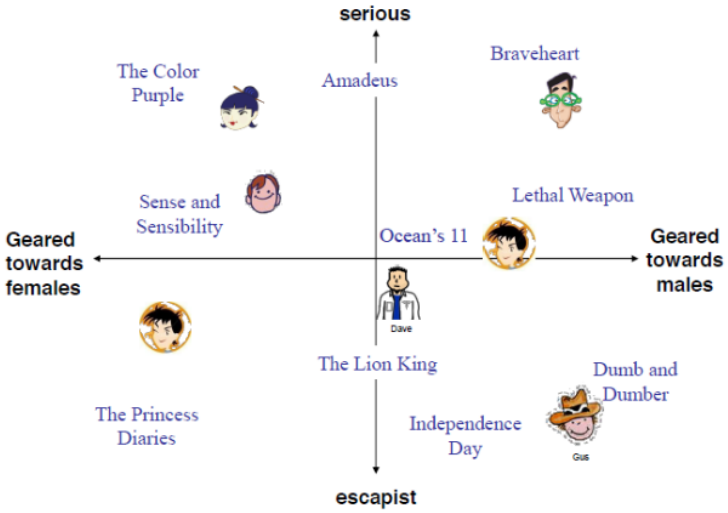
How we are using this method in our project? Here we are using **R = P * trans(Q)** Where:

- P = U, and
- trans(Q) = D * trans(V)



How we find missing values? As in this diagram value of cell(1, 1) is missing. We take first row of "user matrix" and first colum of "item matrix" and apply matrix multiplication we get value of cell(1, 1).
user -> (n, k)
movies -> (k, m)
rating -> (n, m)

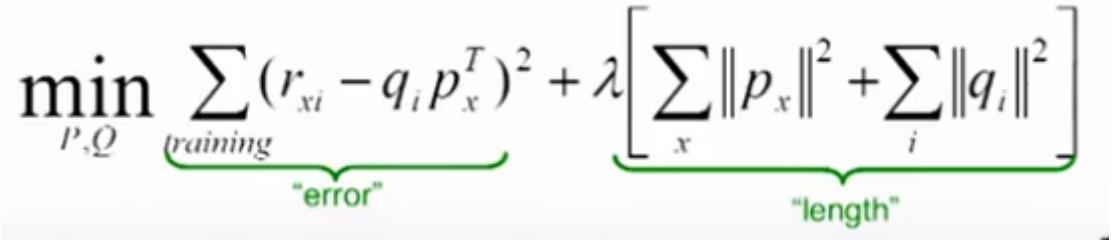
P and Q are mapped in 2Dimensions as k = 2:



When we stop decomposition as we don't using D matrix? We are not using D matrix seprately but still our goal is to make good recommendations. We check our quality of recommendations using RMSE.

Lower the RMSE -> Better the recommendations

IN some cases, our model minimize SSE on training data but give worse result on test data, it is called overfitting.
Overfitting occurs when a model tries to predict a trend in data that is too noisy. Overfitting is the result of an overly complex model with too many parameters. A model that is overfitted is inaccurate because the trend does not reflect the reality of the data.
To solve this problem we use regularization. which "handle" error and "complexity" of model.



Regularization

Handle new users/movies:

Accurate recommendations cannot be made for new users/movies with no or very little information. This is referred to as the cold start problem. This is a issue for SVD also. Some heuristics can be used. For a new user, the most popular items in the user's area could be recommended. For a new item, some rule-based similarity criteria can be defined. For example, Airbnb used the average of 3 geographically closest listings of the same type and price range to approximate a new listing(or) user like most of the movies in the item's area could be good for recommended.

In [1]:

```
from scipy import sparse
from scipy.sparse import coo_matrix

import pandas as pd
import numpy as np
import time
```

In [2]:

```
np.random.seed(42)
dims = 15         # Dimensions of the data, 15 # k as (u_num * k, k * i_num)
```

In [3]:

```
def readFile(filname):
    col_names = ["user", "item", "rate", "timeStamp"]
    df = pd.read_csv(filname)
    df.columns = col_names
    return df
```

In [4]:

```
def getData(address):
    df = readFile(address)
    rows = len(df)
    df = df.iloc[np.random.permutation(rows)].reset_index(drop=True)

    u = df["user"].max()
    i = df["item"].max()
    df = df.sub([1, 1, 0, 0], axis='columns')

    # Separate data into train and test, 85% for train and 15% for test
    split_index = int(rows*0.85)
    df_train = df[0:split_index]
    df_test = df[split_index:].reset_index(drop=True)

    R_train = coo_matrix((df_train['rate'].astype(float),
                                (df_train['user'], df_train['item'])), shape = (u, i))
    R_test = coo_matrix((df_test['rate'].astype(float),
                                (df_test['user'], df_test['item'])), shape = (u, i))
    return R_train, R_test, u, i
```

In [5]:

```
address = "C:\\Users\\admin\\Desktop\\MINE\\project\\ml-latest-small\\ml-latest-small\\
ratings.csv"
R_train, R_test, u_num, i_num = getData(address)
```

In [6]:

```
U = np.random.rand(u_num, dims)
P = np.random.rand(dims, i_num)
```

In [7]:

```
from numpy.linalg import norm

def error(R, U, P, lamda=0.02):
    ratings = R.data
    rows = R.row
    cols = R.col
    ans = 0
    for ui in range(len(ratings)):
        rui=ratings[ui]
        u = rows[ui]
        i = cols[ui]
        if rui > 0:
            ans = ans + pow(rui-np.dot(U[u,:],P[:,i]),2) + lamda*(pow(norm(U[u,:]),2) +
pow(norm(P[:,i]),2))
    return ans

def RMSError(R, U, P, lamda=0.02):
    return np.sqrt(error(R,U,P,lamda)/len(R.data))
```

In [8]:

```
def LF(R, lamda=0.02,steps=10, gamma=0.001): # Latent factor
    (u_num, i_num) = R.shape
    U = np.random.rand( u_num, dims)
    P = np.random.rand( dims, i_num)
    rmse = RMSError(R, U, P, lamda)
    print("Initial RMSE: " + str(rmse))

    for step in range(steps):
        for ui in range(len(R.data)):
            rui=R.data[ui]
            u = R.row[ui]
            i = R.col[ui]
            if rui > 0:
                eui = (rui - np.dot(U[u, :],P[:, i])) # Error user item
                U[u, :] += ( gamma*2*((eui * P[:, i]) - (lamda * U[u, :])))
                P[:, i] += ( gamma*2*((eui * U[u, :]) - (lamda * P[:, i])))
        if (step % 10)  == 0:
            rmse = RMSError(R, U, P, lamda)
            print("RMSE: " + str((step / 10)) + "    " + str(rmse))

    rmse = RMSError(R, U, P, lamda)
    print("Final RMSE: " + str(rmse))
    return U,P
```

In [9]:

```
def addUser(R):
    avg = np.array([np.mean(R, axis=0)])
    return np.append(R, avg, axis=0)

def addMovie(R):
    avg = np.array([np.mean(R, axis=1)]).transpose()
    return np.append(R, avg, axis=1)
```

In [10]:

```
def topX(R, x):
    # arr = R.toarray()
    (u_num, _) = R.shape
    mstLiked = np.zeros((u_num, x)) # Most liked movies for each user

    for ui in range(u_num):
        a = np.argsort(R[ui])
        for poss in range(x):
            y = i_num - x - 1 + poss
            mstLiked[ui][poss] = a[y]
    return mstLiked
```

In [11]:

```
(U, P) = LF(R_train, gamma=0.0003, lamda=0.001, steps=100)
```

```
Initial RMSE: 1.3975495029497853
RMSE: 0.0   1.155640345635335
RMSE: 1.0   0.9527531734875853
RMSE: 2.0   0.9030341234785223
RMSE: 3.0   0.8751667470278286
RMSE: 4.0   0.8560277216610237
RMSE: 5.0   0.8413837133010471
RMSE: 6.0   0.8293707981616322
RMSE: 7.0   0.8190152270743185
RMSE: 8.0   0.8097477667029721
RMSE: 9.0   0.8012083652525946
Final RMSE: 0.7939448733064716
```

In [12]:

```
print("RMSE test: " + "    " + str(RMSError(R_test, U, P, 0.001)))
```

```
RMSE test:    0.9223447150348119
```

In [13]:

```
from scipy import sparse
# RMatrix = csr_matrix(np.dot(U, P)) # Recommend matrix
```

In [14]:

```
RMatrix = np.dot(U, P) # Recommend matrix
```

In [15]:

```
RMatrix = addUser(RMatrix)
RMatrix = addMovie(RMatrix)
```

In [16]:

```
RMatrix.shape
```

Out[16]:

```
(611, 193610)
```

In [17]:

```
mlby = topX(RMatrix, 5) # Most liked by user
```