Example ripped from here: https://www.igvita.com/2007/01/15/svd-recommendation-system-in-ruby/ (https://www.igvita.com/2007/01/15/svd-recommendation-system-in-ruby/)

In [3]:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.linalg as sla
%matplotlib inline
```

Building a Recommendation System

Recommendations systems are a **huge** industry. A few years ago Netflix announced a one million dollar competition for an improved ranking system; one that beats their own algorithm by more than 10 percent.

One of the most successful (and simplest!) is the SVD. This is also known as latent semantic indexing, dimension reduction, projection etc.

SVD

So what is the SVD?
https://en.wikipedia.org/wiki/Singular_value_decomposition#Statement_of_the_theorem (https://en.wikipedia.org/wiki/Singular_value_decomposition#Statement_of_the_theorem)

Suppose A is a $m \times n$ matrix whose entries are real numbers or complex numbers. Then there exists a factorization, called a singular value decomposition of A, of the form

$$A = U\Sigma V^*$$

where

- U is a $m \times m$, unitary matrix,
- Σ is a $m \times n$ diagonal matrix with non-negative real numbers on the diagonal
- V^* is a $n \times n$, unitary matrix

The diagonal entries, σ_i, of Σ are known as the singular values of A. A common convention is to list the singular values in descending order. In this case, the diagonal matrix, Σ, is uniquely determined by A (though not the matrices U and V).

Problem Setup

Let's define our matrix A.

A will be a matrix where each column represents a *user* in our ranking system and each row represents the resource (e.g. movies or shows or books or whatever). FOr this example we'll have M users and N seasons of The Family Guy.

We will decompose A into U, Σ, and V in order to *approximate* A.

Let's set up 4 users and their ranking:

	Ryne	**Erin**	**Nathan**	**Pete**
season 1	5	5	0	5
season 2	5	0	3	4
season 3	3	4	0	3
season 4	0	0	5	3
season 5	5	4	4	5
season 6	5	4	5	5

In [4]:

```
seasons = [1,2,3,4,5,6]
users = ['Ryne', 'Erin', 'Nathan', 'Pete']

A = np.array([
    [5,5,0,5], # season 1
    [5,0,3,4], # season 2
    [3,4,0,3], # season 3
    [0,0,5,3], # season 4
    [5,4,4,5], # season 5
    [5,4,5,5]  # season 6
    ], dtype=float)
```

The SVD is going to be an important tool here. One fundamental concept is the idea of *compression*, or the ability to represent large pieces of data with small amounts of information. This will also allow us to identify the important features and recommend something to a new user.

Using an SVD here will compress the matrix into a smaller dimensional space where we can observe correlated items appearing in clusters. The small dimensions allow us to "see" the large amounts of information as single "features" in the data.

Create the SVD

In [7]:

```
U, S, V = sla.svd(A)
print(U.shape)
print(S.shape)
print(V.shape)
print(S)
```

```
(6, 6)
(4,)
(4, 4)
[ 17.71392084   6.39167145   3.09796097   1.32897797]
```

selecting a 2-dimensional view of the data

Here we have 4 users and 6 seasons. Let's try to project this to a 2-dimentional space. Notice that the first two singular values are the largest, so let's use this as our "important" subspace:

In [8]:

```
U2 = U[:,:2]
V2 = V.T[:,:2]
S2 = np.diag(S[:2])
print(U2.round(2))
print(S2.round(2))
print(V2.round(2))
```

```
[[-0.45 -0.54]
 [-0.36  0.25]
 [-0.29 -0.4 ]
 [-0.21  0.67]
 [-0.51  0.06]
 [-0.53  0.19]]
[[ 17.71   0.  ]
 [  0.     6.39]]
[[-0.57 -0.22]
 [-0.43 -0.52]
 [-0.38  0.82]
 [-0.59  0.05]]
```

Plot the projection

Now let's plot this. If the data was only 2-D, then the first to columns of U tell us the whole story about the seasons. Likewise V will tell us about the users.

Let's use the x-coordinate as the first column and the y-coordinate as the second column.

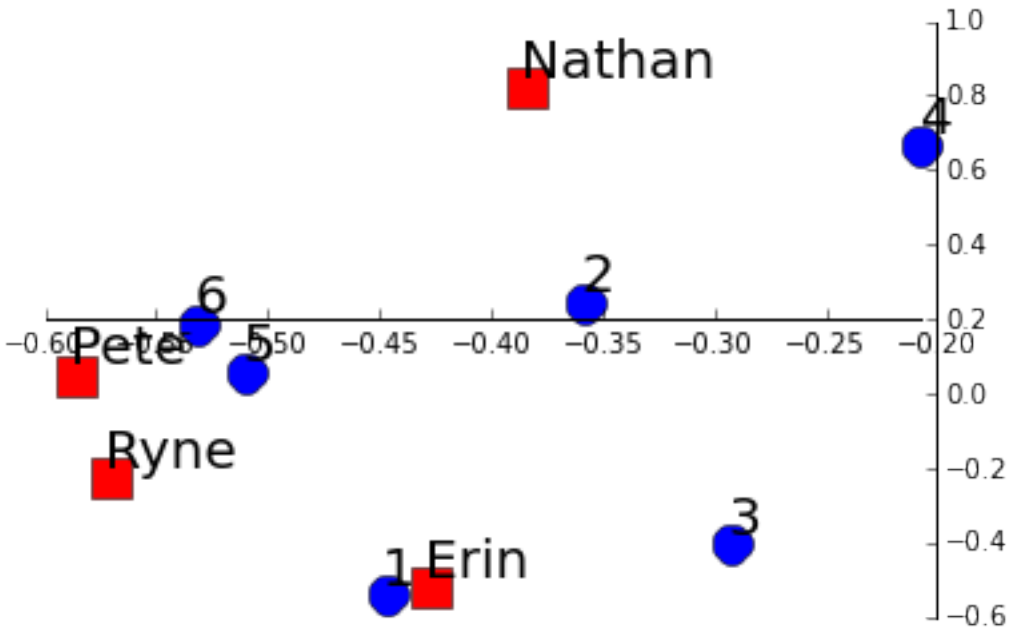
In [9]:

```
plt.plot(U2[:,0], U2[:,1], 'bo', markersize=15, clip_on=False, label='seasons')
plt.plot(V2[:,0], V2[:,1], 'rs', markersize=15, clip_on=False, label='users')

ax = plt.gca()
for i, txt in enumerate(seasons):
    ax.text(U2[i,0], U2[i,1], txt, ha='left', va='bottom', fontsize=20)

for i, txt in enumerate(users):
    ax.text(V2[i,0], V2[i,1], txt, ha='left', va='bottom', fontsize=20)

# axis trickery
ax = plt.gca()
ax.spines['left'].set_color('none')
ax.spines['bottom'].set_position('center')
ax.spines['top'].set_color('none')
ax.spines['left'].set_smart_bounds(True)
ax.spines['bottom'].set_smart_bounds(True)
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('right')
```



In [ ]:

```
```

Notice Pete and Ryne form a *User Cluster* when looking at this compressed data. Likewise, Seasons 5 and 6 form a *Season Cluster*, which is clear from the original data.

Dimension reduction is able to capture the proximity of users and seasons.

Find a similar user

Ok, now let's add a new user, Luke.

Luke rates the seasons at 5,5,0,0,0,5. What recommendation can we give? We want to find users that are similar to Luke based on Luke's ranking. So we try to *embed* Luke's data into the same 2-D space.

Let's set up the data

In [10]:

```
luke = np.array([5,5,0,0,0,5])
print(luke)
```

```
[5 5 0 0 0 5]
```

Now let's let this be L.

To *project* this onto the 2D space (a lot more on this later):

$$L^T * U_2 * S_2^{-1}$$

or...

In [13]:

```
luke2d = luke.dot(U2.dot(np.linalg.inv(S2)))
print(luke2d)
```

```
[-0.37752201 -0.08020351]
```

Plot the user projected onto the 2D data

Let's plot Luke's projected rating:

In [14]:

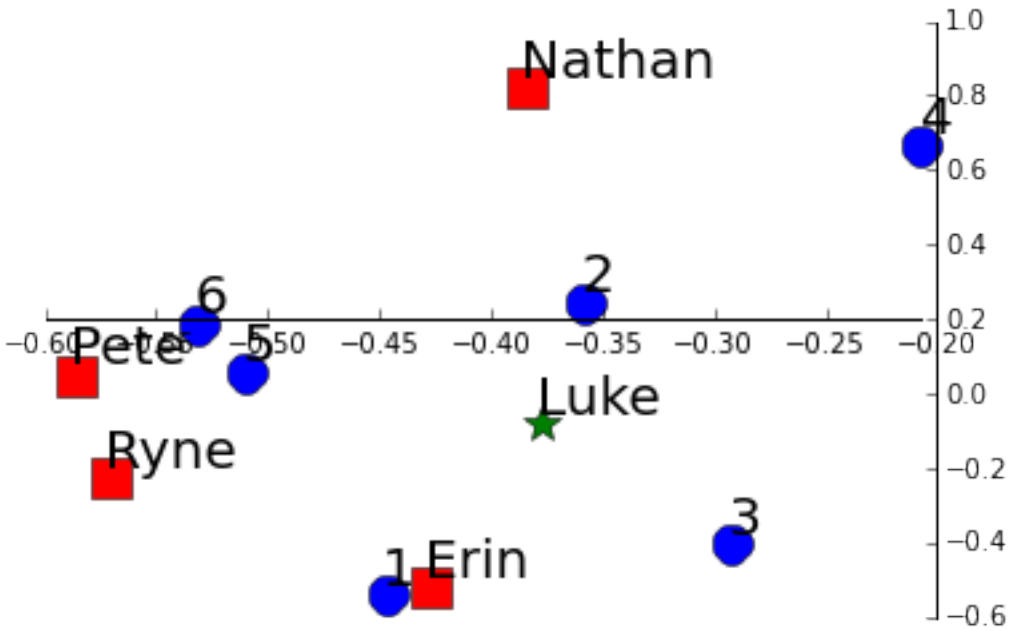
```
plt.plot(U2[:,0], U2[:,1], 'bo', markersize=15, clip_on=False, label='seasons')
plt.plot(V2[:,0], V2[:,1], 'rs', markersize=15, clip_on=False, label='users')

ax = plt.gca()
for i, txt in enumerate(seasons):
    ax.text(U2[i,0], U2[i,1], txt, ha='left', va='bottom', fontsize=20)

for i, txt in enumerate(users):
    ax.text(V2[i,0], V2[i,1], txt, ha='left', va='bottom', fontsize=20)

plt.plot(luke2d[0], luke2d[1], 'g*', markersize=15, clip_on=False, label='luke')
ax.text(luke2d[0], luke2d[1], 'Luke', ha='left', va='bottom', fontsize=20)


# axis trickery
ax = plt.gca()
ax.spines['left'].set_color('none')
ax.spines['bottom'].set_position('center')
ax.spines['top'].set_color('none')
ax.spines['left'].set_smart_bounds(True)
ax.spines['bottom'].set_smart_bounds(True)
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('right')
```



Notice that the angle from Luke to Pete and Ryne is the shortest.

How close is Luke's rating?

To quantify this we'll use something call cosine similarity:

$$similarity = \frac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{v}\|\|\mathbf{w}\|}$$

In [136]:

```
for i, xy in enumerate(V2):
    angle = np.dot(xy, l) / (np.linalg.norm(xy) * np.linalg.norm(l))
    print('Angle between %s and %s: %2.2g' % ('luke', users[i], angle))
```

```
Angle between luke and Ryne: 0.99
Angle between luke and Erin: 0.78
Angle between luke and Nathan: 0.23
Angle between luke and Pete: 0.96
```