Lab 9: PCA for Movie Recommendations

A common application of PCA is for recommendation systems. In this lab, we will use PCA to create a very primitive recommendation system for movies. Through the lab, you will learn to:

- Represent ratings data as a sparse matrix
- Perform PCA on the rating matrix to find reccomendations
- Interpret PCA loadings of rating data

Loading the MovieLens Dataset

We first load some common packages.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

GroupLens (https://grouplens.org/) is a research organization at the University of Minnesota that has done extensive work in recommendation systems among other topics. They have excellent datasets on movie recommendations as part of their MovieLens project (https://movielens.org/). In this lab, we will use a very small dataset that is useful for illustrating basic ideas. But, if you are interested in continuing research in this area, they have much larger datasets.

To get the data, go to the webpage:

https://grouplens.org/datasets/movielens/latest/ (https://grouplens.org/datasets/movielens/latest/)

and download and unzip the files, `ml-data-small.zip`.

Once, the data is downloaded, use the `pd.read_csv` command to load the `movies.csv` file and store the results in a pandas dataframe `movies`. The `movies` dataframe will have the title and genres of the movies that are to be rated. Use the `head` method to print the first 5 rows of the `movies` dataframe.

In [2]:
```
# TODO:  Read the movies
# movies = pd.read_csv(...)
movies = pd.read_csv('ml-latest-small/movies.csv')
movies.head()
```

Out[2]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Extract the following columns from the `movies` dataframe:

- Extract the `movieId` column, convert to an `np.array` and store in `movie_ids`
- Extract the `title` column, convert to a list (using `.tolist()`) and store in `titles`

In [3]:
```
# TODO:
# movie_id = ...
# titles = ...
movie_ids = np.array(movies['movieId'],dtype=int)
titles = movies['title'].tolist()
```

The following function returns the string of a movie title, given its movie id.

In [4]:
```
def get_movie_title(movie_id):
    I = np.where(movie_ids == movie_id)[0]
    if len(I) == 0:
        return 'unknown'
    else:
        return titles[I[0]]
```

Load the `ratings.csv` file into a `pandas` dataframe `ratings`. Use the `head` method to print the first five rows of the dataframe.

In [5]:
```
# TODO
# ratings = ...
ratings = pd.read_csv('ml-latest-small/ratings.csv')
ratings.head()
```

Out[5]:

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

Extract three columns from the `ratings` dataframe: `user_ids`, `user_movies` and `user_ratings` with the user id, movie id and rating score Convert to each to an `np.array`.

Create a Ratings Matrix

We now create a ratings matrix from the ratings using the `pivot_table` commmand as follows.

In [6]:
```
M = ratings.pivot_table(index=['userId'], columns=['movieId'], values='rating'
)
movie_col = M.columns.tolist()
user_row = M.index.tolist()
```

Display the data frame using the `M.head()` command.

In [7]:
```
M.head()
```

Out[7]:

movieId	1	2	3	4	5	6	7	8	9	10	...	161084	161155	161
userId														
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	...	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	...	NaN	NaN	NaN
5	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN

5 rows × 9066 columns

You shoul see that most of the entries are `NaN` since most of the movies were not rated. A key challenge in recommendation systems is to fill these in.

For this lab, use the `fillna` command to fill in all the `NaN` entries with zeros. Store the filled in dataframe in `Mfill`. Print the first few rows of the new dataframe.

Filling in with zeros is not the best idea, but it is simple and will be OK for this lab. But, real recommendation do something more sophisticated called *matrix completion*.

In [8]:
```
# TODO
# Mfill = ...
#Mfill = M.fillna(M.mean())
Mfill = M.fillna(0)
Mfill.head()
```

Out[8]:

movieId	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**...**	**161084**	**161155**	**161594**	**161830**	**1**
userId																
1	0.0	0.0	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	0.0	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	(
3	0.0	0.0	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	0.0	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	(
5	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	(

5 rows × 9066 columns

Convert `Mfill` to an `np.array`.

In [9]:
```
# TODO
Mfill = np.array(Mfill)
```

Using the shape of `Mfill`, find the number of users and movies and print the results.

In [10]:
```
# TODO
nusers, nmovies = Mfill.shape
print('Number of users =  %d' % nusers)
print('Number of movies = %d' % nmovies)
```

```
Number of users =  671
Number of movies = 9066
```

Take a PCA of the Ratings Matrix

We now take a PCA of the traings matrix. First, create a matrix `X` formed by standardizing the matrix `Mfill`. That is, subtract the mean and divide by the standard deviation of each column of `Mfill`.

In [12]:
```
# TODO:  Standardize Mfill
# Mmean = ...
# Mstd = ...
# X = ...
Mmean = np.mean(Mfill,axis=0)
Mstd = np.std(Mfill,axis=0)
X = (Mfill-Mmean[None,:])/Mstd[None,:]
```

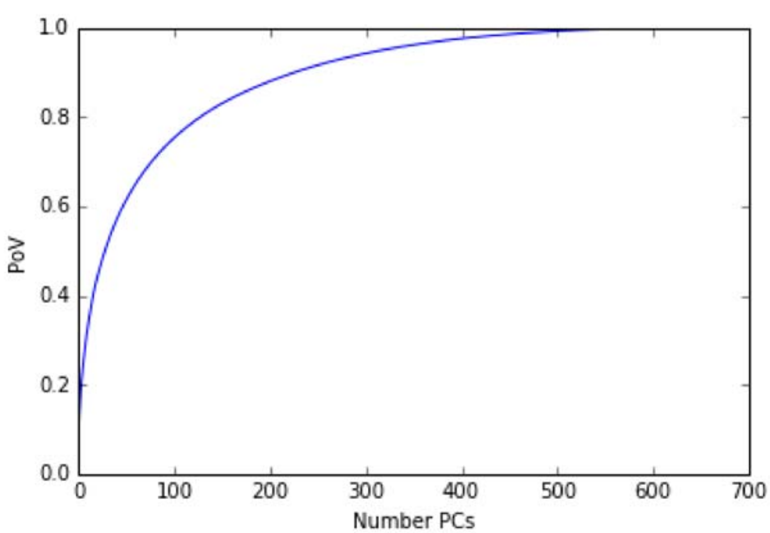
Now, take an SVD of X to perform the PCA. Use the `np.linalg.svd` method with `full_matrices=False`. Due to the size of the matrix, this make take a minute or so.

In [13]:
```
# TODO
# U,S,Vt = ...
U,S,Vt = np.linalg.svd(X, full_matrices=False)
```

Plot the portion of variance as a function of the number of PCs. In this example, you will see that the data is not that low rank. This arises since we have filled in many entries with their mean values.

In [14]:
```
# TODO Plot the PoV
pov = np.cumsum(S**2)/np.sum(S**2)
plt.plot(pov)
plt.xlabel('Number PCs')
plt.ylabel('PoV')
```

Out[14]: `<matplotlib.text.Text at 0x1764ac48438>`



Making a Recommendation

We can now use our PCA to make recommendations. First, create a matrix `Xest` by taking a rank `r=50` approximation of the original matrix `X`.

In [15]:
```
# TODO
# Xest = ...
r = 50
Xest = (U[:,:r]*S[None,:r]).dot(Vt[:r,:])
```

Now, using the mean and standard deviation from the above, compute `Mest`, the corresponding low-rank approximation of the `Mred`.

In [16]:
```
Mest = Xest*Mstd[None,:] + Mmean[None,:]
```

Now, take some row of the estimated rating matrix, say the row with index, `ind=10`. The predicted ratings for that user will be in `Mest[ind,:]`. Find the 20 indices `j` where `Mest[ind,j]` is the largest. For each `j`, print:

- movie title
- the predicted rating `Mest[ind,j]`
- the actual rating `Mfill[ind,j]`

Note that you must use `movie_col` and `get_movie_title()` to find the movie title.

You will notice that the predicted rating is very low. This is because we filled in the unknown entries with zeros. But, you should see that the values of `Mest` that are large correspond to movies that the user rated well (4 or 5).

In [17]:
```
ind = 10   # Row index
ntop = 20  # Print the ntop movie recommendations

# TODO
I = np.argsort(Mest[ind,:])[::-1]
for i in range(ntop):
    j = I[i]
    movie_id = movie_col[j]
    title = get_movie_title(movie_id)[:40]
    print('%40s %f %f' % (title, Mest[ind,j], Mfill[ind,j]))
```

```
        Shawshank Redemption, The (1994) 1.603652 0.000000
                     Pulp Fiction (1994) 1.521500 5.000000
                     Forrest Gump (1994) 1.518066 0.000000
        Silence of the Lambs, The (1991) 1.297956 0.000000
Star Wars: Episode IV - A New Hope (1977 1.253993 0.000000
                      Matrix, The (1999) 1.135934 0.000000
                 Schindler's List (1993) 1.014050 0.000000
                       Fight Club (1999) 0.959955 0.000000
                    Jurassic Park (1993) 0.955836 0.000000
                 Dark Knight, The (2008) 0.937898 0.000000
                        Toy Story (1995) 0.937703 0.000000
                            Fargo (1996) 0.914210 0.000000
                   Godfather, The (1972) 0.890318 0.000000
Star Wars: Episode V - The Empire Strike 0.870297 0.000000
       Terminator 2: Judgment Day (1991) 0.865886 0.000000
                        Inception (2010) 0.838583 4.000000
                  American Beauty (1999) 0.814838 0.000000
Lord of the Rings: The Fellowship of the 0.809510 0.000000
Lord of the Rings: The Return of the Kin 0.794779 0.000000
Raiders of the Lost Ark (Indiana Jones a 0.773624 0.000000
```

To evaluate if these are *good ratings*, we could split the data into training and test. Then, we would fit the PCA on the training data, and then compare the predicted ratings on the test data. But, we won't do this here.

Interpreting the PCs

It is useful to examine the principal components to see which movies figure prominently in each component. Recall that the `i-th` PC is in the vector, `Vt[i,:]`. For the top `npc=4` principal components, find the indices `j` where `Vt[i,j]` has the largest absolute value and print the corresponding movie titles.

Ideally, each PC would correspond to some aspect of the movies and hence the movies with the highest loading values in the same PC will have some common aspect. Since we did a very simple completion, we may not see such a grouping here.

In [19]:
```
npc = 4
ntop = 5
for i in range(npc):
    # Find the largest components in the i-th PC
    I = np.argsort(np.abs(Vt[i,:]))[::-1]
    print('PC %d:' % i)

    # Print the movie titles
    for j in range(ntop):
        ind = movie_col[I[j]]
        title = get_movie_title(ind)[:40]
        print('%40s' % title)
    print('')
```

```
PC 0:
                  Wasp Woman, The (1959)
              Littlest Rebel, The (1935)
                  Mondo Hollywood (1967)
 Killing of a Chinese Bookie, The (1976)
                  Take This Waltz (2011)

PC 1:
                    Shadow Dancer (2012)
                        Hitchcock (2012)
                     Last Knights (2015)
                     Blue Thunder (1983)
                    Winter's Tale (2014)

PC 2:
                         H.O.T.S. (1979)
                      Carnosaur 2 (1995)
                 Gumby: The Movie (1995)
Savage Nights (Nuits fauves, Les) (1992)
                         Reckless (1995)

PC 3:
                    Above the Rim (1994)
       Legendary Weapons of China (1982)
Dragon Ball Z: Broly - The Legendary Sup
                             Slam (1998)
Dragon Ball Z: Bardock - The Father of G
```

More Fun

Recommendation systems is a large area in machine learning. If you want to explore more, you can do the following:

- Most importantly, you will want to do something better than filling in the unrated items with zeros. One popular method is called *low-rank matrix completion*. There are several excellent packages on python for this now
- Use larger datasets in the MovieLens projects. They have sets with 1 million entries!
- To move to larger datasets, you will need to use sparse matrices for the storage.
- You can also explore `sklearn`'s `PCA` package instead of performing the PCA manually.