CSCI 416 - HW3

Name: Nicholas Wilson

Problem 3

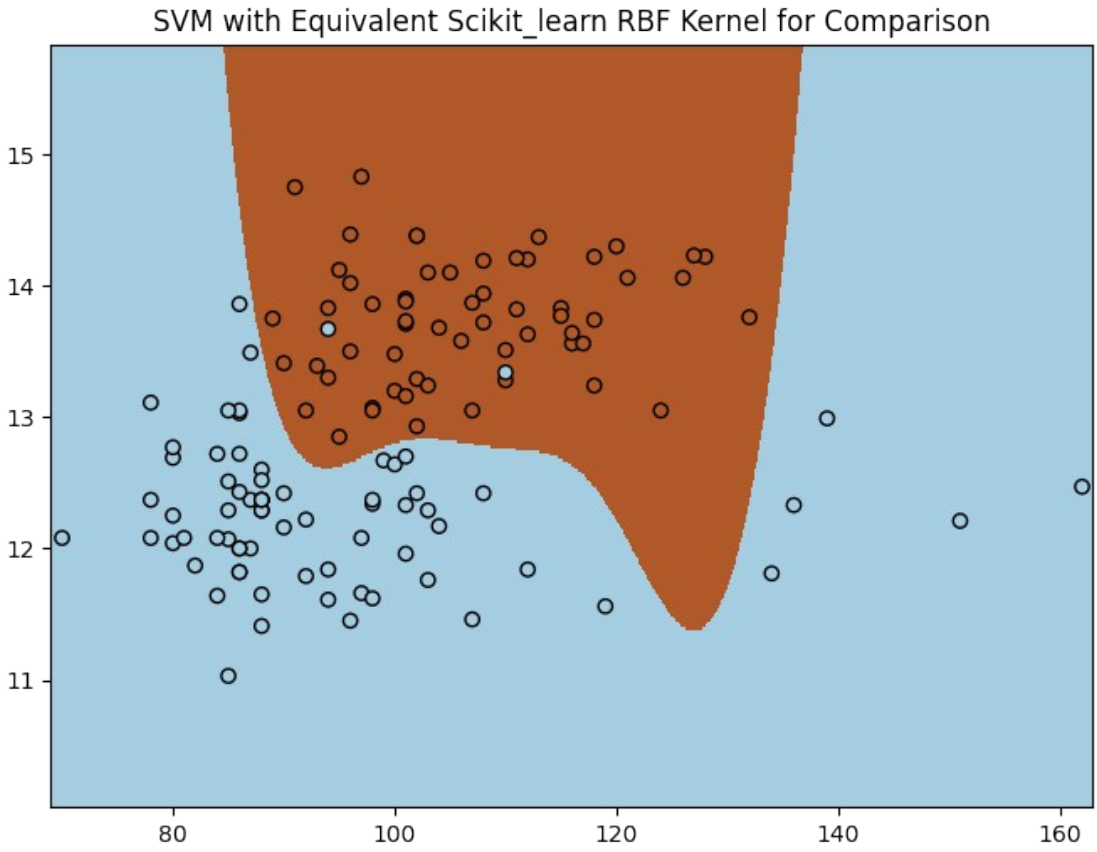
```
"""
=====================================
Test SVM with custom Gaussian kernels
=====================================


Author: Eric Eaton, 2014

Adapted from scikit_learn documentation.

"""
print(__doc__)

from numpy import loadtxt, ones, zeros, where
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from svmKernels import myGaussianKernel
from svmKernels import _gaussSigma

# import some data to play with

filename = 'data/svmTuningData.dat'
data = loadtxt(filename, delimiter=',')
X_raw = data[:, 0:-1]
Y_raw = np.squeeze(np.array([data[:, 2]]).T)
m, d = X_raw.shape
#print(m,d)

print("Training the SVMs...")

trials = 10
folds = 10
fold_size = m//folds
train_size = fold_size * (folds - 1)
#print(fold_size, train_size)
validation_size = fold_size
```

```
# Best parameters
best=[0,0,0]

# Search parameters through a grid
#TO DO: CHANGE THIS PART

sigma_vals = 10**np.arange(-3., 5.)
C_vals = np.linspace(1,100, num=100)

for c in range(0,100):
    for g in range(0,8):
      C = C_vals[c]
      _gaussSigma = sigma_vals[g]
      accuracy = 0

      for t  in range(0,trials):

        #randomize the data set
        p = np.random.permutation(m)
        order = p[0:m]
        X = X_raw[order,:]
        Y = Y_raw[order]

        for f in range(0,folds):

            #cross validation: get train set and test set
            # TO DO
            # You need to finish the following for cross validation
            #Use fold_size and validation_size

            #X_train is Equal to all but the "fth" fold of X
            X_train = np.ones((train_size, d)) #Create an array of the
correct size
            #Grab all the rows of X up to the f * fold_size row
            X_train[0:f * fold_size,:]  = X[0:f * fold_size,:]
            #Grab all the rows of X after f * fold_size +
validation_size row
            X_train[f * fold_size:m,:]  = X[f * fold_size +
validation_size:m,:]
            # print(X.shape)
            # print(X)
            # print()
            # print(X_train.shape)
            # print(X_train)
            Y_train = np.ones((train_size)) #Equal to all but the
"fth" fold of Y
            Y_train[0:f*fold_size] = Y[0:f*fold_size]
            Y_train[f*fold_size:] = Y[f*fold_size + validation_size:]
            # print(Y.shape)
```

```
            # print(Y)
            # print()
            # print(Y_train.shape)
            # print(Y_train)
            #X_test = np.ones(validation_size, d) #Equal to the "fth"
fold of X
            X_test = X[f * fold_size: f * fold_size +
validation_size,:]
            # print(X_test)
            # print(X_test.shape)
            Y_test =Y[f * fold_size: f * fold_size + validation_size]
#Equal to the "fth" fold of Y
            # print()
            # print(Y_test.shape)
            # print(Y_test)
            # create an instance of SVM with build in RBF kernel and
train it
            equivalentGamma = 1.0 / (2 * _gaussSigma ** 2)
            model = svm.SVC(C=C, kernel='rbf', gamma=equivalentGamma)
            model.fit(X_train, Y_train)

            predictions_test =
model.predict(np.c_[X_test[:,0],X_test[:,1]])
            a = np.mean(Y_test==predictions_test)
            accuracy += a


        # Best Accuracy So Far
        average_accuracy = accuracy/(folds*trials)
        if average_accuracy > best[2]:
            best[0] = C
            best[1] = _gaussSigma
            best[2] = average_accuracy


print(best)


====================================
Test SVM with custom Gaussian kernels
====================================


Author: Eric Eaton, 2014

Adapted from scikit_learn documentation.


Training the SVMs...
[82.0, 10.0, 0.9830769230769227]
```

```
# create an instance of SVM with build in RBF kernel and train it
print("The best parameters are: ", best)
C = best[0]
_gaussSigma = best[1]
equivalentGamma = 1.0 / (2 * _gaussSigma ** 2)
model = svm.SVC(C=C, kernel='rbf', gamma=equivalentGamma)
model.fit(X, Y)

h = .02  # step size in the mesh

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, m_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))


predictions = model.predict(np.c_[xx.ravel(), yy.ravel()])
predictions = predictions.reshape(xx.shape)

# plot my results
plt.figure(figsize=(8, 6), dpi=100)

plt.pcolormesh(xx, yy, predictions, cmap="Paired")
plt.scatter(X[:, 0], X[:, 1], c=Y, cmap="Paired", edgecolors="black")
# Plot the training points
plt.title('SVM with Equivalent Scikit_learn RBF Kernel for
Comparison')
plt.axis('tight')

plt.show()
```

```
The best parameters are:  [82.0, 10.0, 0.9830769230769227]
```



Report optimal values and the corresponding estimated accuracy. And explain how you find those optimal values.

My best C was 82.0 and my best sigma was 10.0. Corresponding accuracy was 0.9830769230769227, or 98.308% accurate. I found those by testing every sigma value between 0.001 and 10000 and every C value between 1 and 100. In each test, I did 10 trials. Within each trial, I randomized the data, split it into 10 folds and used 9 of those folds for training data and the remaining fold for testing data, and used that to calculate accuracy for that trial. For each C and sigma value, I then calculated the average accuracy across those 10 trials, then reported the highest average accuracy.

Problem 4

Movie Recommendations

user
Alice
Bob

Carol

David

Eve

What movie should I recommend to Bob? Will Carol like Frozen?

Goal: Fill in entries of the "rating matrix"

Problem Setup

Let's formalize this as a machine learning problem. To make it concrete, let's load some data and see what it looks like.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import display
import scipy.io

# Load train and test data
data = scipy.io.loadmat('movies.mat')

titles = [t[0] for t in data['movieData']['title'][0,0].ravel()]

for x,y in data.items():
    if isinstance(y, (np.ndarray)) and len(y)==1:
        data[x] = np.asscalar(y)
    elif isinstance(y, (np.ndarray)):
        data[x] = y.ravel()

nUsers    = data['nUsers']
nMovies   = data['nMovies']
userData  = data['userData']
movieData = data['movieData']

train_user   = data['train_user']-1   # matlab 1-index correction
train_movie  = data['train_movie']-1  # matlab 1-index correction
train_rating = data['train_rating']

valid_user   = data['valid_user']-1   # matlab 1-index correction
valid_movie  = data['valid_movie']-1  # matlab 1-index correction
valid_rating = data['valid_rating']

test_user    = data['test_user']-1    # matlab 1-index correction
test_movie   = data['test_movie']-1   # matlab 1-index correction
```

```
# Create a pandas data frame for training data to facilitate
# visualization and inspection

train_title = [titles[i] for i in train_movie]

train_data = pd.DataFrame(data = {'user_id' : train_user,
                                  'movie_id' : train_movie,
                                  'rating' : train_rating,
                                  'title': train_title},
                          columns = ['user_id', 'movie_id', 'rating',
'title'])

# subsample to 5000 rows to more easily see a small sampling of
ratings for each user
train_data = train_data[:5000]

# sort by user
train_data = train_data.sort_values(by=['user_id', 'rating'])

display(train_data)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:15:
DeprecationWarning: np.asscalar(a) is deprecated since NumPy v1.16,
use a.item() instead
  from ipykernel import kernelapp as app
```

```
      user_id  movie_id  rating
title
2070        0       242       1                          Jungle2Jungle
(1997)
2175        0        73       1           Faster Pussycat! Kill! Kill!
(1965)
984         0       101       2                        Aristocats, The
(1970)
2400        0       236       2                          Jerry Maguire
(1996)
4364        0       179       3                         Apocalypse Now
(1979)
...       ...       ...     ...
...
1373      942        61       3                                Stargate
(1994)
724       942       150       4  Willy Wonka and the Chocolate Factory
(1971)
1883      942        23       4                    Rumble in the Bronx
(1995)
3403      942       731       4                                   Dave
(1993)
```

```
1851      942        11       5                    Usual Suspects, The
(1995)

[5000 rows x 4 columns]
```

Training Data

As we can see, the training data presents observed entries of the "ratings" matrix as list of triples (i_k, j_k, r_k) where

- i_k is the user index of kth rating
- j_k is the movie index of kth rating
- r_k is the value of kth rating (1-5)

In our code we will store the entries of the tuples in three separate 1d arrays of the same length, so the kth rating is represented by the values `train_user[k]`, `train_movie[k]`, and `train_rating[k]`.

Problem Formulation

Now, let's formulate the problem mathematically. Suppose there are m users and n movies. Let R be the $m \times n$ "rating" matrix, where R_{ij} is the (possibly unknown) rating for user i on movie j.

Our training data gives us some of the entries of the rating matrix. Our goal is to learn a parametric model to predict entries that we don't observe.

But Where are the Features?

What sort of predictive model can we use for entries of R?

In past learning problems we had *feature vectors* and we learned *weight vectors* to make predictions (using dot products).

Now we do not have feature vectors. What should we do?

Matrix Factorization Model

Our solution is to **learn weight vectors for both users and movies**.

Let $\u_i \in \R^d$ be the weight vector for use i and $\v_j \in \R^d$ be the weight vector for movie j. Then we can predict the rating for user i on movie j as:

$$ H_{ij} =\u_i^T \v_j $$

Our goal is to learn weight vectors for every user and movie so that $R_{ij} \approx H_{ij}$ for those entries of the rating matrix that we observe.

Problem statement: Given observed entries of the rating matrix presented as triples (i_k, j_k, r_k) for $k=1,\ldots,n_{\text{train}}$, find weight vectors u_i for each user i and v_j for each movie j such that:

$$r_k \approx u_{i_k}^T v_{j_k}, k=1,2,\ldots,n_{\text{train}}$$

Why is This Called Matrix Factorization?

- Place the user weight vectors $\u_i$ into the rows of a matrix U and the movie feature vectors $\v_j$ into the rows of a matrix V

$$ \newcommand{\line}{-} U = \begin{bmatrix} \line \u_1^T \line \\ \line \u_2^T \line \\ \ldots \\ \line \u_m^T \line \\ \end{bmatrix} \in \R^{m \times d} \qquad V = \begin{bmatrix} \line \v_1^T \line \\ \line \v_2^T \line \\ \ldots \\ \line \v_n^T \line \\ \end{bmatrix} \in \R^{n \times d} $$

- Consider the product $U V^T$:

$$U$$
¿

- It is easy to check that (i, j) entry of $U V^T$ is equal to $\u_i^T \v_j$, which is our prediction for the (i, j) entry of R
- In other words, our model is that $R \approx U V^T$ (a **factorization** of R)
- We choose U and V to get good predictions for those entries of R that we can observe. As long as we don't overfit, this gives us power to generalize to entries we don't observe
- The "hidden dimension" d (the length of each weight vector) is a hyperparameter that must be tuned with hold-out data.

Your Job: Solve the Learning Problem

- Formulate a squared error cost function corresponding to the problem statement above.
- Add regularization for *every* user weight vector $\u_i$ and movie weight vector $\v_j$ to get a regularized cost function
- Write down the partial derivatives of your regularized cost function with respect to the entries of $\u_i$ and $\v_j$
- Plug the partial derivatives into stochastic gradient descent (SGD) and write down the update rule
- Implement SGD
- Tune parameters (e.g., dimension d, regularization parameter) get good performance on the validation set

Logistics

- Submit predictions on test set

- Evaluation: root-mean squared error (RMSE) on test set

$$\text{RMSE} = \sqrt{\frac{1}{n_{\text{test}}} \sum_{(i,j) \in \text{ test set}} \left(H_{ij} - R_{ij}\right)^2}$$

- Your grade:

RMSE
<= 1.0
<= 0.97
<= 0.95
<= 0.94

(Review on your own) Model Extension: Add Biases

To get really great performance, consider this extended model for a predicted rating:

$$ H_{ij} = \mu + a_i + b_j + \u_i^T \v_j $$

This adds several terms to the prediction for user i on movie j:

- μ is an overall baseline rating. For example, the overall average rating of all users on all movies may be $\mu = 3.3$
- a_i is a user-specific adjustment or "bias". For example, perhaps Alice really loves movies and gives them all high ratings. Then, her bias might be $a_i = +0.4$. But Bob is hard to please, so his bias is $a_i = -0.7$.
- b_j is a movie-specific bias. For example, perhaps Inside Out is universally loved, so its bias is $b_j = +0.7$. A really bad movie would have a negative bias.

The set of parameters of this model includes:

- μ
- $a_i, i = 1, \ldots, m$
- $b_j, j = 1, \ldots, n$
- $\u_i \in \R^d$, $i = 1, \ldots, m$
- $\v_j \in \R^d$, $j = 1, \ldots, n$

To learn these parameters, derive partial derivatives of the regularized cost function with respect to *all* of the above parameters, and update them all within your stochastic gradient descent loop.

Further Reading

Matrix Factorization Techniques for Recommender Systems by Yehuda Koren, Robert Bell and Chris Volinsky

- Authors were on the winning team of Netflix prize
- Paper includes algorithms---but beware different notation

Step 0: Familiarize Yourself With Variables

Here are the variables we populated while loading the data above --- make sure you run that cell first.

```
# 1) Metadata
#
#     nUsers     # of users
#     nMovies    # of movies
#     titles     list of movie titles
#
#
# 2) Training data (60K ratings). This consists of three 1d arrays,
#    each of length 60K:
#
#      train_user, train_movie, train_rating
#
#    The entries specify the ratings:
#
#      train_user[k]    user index  of kth rating
#      train_movie[k]   movie index of kth rating
#      train_rating[k]  value (1-5) of kth rating
#
# 2) Validation data (20K ratings). Three vectors of length 20K:
#
#      valid_user, valid_movie, valid_rating
#
#    Use this to evaluate your model and tune parameters.
#
# 3) Test set (20K user-movie pairs without ratings):
#
#      test_user, test_movie
#
#    You will create predictions for these pairs and submit them for
#    grading.
```

Step 1: Look at the Prediction Method

To make things concrete, first take a look at the prediction method below. This is just a stub for now that returns the same value `mu` for every prediction. Later you will update this to make predictions given the weight vectors and biases.

```
from os import X_OK
def rmse(h, r):
    resid = h - r
    cost = np.sqrt(np.mean(resid**2))
    return cost
```

```
def predict(mu, uBiases, mBiases, uWeights, mWeights, user, movie):
    '''
    PREDICT Make predictions for user/movie pairs
    Inputs:
      model parameters
      mu                average user rating for all movies
      uBiases           vector of biases for each user
      mBiases           vector of biases for each movie
      uWeights          matrix of user weights (eg U)
      mWeights          matrix of movie weights (eg V)
      user              vector of users (eg train_user)
      movie             vector of movies (eg train_movie)

    Output:
      predictions       vector of predictions
    '''

    # This is a stub that predicts the mean rating for all user-movie
pairs
    # Replace with your code.

    L = len(user)
    predictions = np.zeros(L)
    for k in range(L):
      i = user[k]
      x = movie[k]
      predictions[k] = mu + uBiases[i] + mBiases[x] +
np.dot(uWeights[i].T, mWeights[x])
    #predictions[:] = mu

    return predictions
```

Step 2: Learning and Validation

Write code here to do the learning and validation. Stubs are provided. Make sure you derive the partial derivatives on paper before you try to code them.

```
##############################################
# Tunable parameters (you will add more)
##############################################

nDims = 30 #100

#Regularization constants
lambda_U = 5e-3 #Regularization constant for user vectors, 4
lambda_V = 5e-3 #Regularization constant for movie vectors, 15
lambda_a = 1e-4 #Regularization constant for user biases, 0.05
lambda_b = 1e-4 #Regularization constant for movie biases, 0.05
#lambdas = [lambda_U,lambda_V,lambda_a,lambda_b]
```

```
#Alpha/learning rates
alpha_U = 2.5e-3 #User vectors, 1e-2
alpha_V = 2.5e-3 #Movie vectors, 1e-2
alpha_a = 2.5e-3 #User biases, 1e-2
alpha_b = 2.5e-3 #Movie biases, 1e-2
#alphas = [alpha_U,alpha_V,alpha_a,alpha_b]

#Number of iterations
iterations = 1800000

##############################################
# Initialize parameters
##############################################

mu = np.mean(train_rating)
a  = np.zeros(nUsers)
b  = np.zeros(nMovies)
U  = np.random.randn(nUsers, nDims)  *.01 # User weights
V  = np.random.randn(nMovies, nDims) *.01 # Movie features

##############################################
# Training and validation
##############################################

# TODO: write code to train model and evaluate performance on
validation set
#
#  predict() is a stub that predicts the overall mean for all user-
movie
#  pairs. Update it to take more parameters and make real predictions.

#Write the SGD loop HERE
#Use it to modify a, b, U, and V
#THEN use it to make predictions
num_ratings = len(train_rating)
for iter in range(iterations): #Run SGD loop iterations times
  #Select a data point randomly
  k = np.random.randint(0,num_ratings)
  #Get the user and movie corresponding to this data point
  i = train_user[k]
  x = train_movie[k]
  #Calculate epsilon_k
  e_k = 2 * (train_rating[k] - (np.dot(U[i].T, V[x]) + mu + a[i] +
b[x]))
  #Update U[i], V[x], a[i] and b[x]
  U[i] = U[i] + alpha_U * (e_k * V[x] - lambda_U * U[i])
  V[x] = V[x] + alpha_V * (e_k * U[i] - lambda_V * V[x])
  a[i] = a[i] + alpha_a * (e_k - lambda_a * a[i])
```

```
  b[x] = b[x] + alpha_b * (e_k - lambda_b * b[x])


#Now I have a trained  set of user vectors U and movie vectors V
train_predictions = predict(mu, a, b, U, V, train_user, train_movie)
valid_predictions = predict(mu, a, b, U, V, valid_user, valid_movie)

train_rmse = rmse(train_predictions, train_rating)
valid_rmse = rmse(valid_predictions, valid_rating)

print('train_rmse=%.3f, valid_rmse=%.3f' % (train_rmse, valid_rmse))

##############################################
# Testing
##############################################

# Make and save predictions for test set
test_predictions = predict(mu, a, b, U, V, test_user, test_movie)
np.savetxt('test_predictions.txt', test_predictions)
```

```
train_rmse=0.897, valid_rmse=0.938
```

```
##############################################
# Tunable parameters (you will add more)
##############################################

nDims = 30 #100

#Regularization constants
lambda_U = 5e-3 #Regularization constant for user vectors, 4
lambda_V = 5e-3 #Regularization constant for movie vectors, 15
lambda_a = 1e-4 #Regularization constant for user biases, 0.05
lambda_b = 1e-4 #Regularization constant for movie biases, 0.05
#lambdas = [lambda_U,lambda_V,lambda_a,lambda_b]

#Alpha/learning rates
alpha_U = 2.5e-3 #User vectors, 1e-2
alpha_V = 2.5e-3 #Movie vectors, 1e-2
alpha_a = 2.5e-3 #User biases, 1e-2
alpha_b = 2.5e-3 #Movie biases, 1e-2

#Number of iterations
iterations = 50

##############################################
# Initialize parameters
##############################################

mu = np.mean(train_rating)
```

```
a  = np.zeros(nUsers)
b  = np.zeros(nMovies)
U  = np.random.randn(nUsers, nDims)  *.01 # User weights
V  = np.random.randn(nMovies, nDims) *.01 # Movie features

##############################################
# Training and validation
##############################################

# TODO: write code to train model and evaluate performance on
validation set
#
#  predict() is a stub that predicts the overall mean for all user-
movie
#  pairs. Update it to take more parameters and make real predictions.

#Write the SGD loop HERE
#Use it to modify a, b, U, and V
#THEN use it to make predictions
num_ratings = len(train_rating)
for iter in range(iterations): #Run SGD loop iterations times
  for k in range(num_ratings):
    i = train_user[k]
    x = train_movie[k]
    #Calculate epsilon_k
    e_k = 2 * (train_rating[k] - (np.dot(U[i].T, V[x]) + mu + a[i] +
b[x]))
    #Update U[i], V[x], a[i] and b[x]
    U[i] = U[i] + alpha_U * (e_k * V[x] - lambda_U * U[i])
    V[x] = V[x] + alpha_V * (e_k * U[i] - lambda_V * V[x])
    a[i] = a[i] + alpha_a * (e_k - lambda_a * a[i])
    b[x] = b[x] + alpha_b * (e_k - lambda_b * b[x])



##############################################
# Testing
##############################################
train_predictions = predict(mu, a, b, U, V, train_user, train_movie)
valid_predictions = predict(mu, a, b, U, V, valid_user, valid_movie)

train_rmse = rmse(train_predictions, train_rating)
valid_rmse = rmse(valid_predictions, valid_rating)

print('train_rmse=%.5f, valid_rmse=%.5f' % (train_rmse, valid_rmse))

# Make and save predictions for test set
test_predictions = predict(mu, a, b, U, V, test_user, test_movie)
np.savetxt('test_predictions.txt', test_predictions)
```

```
train_rmse=0.74777, valid_rmse=0.92920
```

Bonus Material: Inspect Predictions for Different Users

After you have learned a good model, you may wish to interpret what it has learned. We can do this by looking at the most positive and most negative predictions for different users (or the movies that are bumped up or down from the baseline the most).

Read and run the code below to see if you can understand the predictions. (Note: the predictions won't make sense until you have learned a good model!)

```
all_movies = range(nMovies)

def get_lowest(vals):
    most_negative = np.argsort(vals)
    return most_negative

def get_highest(vals):
    most_negative = np.argsort(vals)
    most_positive = most_negative[::-1]
    return most_positive

k = 8
all_users = range(nUsers)
users_to_examine = all_users[0:5]

for user in users_to_examine:

    # Changes from baseline movie predictions for this user
    delta = np.dot(V, U[user,:])

    print('*** User %d ***' % (user))
    print('  Top movies')
    for i in get_highest(delta)[0:k]:
        print('    %+.4f  %s' % (delta[i], titles[i]))
    print('')

    print('  Bottom movies')
    for i in get_lowest(delta)[0:k]:
        print('    %+.4f  %s' % (delta[i], titles[i]))
    print('')
```

```
*** User 0 ***
  Top movies
    +0.0000  Crude Oasis, The (1995)
    +0.0000  Last Klezmer: Leopold Kozlowski, His Life and Music, The
(1995)
    +0.0000  Wend Kuuni (God's Gift) (1982)
    +0.0000  Glass Shield, The (1994)
    +0.0000  Scream of Stone (Schrei aus Stein) (1991)
```

```
    +0.0000  8 Heads in a Duffel Bag (1997)
    +0.0000  Lashou shentan (1992)
    +0.0000  Coldblooded (1995)

  Bottom movies
    -0.0000  Tainted (1998)
    -0.0000  Girls Town (1996)
    -0.0000  Walk in the Sun, A (1945)
    -0.0000  Wings of Courage (1995)
    -0.0000  Other Voices, Other Rooms (1997)
    -0.0000  Silence of the Palace, The (Saimt el Qusur) (1994)
    -0.0000  My Favorite Season (1993)
    -0.0000  Substance of Fire, The (1996)

*** User 1 ***
  Top movies
    +0.0000  Other Voices, Other Rooms (1997)
    +0.0000  Leopard Son, The (1996)
    +0.0000  Stranger, The (1994)
    +0.0000  Office Killer (1997)
    +0.0000  War at Home, The (1996)
    +0.0000  Nosferatu a Venezia (1986)
    +0.0000  Killer: A Journal of Murder (1995)
    +0.0000  Sliding Doors (1998)

  Bottom movies
    -0.0000  Further Gesture, A (1996)
    -0.0000  Land and Freedom (Tierra y libertad) (1995)
    -0.0000  I, Worst of All (Yo, la peor de todas) (1990)
    -0.0000  Silence of the Palace, The (Saimt el Qusur) (1994)
    -0.0000  Modern Affair, A (1995)
    -0.0000  Salut cousin! (1996)
    -0.0000  Wooden Man's Bride, The (Wu Kui) (1994)
    -0.0000  To Cross the Rubicon (1991)

*** User 2 ***
  Top movies
    +0.0000  Glass Shield, The (1994)
    +0.0000  Line King: Al Hirschfeld, The (1996)
    +0.0000  Sunchaser, The (1996)
    +0.0000  Land and Freedom (Tierra y libertad) (1995)
    +0.0000  Brothers in Trouble (1995)
    +0.0000  Golden Earrings (1947)
    +0.0000  Angela (1995)
    +0.0000  I, Worst of All (Yo, la peor de todas) (1990)

  Bottom movies
    -0.0000  Someone Else's America (1995)
    -0.0000  My Favorite Season (1993)
    -0.0000  Designated Mourner, The (1997)
```

```
    -0.0000  Nothing Personal (1995)
    -0.0000  Last Klezmer: Leopold Kozlowski, His Life and Music, The
(1995)
    -0.0000  Spirits of the Dead (Tre passi nel delirio) (1968)
    -0.0000  Death in Brunswick (1991)
    -0.0000  Foreign Student (1994)

*** User 3 ***
  Top movies
    +0.0005  Angel on My Shoulder (1946)
    +0.0004  Girls Town (1996)
    +0.0004  Mat' i syn (1997)
    +0.0003  Daens (1992)
    +0.0003  Eighth Day, The (1996)
    +0.0003  Lady of Burlesque (1943)
    +0.0003  Truth or Consequences, N.M. (1997)
    +0.0003  Silence of the Palace, The (Saimt el Qusur) (1994)

  Bottom movies
    -0.0005  Leopard Son, The (1996)
    -0.0004  Salut cousin! (1996)
    -0.0003  Further Gesture, A (1996)
    -0.0003  Promise, The (Versprechen, Das) (1994)
    -0.0003  Men With Guns (1997)
    -0.0003  Woman in Question, The (1950)
    -0.0002  Foreign Student (1994)
    -0.0002  Invitation, The (Zaproszenie) (1986)

*** User 4 ***
  Top movies
    +0.0000  Target (1995)
    +0.0000  Nothing Personal (1995)
    +0.0000  Death in Brunswick (1991)
    +0.0000  Dadetown (1995)
    +0.0000  Brothers in Trouble (1995)
    +0.0000  Mat' i syn (1997)
    +0.0000  Land and Freedom (Tierra y libertad) (1995)
    +0.0000  Silence of the Palace, The (Saimt el Qusur) (1994)

  Bottom movies
    -0.0000  Other Voices, Other Rooms (1997)
    -0.0000  Every Other Weekend (1990)
    -0.0000  Spanish Prisoner, The (1997)
    -0.0000  Wend Kuuni (God's Gift) (1982)
    -0.0000  Inkwell, The (1994)
    -0.0000  August (1996)
    -0.0000  Sleepover (1995)
    -0.0000  You So Crazy (1994)
```

More Bonus Material: Interpretation of Weight Vectors as Features

- So far we have described both $\u_i$ and $\v_j$ as *weight vectors* (since we don't have any features of movies and users). But, it is possible to interpret one or both of these vectors as **learned features**.
- For example, the first learned feature may discover a preference for comedy vs. drama. In this case:
  - The user feature value u_{i1} should be high if the user likes comedies and low if the user likes dramas better.
  - The movie feature value v_{j1} should be high if the movie is a comedy and low if it is a drama.
- Similarly, feature 2 might describe whether a movie is geared toward kids or adults
- In practice, the feature interpretations often find recognizable patterns but are not quite so clean to describe as the two examples above.

Run the code below to examine the movies with the highest and lowest feature values for some of the features in your learned model.

```
k = 5

features_to_examine = np.arange(0,10)

for feature in features_to_examine:

    feature_vals = V[:,feature]

    print ('*** Feature %d ***' % (feature))
    print ('  Movies with highest feature value')
    for i in get_highest(feature_vals)[0:k]:
        print ('    %+.4f  %s' % (feature_vals[i], titles[i]))
    print ('')

    print ('  Movies with lowest feature value')
    for i in get_lowest(feature_vals)[0:k]:
        print ('    %+.4f  %s' % (feature_vals[i], titles[i]))
    print ('')
```

```
*** Feature 0 ***
  Movies with highest feature value
    +0.0270  Love and Death on Long Island (1997)
    +0.0201  Crude Oasis, The (1995)
    +0.0179  Woman in Question, The (1950)
    +0.0171  Leopard Son, The (1996)
    +0.0167  Girls Town (1996)

  Movies with lowest feature value
    -0.0218  Symphonie pastorale, La (1946)
```

```
    -0.0173  Silence of the Palace, The (Saimt el Qusur) (1994)
    -0.0169  To Cross the Rubicon (1991)
    -0.0153  Further Gesture, A (1996)
    -0.0144  Jupiter's Wife (1994)

*** Feature 1 ***
  Movies with highest feature value
    +0.0179  Jupiter's Wife (1994)
    +0.0166  Angela (1995)
    +0.0139  Tigrero: A Film That Was Never Made (1994)
    +0.0128  To Cross the Rubicon (1991)
    +0.0122  Death in the Garden (Mort en ce jardin, La) (1956)

  Movies with lowest feature value
    -0.0249  Butcher Boy, The (1998)
    -0.0231  August (1996)
    -0.0213  Woman in Question, The (1950)
    -0.0201  Nemesis 2: Nebula (1995)
    -0.0155  Sleepover (1995)

*** Feature 2 ***
  Movies with highest feature value
    +0.0216  Vermont Is For Lovers (1992)
    +0.0195  I, Worst of All (Yo, la peor de todas) (1990)
    +0.0166  Brothers in Trouble (1995)
    +0.0156  Sliding Doors (1998)
    +0.0154  Leopard Son, The (1996)

  Movies with lowest feature value
    -0.0234  Hostile Intentions (1994)
    -0.0225  Star Kid (1997)
    -0.0218  Walk in the Sun, A (1945)
    -0.0182  Truth or Consequences, N.M. (1997)
    -0.0182  Target (1995)

*** Feature 3 ***
  Movies with highest feature value
    +0.0288  Tigrero: A Film That Was Never Made (1994)
    +0.0172  Crude Oasis, The (1995)
    +0.0166  Dadetown (1995)
    +0.0159  Promise, The (Versprechen, Das) (1994)
    +0.0157  Someone Else's America (1995)

  Movies with lowest feature value
    -0.0258  Wings of Courage (1995)
    -0.0219  Sliding Doors (1998)
    -0.0166  Brothers in Trouble (1995)
    -0.0164  Butcher Boy, The (1998)
    -0.0158  Line King: Al Hirschfeld, The (1996)
```

```
*** Feature 4 ***
  Movies with highest feature value
    +0.0250  Promise, The (Versprechen, Das) (1994)
    +0.0224  Woman in Question, The (1950)
    +0.0194  Wooden Man's Bride, The (Wu Kui) (1994)
    +0.0173  August (1996)
    +0.0161  Nightwatch (1997)

  Movies with lowest feature value
    -0.0277  Wend Kuuni (God's Gift) (1982)
    -0.0258  Last Klezmer: Leopold Kozlowski, His Life and Music, The
(1995)
    -0.0233  Coldblooded (1995)
    -0.0195  Leopard Son, The (1996)
    -0.0134  Every Other Weekend (1990)

*** Feature 5 ***
  Movies with highest feature value
    +0.0203  Death in Brunswick (1991)
    +0.0189  Dadetown (1995)
    +0.0188  Venice/Venice (1992)
    +0.0151  Target (1995)
    +0.0143  Object of My Affection, The (1998)

  Movies with lowest feature value
    -0.0233  Big One, The (1997)
    -0.0220  Glass Shield, The (1994)
    -0.0220  Crude Oasis, The (1995)
    -0.0135  Wend Kuuni (God's Gift) (1982)
    -0.0124  Getting Away With Murder (1996)

*** Feature 6 ***
  Movies with highest feature value
    +0.0198  Last Klezmer: Leopold Kozlowski, His Life and Music, The
(1995)
    +0.0166  Power 98 (1995)
    +0.0148  Every Other Weekend (1990)
    +0.0145  Daens (1992)
    +0.0144  Woman in Question, The (1950)

  Movies with lowest feature value
    -0.0262  Nothing Personal (1995)
    -0.0233  To Cross the Rubicon (1991)
    -0.0185  Shadows (Cienie) (1988)
    -0.0172  Salut cousin! (1996)
    -0.0156  Land and Freedom (Tierra y libertad) (1995)

*** Feature 7 ***
```

```
  Movies with highest feature value
    +0.0259  King of New York (1990)
    +0.0216  8 Heads in a Duffel Bag (1997)
    +0.0168  Mat' i syn (1997)
    +0.0164  Power 98 (1995)
    +0.0157  Pushing Hands (1992)

  Movies with lowest feature value
    -0.0219  Sliding Doors (1998)
    -0.0215  Other Voices, Other Rooms (1997)
    -0.0147  Angel on My Shoulder (1946)
    -0.0138  Wooden Man's Bride, The (Wu Kui) (1994)
    -0.0123  Butcher Boy, The (1998)

*** Feature 8 ***
  Movies with highest feature value
    +0.0223  Lashou shentan (1992)
    +0.0213  Power 98 (1995)
    +0.0193  Shadows (Cienie) (1988)
    +0.0153  I Don't Want to Talk About It (De eso no se habla) (1993)
    +0.0141  Coldblooded (1995)

  Movies with lowest feature value
    -0.0279  Modern Affair, A (1995)
    -0.0274  Walk in the Sun, A (1945)
    -0.0233  Dadetown (1995)
    -0.0206  Angel on My Shoulder (1946)
    -0.0189  Brothers in Trouble (1995)

*** Feature 9 ***
  Movies with highest feature value
    +0.0227  Substance of Fire, The (1996)
    +0.0224  Getting Away With Murder (1996)
    +0.0184  Every Other Weekend (1990)
    +0.0155  Good Morning (1971)
    +0.0130  Nico Icon (1995)

  Movies with lowest feature value
    -0.0213  Object of My Affection, The (1998)
    -0.0188  Brothers in Trouble (1995)
    -0.0184  Love and Death on Long Island (1997)
    -0.0175  Lady of Burlesque (1943)
    -0.0174  Lamerica (1994)
```