Deep Learning for Content-Based Filtering

In this exercise, we implemented content-based filtering using a neural network to build a recommender system for movies.

In [1]:
```
import numpy as np
import numpy.ma as ma
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
import tabulate
```

2 - Movie ratings dataset

The data set is derived from the MovieLens ml-latest-small dataset.

In [2]:
```
top10_df = pd.read_csv("..../top10_df.csv")
bygenre_df = pd.read_csv("..../bygenre_df.csv")
top10_df
```

Out[2]:

	movie id	num ratings	ave rating	title	genres
0	4993	198	4.1	Lord of the Rings: The Fellowship of the Ring,...	Adventure|Fantasy
1	5952	188	4.0	Lord of the Rings: The Two Towers, The	Adventure|Fantasy
2	7153	185	4.1	Lord of the Rings: The Return of the King, The	Action|Adventure|Drama|Fantasy
3	4306	170	3.9	Shrek	Adventure|Animation|Children|Comedy|Fantasy|Ro...
4	58559	149	4.2	Dark Knight, The	Action|Crime|Drama
5	6539	149	3.8	Pirates of the Caribbean: The Curse of the Bla...	Action|Adventure|Comedy|Fantasy
6	79132	143	4.1	Inception	Action|Crime|Drama|Mystery|Sci-Fi|Thriller
7	6377	141	4.0	Finding Nemo	Adventure|Animation|Children|Comedy
8	4886	132	3.9	Monsters, Inc.	Adventure|Animation|Children|Comedy|Fantasy
9	7361	131	4.2	Eternal Sunshine of the Spotless Mind	Drama|Romance|Sci-Fi

The next table shows information sorted by genre. The number of ratings per genre vary substantially.

In [3]:
```
bygenre_df
```

Out[3]:

	genre	num movies	ave rating/genre	ratings per genre
0	Action	321	3.4	10377
1	Adventure	234	3.4	8785
2	Animation	76	3.6	2588
3	Children	69	3.4	2472
4	Comedy	326	3.4	8911
5	Crime	139	3.5	4671
6	Documentary	13	3.8	280
7	Drama	342	3.6	10201
8	Fantasy	124	3.4	4468
9	Horror	56	3.2	1345
10	Mystery	68	3.6	2497
11	Romance	151	3.4	4468
12	Sci-Fi	174	3.4	5894
13	Thriller	245	3.4	7659

3 - Content-based filtering with a neural network

3.1 Training Data

In [4]:
```
# Load Data, set configuration variables
item_train, user_train, y_train, item_features, user_features, item_vecs, movie_dict,

num_user_features = user_train.shape[1] - 3  # remove userid, rating count and ave rat
num_item_features = item_train.shape[1] - 1  # remove movie id at train time
uvs = 3  # user genre vector start
ivs = 3  # item genre vector start
u_s = 3  # start of columns to use in training, user
i_s = 1  # start of columns to use in training, items
print(f"Number of training vectors: {len(item_train)}")
```

```
Number of training vectors: 50884
```

In [5]:
```
pprint_train(user_train, user_features, uvs,  u_s, maxcount=5)
```

Out[5]:

[user id]	[rating count]	[rating ave]	Act ion	Adve nture	Anim ation	Chil dren	Com edy	Crime	Docum entary	Drama	Fan tasy	Hor ror	Mys tery	Rom ance
2	22	4.0	4.0	4.2	0.0	0.0	4.0	4.1	4.0	4.0	0.0	3.0	4.0	0.0
2	22	4.0	4.0	4.2	0.0	0.0	4.0	4.1	4.0	4.0	0.0	3.0	4.0	0.0
2	22	4.0	4.0	4.2	0.0	0.0	4.0	4.1	4.0	4.0	0.0	3.0	4.0	0.0
2	22	4.0	4.0	4.2	0.0	0.0	4.0	4.1	4.0	4.0	0.0	3.0	4.0	0.0
2	22	4.0	4.0	4.2	0.0	0.0	4.0	4.1	4.0	4.0	0.0	3.0	4.0	0.0

The features in brackets "[]" such as the "user id", "rating count" and "rating ave" are not included when the model is trained and used.For each user, Zero entries are genre's which the user had not rated. The movie array contains the year the film was released, the average rating and an indicator for each potential genre.The target, y, is the movie rating given by the user.

In [6]:

```
pprint_train(item_train, item_features, ivs, i_s, maxcount=5, user=False)
```

Out[6]:

[movie id]	year	ave rating	Act ion	Adve nture	Anim ation	Chil dren	Com edy	Crime	Docum entary	Drama	Fan tasy	Hor ror	Mys tery	Rom ance
6874	2003	4.0	1	0	0	0	0	1	0	0	0	0	0	0
8798	2004	3.8	1	0	0	0	0	1	0	1	0	0	0	0
46970	2006	3.2	1	0	0	0	1	0	0	0	0	0	0	0
48516	2006	4.3	0	0	0	0	0	1	0	1	0	0	0	0
58559	2008	4.2	1	0	0	0	0	1	0	1	0	0	0	0

3.2 Preparing the training data

In [8]:

```
# scale training data
item_train_unscaled = item_train
user_train_unscaled = user_train
y_train_unscaled    = y_train

scalerItem = StandardScaler()
scalerItem.fit(item_train)
item_train = scalerItem.transform(item_train)

scalerUser = StandardScaler()
scalerUser.fit(user_train)
user_train = scalerUser.transform(user_train)

scalerTarget = MinMaxScaler((-1, 1))
scalerTarget.fit(y_train.reshape(-1, 1))
y_train = scalerTarget.transform(y_train.reshape(-1, 1))

# check if transformed data same with unscaled data
print(np.allclose(item_train_unscaled, scalerItem.inverse_transform(item_train)))
print(np.allclose(user_train_unscaled, scalerUser.inverse_transform(user_train)))
```

```
True
True
```

In [9]:
```
## split data into 80% train and 20% test
item_train, item_test = train_test_split(item_train, train_size=0.80, shuffle=True, ra
user_train, user_test = train_test_split(user_train, train_size=0.80, shuffle=True, ra
y_train, y_test       = train_test_split(y_train,    train_size=0.80, shuffle=True, ra
print(f"movie/item training data shape: {item_train.shape}")
print(f"movie/item test data shape: {item_test.shape}")
```

```
movie/item training data shape: (40707, 17)
movie/item test data shape: (10177, 17)
```

4 - Neural Network for content-based filtering

In [12]:
```
# we constructure 3 layers NN with the first layer 256 nodes, the second layer 128 noc


num_outputs = 32
tf.random.set_seed(1)
user_NN = tf.keras.models.Sequential([

    tf.keras.layers.Dense(256,activation="relu"),
    tf.keras.layers.Dense(128,activation="relu"),
    tf.keras.layers.Dense(32)

])

item_NN = tf.keras.models.Sequential([

    tf.keras.layers.Dense(256,activation="relu"),
    tf.keras.layers.Dense(128,activation="relu"),
    tf.keras.layers.Dense(32)

])

# create the user input and point to the base network
input_user = tf.keras.layers.Input(shape=(num_user_features))
vu = user_NN(input_user)
vu = tf.linalg.l2_normalize(vu, axis=1)

# create the item input and point to the base network
input_item = tf.keras.layers.Input(shape=(num_item_features))
vm = item_NN(input_item)
vm = tf.linalg.l2_normalize(vm, axis=1)

# compute the dot product of the two vectors vu and vm
output = tf.keras.layers.Dot(axes=1)([vu, vm])

# specify the inputs and output of the model
model = tf.keras.Model([input_user, input_item], output)

model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 14)]         0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 16)]         0
__________________________________________________________________________________________________
sequential (Sequential)         (None, 32)           40864       input_1[0][0]
__________________________________________________________________________________________________
sequential_1 (Sequential)       (None, 32)           41376       input_2[0][0]
__________________________________________________________________________________________________
tf_op_layer_l2_normalize/Square [(None, 32)]         0           sequential[0][0]
__________________________________________________________________________________________________
tf_op_layer_l2_normalize_1/Squa [(None, 32)]         0           sequential_1[0][0]
__________________________________________________________________________________________________
tf_op_layer_l2_normalize/Sum (T [(None, 1)]          0           tf_op_layer_l2_norma
lize/Square[0
__________________________________________________________________________________________________
tf_op_layer_l2_normalize_1/Sum  [(None, 1)]          0           tf_op_layer_l2_norma
lize_1/Square
__________________________________________________________________________________________________
tf_op_layer_l2_normalize/Maximu [(None, 1)]          0           tf_op_layer_l2_norma
lize/Sum[0][0
__________________________________________________________________________________________________
tf_op_layer_l2_normalize_1/Maxi [(None, 1)]          0           tf_op_layer_l2_norma
lize_1/Sum[0]
__________________________________________________________________________________________________
tf_op_layer_l2_normalize/Rsqrt  [(None, 1)]          0           tf_op_layer_l2_norma
lize/Maximum[
__________________________________________________________________________________________________
tf_op_layer_l2_normalize_1/Rsqr [(None, 1)]          0           tf_op_layer_l2_norma
lize_1/Maximu
__________________________________________________________________________________________________
tf_op_layer_l2_normalize (Tenso [(None, 32)]         0           sequential[0][0]
                                                                 tf_op_layer_l2_norma

lize/Rsqrt[0]
__________________________________________________________________________________________________
tf_op_layer_l2_normalize_1 (Ten [(None, 32)]         0           sequential_1[0][0]
                                                                 tf_op_layer_l2_norma

lize_1/Rsqrt[
__________________________________________________________________________________________________
dot (Dot)                       (None, 1)            0           tf_op_layer_l2_norma
lize[0][0]
```

```
                                                                    tf_op_layer_l2_norma
lize_1[0][0]
====================================================================================
=============
Total params: 82,240
Trainable params: 82,240
Non-trainable params: 0
____________________________________________________________________________________
_____________
```

In [14]:
```
# mean squared error loss and Adam optimizer
tf.random.set_seed(1)
cost_fn = tf.keras.losses.MeanSquaredError()
opt = keras.optimizers.Adam(learning_rate=0.01)
model.compile(optimizer=opt,
              loss=cost_fn)
```

In [15]:
```
# model fitting
tf.random.set_seed(1)
model.fit([user_train[:, u_s:], item_train[:, i_s:]], y_train, epochs=30)
```

```
Train on 40707 samples
Epoch 1/30
40707/40707 [==============================] - 6s 138us/sample - loss: 0.1232
Epoch 2/30
40707/40707 [==============================] - 5s 126us/sample - loss: 0.1146
Epoch 3/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.1089
Epoch 4/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.1039
Epoch 5/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.1001
Epoch 6/30
40707/40707 [==============================] - 5s 125us/sample - loss: 0.0973
Epoch 7/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.0956
Epoch 8/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.0935
Epoch 9/30
40707/40707 [==============================] - 5s 125us/sample - loss: 0.0916
Epoch 10/30
40707/40707 [==============================] - 5s 125us/sample - loss: 0.0897
Epoch 11/30
40707/40707 [==============================] - 5s 125us/sample - loss: 0.0880
Epoch 12/30
40707/40707 [==============================] - 5s 125us/sample - loss: 0.0865
Epoch 13/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.0852
Epoch 14/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.0839
Epoch 15/30
40707/40707 [==============================] - 5s 125us/sample - loss: 0.0830
Epoch 16/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.0815
Epoch 17/30
40707/40707 [==============================] - 5s 125us/sample - loss: 0.0807
Epoch 18/30
40707/40707 [==============================] - 5s 125us/sample - loss: 0.0796
Epoch 19/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.0786
Epoch 20/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.0776
Epoch 21/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.0769
Epoch 22/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.0761
Epoch 23/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.0755
Epoch 24/30
40707/40707 [==============================] - 5s 125us/sample - loss: 0.0746
Epoch 25/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.0741
Epoch 26/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.0733
Epoch 27/30
40707/40707 [==============================] - 5s 125us/sample - loss: 0.0728
Epoch 28/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.0723
Epoch 29/30
40707/40707 [==============================] - 5s 123us/sample - loss: 0.0717
```

```
Epoch 30/30
40707/40707 [==============================] - 5s 125us/sample - loss: 0.0713
```

Out[15]:
```
<tensorflow.python.keras.callbacks.History at 0x7fcffe03e050>
```

In [16]:
```
# Evaluate the model with the test data
model.evaluate([user_test[:, u_s:], item_test[:, i_s:]], y_test)
```

```
10177/10177 [==============================] - 0s 38us/sample - loss: 0.0815
```

Out[16]:
```
0.08146006993124337
```

5 - Predictions

5.1 - Predictions for a new user

In [17]:
```
# create a new user and let model suggest movies for this user
new_user_id = 5000
new_rating_ave = 0.0
new_action = 0.0
new_adventure = 5.0
new_animation = 0.0
new_childrens = 0.0
new_comedy = 0.0
new_crime = 0.0
new_documentary = 0.0
new_drama = 0.0
new_fantasy = 5.0
new_horror = 0.0
new_mystery = 0.0
new_romance = 0.0
new_scifi = 0.0
new_thriller = 0.0
new_rating_count = 3

user_vec = np.array([[new_user_id, new_rating_count, new_rating_ave,
                      new_action, new_adventure, new_animation, new_childrens,
                      new_comedy, new_crime, new_documentary,
                      new_drama, new_fantasy, new_horror, new_mystery,
                      new_romance, new_scifi, new_thriller]])
```

In [18]:
```
# generate and replicate the user vector to match the number movies in the data set.
user_vecs = gen_user_vecs(user_vec,len(item_vecs))

# scale our user and item vectors
suser_vecs = scalerUser.transform(user_vecs)
sitem_vecs = scalerItem.transform(item_vecs)

# make a prediction
y_p = model.predict([suser_vecs[:, u_s:], sitem_vecs[:, i_s:]])

# unscale y prediction
y_pu = scalerTarget.inverse_transform(y_p)

# sort the results, highest prediction first
sorted_index = np.argsort(-y_pu,axis=0).reshape(-1).tolist()  #negate to get largest r
sorted_ypu   = y_pu[sorted_index]
sorted_items = item_vecs[sorted_index]  #using unscaled vectors for display
```

```
print_pred_movies(sorted_ypu, sorted_items, movie_dict, maxcount = 10)
```

Out[18]:

y_p	movie id	rating ave	title	genres
4.5	98809	3.8	Hobbit: An Unexpected Journey, The (2012)	Adventure|Fantasy
4.4	8368	3.9	Harry Potter and the Prisoner of Azkaban (2004)	Adventure|Fantasy
4.4	54001	3.9	Harry Potter and the Order of the Phoenix (2007)	Adventure|Drama|Fantasy
4.3	40815	3.8	Harry Potter and the Goblet of Fire (2005)	Adventure|Fantasy|Thriller
4.3	106489	3.6	Hobbit: The Desolation of Smaug, The (2013)	Adventure|Fantasy
4.3	81834	4	Harry Potter and the Deathly Hallows: Part 1 (2010)	Action|Adventure|Fantasy
4.3	59387	4	Fall, The (2006)	Adventure|Drama|Fantasy
4.3	5952	4	Lord of the Rings: The Two Towers, The (2002)	Adventure|Fantasy
4.3	5816	3.6	Harry Potter and the Chamber of Secrets (2002)	Adventure|Fantasy
4.3	54259	3.6	Stardust (2007)	Adventure|Comedy|Fantasy|Romance

5.2 - Predictions for an existing user.

In [19]:

```
uid = 2
# form a set of user vectors. This is the same vector, transformed and repeated.
user_vecs, y_vecs = get_user_vecs(uid, user_train_unscaled, item_vecs, user_to_genre)

# scale our user and item vectors
suser_vecs = scalerUser.transform(user_vecs)
sitem_vecs = scalerItem.transform(item_vecs)

# make a prediction
y_p = model.predict([suser_vecs[:, u_s:], sitem_vecs[:, i_s:]])

# unscale y prediction
y_pu = scalerTarget.inverse_transform(y_p)

# sort the results, highest prediction first
sorted_index = np.argsort(-y_pu,axis=0).reshape(-1).tolist()  #negate to get largest r
sorted_ypu   = y_pu[sorted_index]
sorted_items = item_vecs[sorted_index]  #using unscaled vectors for display
sorted_user  = user_vecs[sorted_index]
sorted_y     = y_vecs[sorted_index]

#print sorted predictions for movies rated by the user
print_existing_user(sorted_ypu, sorted_y.reshape(-1,1), sorted_user, sorted_items, ivs
```

Out[19]:

y_p	y	user	user genre ave	movie rating ave	movie id	title	genres
4.5	5.0	2	[4.0]	4.3	80906	Inside Job (2010)	Documentary
4.2	3.5	2	[4.0,4.0]	3.9	99114	Django Unchained (2012)	Action|Drama
4.1	4.5	2	[4.0,4.0]	4.1	68157	Inglourious Basterds (2009)	Action|Drama
4.1	3.5	2	[4.0,3.9,3.9]	3.9	115713	Ex Machina (2015)	Drama|Sci-Fi|Thriller
4.0	4.0	2	[4.0,4.1,4.0,4.0,3.9,3.9]	4.1	79132	Inception (2010)	Action|Crime|Drama|Mystery|Sci-Fi|Thriller
4.0	4.0	2	[4.1,4.0,3.9]	4.3	48516	Departed, The (2006)	Crime|Drama|Thriller
4.0	4.5	2	[4.0,4.1,4.0]	4.2	58559	Dark Knight, The (2008)	Action|Crime|Drama
4.0	4.0	2	[4.0,4.1,3.9]	4.0	6874	Kill Bill: Vol. 1 (2003)	Action|Crime|Thriller
4.0	3.5	2	[4.0,4.1,4.0,3.9]	3.8	8798	Collateral (2004)	Action|Crime|Drama|Thriller
3.9	5.0	2	[4.0,4.1,4.0]	3.9	106782	Wolf of Wall Street, The (2013)	Comedy|Crime|Drama
3.9	3.5	2	[4.0,4.2,4.1]	4.0	91529	Dark Knight Rises, The (2012)	Action|Adventure|Crime
3.9	4.0	2	[4.0,4.0,3.9]	4.0	74458	Shutter Island (2010)	Drama|Mystery|Thriller
3.9	4.5	2	[4.1,4.0,3.9]	4.0	80489	Town, The (2010)	Crime|Drama|Thriller
3.8	4.0	2	[4.0]	4.0	112552	Whiplash (2014)	Drama
3.8	3.0	2	[3.9]	4.0	109487	Interstellar (2014)	Sci-Fi
3.8	5.0	2	[4.0]	3.7	89774	Warrior (2011)	Drama
3.7	3.0	2	[4.0,4.0,3.0]	3.9	71535	Zombieland (2009)	Action|Comedy|Horror
3.7	5.0	2	[4.0,4.2,3.9,3.9]	3.8	122882	Mad Max: Fury Road (2015)	Action|Adventure|Sci-Fi|Thriller
3.5	5.0	2	[4.0]	3.6	60756	Step Brothers (2008)	Comedy

y_p	y	user	user genre ave	movie rating ave	movie id	title	genres
3.5	2.5	2	[4.0,3.9]	3.5	91658	Girl with the Dragon Tattoo, The (2011)	Drama|Thriller
3.1	3.0	2	[4.0,4.0]	4.0	77455	Exit Through the Gift Shop (2010)	Comedy|Documentary
3.1	4.0	2	[4.0,4.0]	3.2	46970	Talladega Nights: The Ballad of Ricky Bobby (2006)	Action|Comedy

5.3 - Finding Similar Items

In [20]:
```
# find the squared distance between two vectors(Vm(k) and Vm(i))
def sq_dist(a,b):
    d = np.sum((a-b)**2)
    return d
```

In [21]:
```
a1 = np.array([1.0, 2.0, 3.0]); b1 = np.array([1.0, 2.0, 3.0])
a2 = np.array([1.1, 2.1, 3.1]); b2 = np.array([1.0, 2.0, 3.0])
a3 = np.array([0, 1, 0]);       b3 = np.array([1, 0, 0])
print(f"squared distance between a1 and b1: {sq_dist(a1, b1):0.3f}")
print(f"squared distance between a2 and b2: {sq_dist(a2, b2):0.3f}")
print(f"squared distance between a3 and b3: {sq_dist(a3, b3):0.3f}")
```

```
squared distance between a1 and b1: 0.000
squared distance between a2 and b2: 0.030
squared distance between a3 and b3: 2.000
```

In [23]:
```
## using trained 'item_NN' to build a small model to generate vm
input_item_m = tf.keras.layers.Input(shape=(num_item_features))    # input layer
vm_m = item_NN(input_item_m)                                       # use the trained i
vm_m = tf.linalg.l2_normalize(vm_m, axis=1)                        # incorporate normo
model_m = tf.keras.Model(input_item_m, vm_m)
model_m.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 16)]         0
__________________________________________________________________________________________________
sequential_1 (Sequential)       (None, 32)           41376       input_3[0][0]
__________________________________________________________________________________________________
tf_op_layer_l2_normalize_2/Squa [(None, 32)]         0           sequential_1[1][0]
__________________________________________________________________________________________________
tf_op_layer_l2_normalize_2/Sum  [(None, 1)]          0           tf_op_layer_l2_norma
lize_2/Square
__________________________________________________________________________________________________
tf_op_layer_l2_normalize_2/Maxi [(None, 1)]          0           tf_op_layer_l2_norma
lize_2/Sum[0]
__________________________________________________________________________________________________
tf_op_layer_l2_normalize_2/Rsqr [(None, 1)]          0           tf_op_layer_l2_norma
lize_2/Maximu
__________________________________________________________________________________________________
tf_op_layer_l2_normalize_2 (Ten [(None, 32)]         0           sequential_1[1][0]
                                                                 tf_op_layer_l2_norma
lize_2/Rsqrt[
==================================================================================================
Total params: 41,376
Trainable params: 41,376
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [24]:

```python
# using a set of item/movie vectors as input and using the model to predict a set of m
scaled_item_vecs = scalerItem.transform(item_vecs)
vms = model_m.predict(scaled_item_vecs[:,i_s:])
print(f"size of all predicted movie feature vectors: {vms.shape}")
```

```
size of all predicted movie feature vectors: (847, 32)
```

In [25]:

```python
# find the closest movie by finding the minimum along each row
count = 50  # number of movies to display
dim = len(vms)
dist = np.zeros((dim,dim))

for i in range(dim):
    for j in range(dim):
        dist[i,j] = sq_dist(vms[i, :], vms[j, :])

m_dist = ma.masked_array(dist, mask=np.identity(dist.shape[0]))  # mask the diagonal,

disp = [["movie1", "genres", "movie2", "genres"]]
for i in range(count):
    min_idx = np.argmin(m_dist[i])
    movie1_id = int(item_vecs[i,0])
```

```
    movie2_id = int(item_vecs[min_idx,0])
    disp.append( [movie_dict[movie1_id]['title'], movie_dict[movie1_id]['genres'],
                  movie_dict[movie2_id]['title'], movie_dict[movie1_id]['genres']]
               )
table = tabulate.tabulate(disp, tablefmt='html', headers="firstrow")
table
```

Out[25]:

movie1	genres	movie2	
Save the Last Dance (2001)	Drama|Romance	Mona Lisa Smile (2003)	
Wedding Planner, The (2001)	Comedy|Romance	Mr. Deeds (2002)	
Hannibal (2001)	Horror|Thriller	Final Destination 2 (2003)	
Saving Silverman (Evil Woman) (2001)	Comedy|Romance	Down with Love (2003)	
Down to Earth (2001)	Comedy|Fantasy|Romance	Bewitched (2005)	
Mexican, The (2001)	Action|Comedy	Rush Hour 2 (2001)	
15 Minutes (2001)	Thriller	Panic Room (2002)	
Enemy at the Gates (2001)	Drama	Kung Fu Hustle (Gong fu) (2004)	
Heartbreakers (2001)	Comedy|Crime|Romance	Fun with Dick and Jane (2005)	
Spy Kids (2001)	Action|Adventure|Children|Comedy	Tuxedo, The (2002)	Acti
Along Came a Spider (2001)	Action|Crime|Mystery|Thriller	Insomnia (2002)	
Blow (2001)	Crime|Drama	25th Hour (2002)	
Bridget Jones's Diary (2001)	Comedy|Drama|Romance	Punch-Drunk Love (2002)	
Joe Dirt (2001)	Adventure|Comedy|Mystery|Romance	Polar Express, The (2004)	Adven
Crocodile Dundee in Los Angeles (2001)	Comedy|Drama	Bewitched (2005)	
Mummy Returns, The (2001)	Action|Adventure|Comedy|Thriller	Rundown, The (2003)	Ac
Knight's Tale, A (2001)	Action|Comedy|Romance	Legally Blonde	

movie1	genres	movie2	
		(2001)	
Shrek (2001)	Adventure|Animation|Children|Comedy|Fantasy|Romance	Tangled (2010)	Adventure|Animation|Chil
Moulin Rouge (2001)	Drama|Romance	Notebook, The (2004)	
Pearl Harbor (2001)	Action|Drama|Romance	Bridget Jones: The Edge of Reason (2004)	
Animal, The (2001)	Comedy	Dumb and Dumberer: When Harry Met Lloyd (2003)	
Evolution (2001)	Comedy|Sci-Fi	Behind Enemy Lines (2001)	
Swordfish (2001)	Action|Crime|Drama	We Were Soldiers (2002)	
Atlantis: The Lost Empire (2001)	Adventure|Animation|Children|Fantasy	Cloudy with a Chance of Meatballs (2009)	Advent
Lara Croft: Tomb Raider (2001)	Action|Adventure	National Treasure: Book of Secrets (2007)	
Dr. Dolittle 2 (2001)	Comedy	Legally Blonde 2: Red, White & Blonde (2003)	
Fast and the Furious, The (2001)	Action|Crime|Thriller	xXx (2002)	
A.I. Artificial Intelligence (2001)	Adventure|Drama|Sci-Fi	Bubba Ho-tep (2002)	
Cats & Dogs (2001)	Children|Comedy	Robots (2005)	
Scary Movie 2 (2001)	Comedy	Orange County (2002)	
Final Fantasy: The Spirits	Adventure|Animation|Fantasy|Sci-Fi	Madagascar: Escape 2	Adv

movie1	genres	movie2
Within (2001)		Africa (2008)
Legally Blonde (2001)	Comedy|Romance	Serendipity (2001)
Score, The (2001)	Action|Drama	Punisher, The (2004)
Jurassic Park III (2001)	Action|Adventure|Sci-Fi|Thriller	Men in Black II (a.k.a. MIIB) (a.k.a. MIB 2) (2002)
America's Sweethearts (2001)	Comedy|Romance	Maid in Manhattan (2002)
Ghost World (2001)	Comedy|Drama	Station Agent, The (2003)
Planet of the Apes (2001)	Action|Adventure|Drama|Sci-Fi	Day After Tomorrow, The (2004)
Princess Diaries, The (2001)	Children|Comedy|Romance	Lake House, The (2006)
Rush Hour 2 (2001)	Action|Comedy	Mexican, The (2001)
American Pie 2 (2001)	Comedy	Rat Race (2001)
Others, The (2001)	Drama|Horror|Mystery|Thriller	The Machinist (2004)
Rat Race (2001)	Comedy	American Pie 2 (2001)
Jay and Silent Bob Strike Back (2001)	Adventure|Comedy	Mexican, The (2001)
Training Day (2001)	Crime|Drama|Thriller	Frailty (2001)
Zoolander (2001)	Comedy	Old School (2003)
Serendipity (2001)	Comedy|Romance	Legally Blonde (2001)
Mulholland Drive (2001)	Crime|Drama|Mystery|Thriller	Prisoners (2013)
From Hell (2001)	Crime|Horror|Mystery|Thriller	Identity (2003)

movie1	genres	movie2
Waking Life (2001)	Animation|Drama|Fantasy	Warm Bodies (2013)
K-PAX (2001)	Drama|Fantasy|Mystery|Sci-Fi	Gosford Park (2001)

The results show the model will generally suggest a movie with similar genre's.