# Content-Based Filtering

Implementing content-based filtering using a neural network to build a recommender system for movies.

## steps followed by code:

- 1- Movie ratings dataset
- 2- Content-based filtering with a neural network
  - Training Data
  - Preparing the training data
- 3- Neural Network for content-based filtering
- 4- Predictions
  - Predictions for a new user
  - Predictions for an existing user.
  - Finding Similar Items

In [2]:
```
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
from recsysNN_utils import *
pd.set_option("display.precision", 1)
```

### 1 - Movie ratings dataset:

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

[F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. https://doi.org/10.1145/2827872]

The original dataset has roughly 9000 movies rated by 600 users with ratings on a scale of 0.5 to 5 in 0.5 step increments. The dataset has been reduced in size to focus on movies from the years since 2000 and popular genres. The reduced dataset has $n_u = 397$ users, $n_m= 847$ movies and 25521 ratings. For each movie, the dataset provides a movie title, release date, and one or more genres. For example "Toy Story 3" was released in 2010 and has several genres: "Adventure|Animation|Children|Comedy|Fantasy". This dataset contains little information about users other than their ratings.This dataset is used to create training vectors for the neural networks described below.

- The table below shows the top 10 movies ranked by the number of ratings, and these movies also happen to have high average ratings:

In [3]:
```
top10_df = pd.read_csv("E:/new CV/ML/ML projects/content_based_RecSysNN/data/content_top
bygenre_df = pd.read_csv("E:/new CV/ML/ML projects/content_based_RecSysNN/data/content_b
top10_df
```

Loading [MathJax]/extensions/Safe.js

Out[3]:

| | 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, and the number of ratings per genre vary substantially:

In [5]:
```
bygenre_df
```

Out[5]:

| | 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 |

## 2 - Content-based filtering with a neural network:

## Training Data :

The movie content provided to the network is a combination of the original data and some 'engineered

Loading [MathJax]/extensions/Safe.js

iginal features are the year the movie was released and the movie's genre's presented as a one-hot vector. There are 14 genres. The engineered feature is an average rating derived from the user ratings.

The user content is composed of engineered features. A per genre average rating is computed per user. Additionally, a user id, rating count and rating average are available but not included in the training or prediction content. They are carried with the data set because they are useful in interpreting data.

The training set consists of all the ratings made by the users in the data set. Some ratings are repeated to boost the number of training examples of underrepresented genre's. The training set is split into two arrays with the same number of entries, a user array and a movie/item array.

In [6]:
```
# Load Data, set configuration variables
item_train, user_train, y_train, item_features, user_features, item_vecs, movie_dict, us

num_user_features = user_train.shape[1] - 3  # remove userid, rating count and ave ratin
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
```

The first few entries in the user training array:

In [7]:
```
pprint_train(user_train, user_features, uvs,  u_s, maxcount=5)
```

Out[7]:

| [user id] | [rating count] | [rating ave] | Action | Adventure | Animation | Children | Comedy | Crime | Documentary | Drama | Fantasy | Horror | Mystery | Romance | Sci-Fi | Thriller |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 | 3.9 | 3.9 |
| 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 | 3.9 | 3.9 |
| 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 | 3.9 | 3.9 |
| 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 | 3.9 | 3.9 |
| 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 | 3.9 | 3.9 |

Some of the user and item/movie features are not used in training. In the table above, the features in brackets "[]" such as the "user id", "rating count" and "rating ave" are not included when the model is trained and used.The user vector is the same for all the movies rated by a user.

The first few entries of the movie/item array:

In [8]:
```
pprint_train(item_train, item_features, ivs, i_s, maxcount=5, user=False)
```

Out[8]:

| [movie id] | year | ave rating | Action | Adventure | Animation | Children | Comedy | Crime | Documentary | Drama | Fantasy | Horror | Mystery | Romance | Sci-Fi | Thriller |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6874 | 2003 | 4.0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 8798 | 2004 | 3.8 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 46970 | 2006 | 3.2 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 48516 | 2006 | 4.3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 58559 | 2008 | 4.2 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

Loading [MathJax]/extensions/Safe.js

Above, the movie array contains the year the film was released, the average rating and an indicator for each potential genre. The indicator is one for each genre that applies to the movie. The movie id is not used in training but is useful when interpreting the data.

In [9]:
```
print(f"y_train[:5]: {y_train[:5]}")
```

```
y_train[:5]: [4.  3.5 4.  4.  4.5]
```

The target, y, is the movie rating given by the user

A single training example consists of a row from both the user and item arrays and a rating from y_train.

## Preparing the training data:

In [10]:
```
# 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))
#ynorm_test = scalerTarget.transform(y_test.reshape(-1, 1))

print(np.allclose(item_train_unscaled, scalerItem.inverse_transform(item_train)))
print(np.allclose(user_train_unscaled, scalerUser.inverse_transform(user_train)))
```

```
True
True
```

The data is splitted into training and test sets (using sklean train_test_split to split and shuffle the data).

In [11]:
```
item_train, item_test = train_test_split(item_train, train_size=0.80, shuffle=True, rand
user_train, user_test = train_test_split(user_train, train_size=0.80, shuffle=True, rand
y_train, y_test       = train_test_split(y_train,    train_size=0.80, shuffle=True, rand
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)
```

The scaled, shuffled data now has a mean of zero.

In [12]:
```
pprint_train(user_train, user_features, uvs, u_s, maxcount=5)
```

Loading [MathJax]/extensions/Safe.js

Out[12]:

| [user id] | [rating count] | [rating ave] | Action | Adventure | Animation | Children | Comedy | Crime | Documentary | Drama | Fantasy | Horror | Mystery | Romance | Sci-Fi | Thriller |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | -1.0 | -0.8 | -0.7 | 0.1 | -0.0 | -1.2 | -0.4 | 0.6 | -0.5 | -0.5 | -0.1 | -0.6 | -0.6 | -0.7 | -0.7 |
| 0 | 1 | -0.7 | -0.5 | -0.7 | -0.1 | -0.2 | -0.6 | -0.2 | 0.7 | -0.5 | -0.8 | 0.1 | -0.0 | -0.6 | -0.5 | -0.4 |
| -1 | -1 | -0.2 | 0.3 | -0.4 | 0.4 | 0.5 | 1.0 | 0.6 | -1.2 | -0.3 | -0.6 | -2.3 | -0.1 | 0.0 | 0.4 | -0.0 |
| 0 | -1 | 0.6 | 0.5 | 0.5 | 0.2 | 0.6 | -0.1 | 0.5 | -1.2 | 0.9 | 1.2 | -2.3 | -0.1 | 0.0 | 0.2 | 0.3 |
| -1 | 0 | 0.7 | 0.6 | 0.5 | 0.3 | 0.5 | 0.4 | 0.6 | 1.0 | 0.6 | 0.3 | 0.8 | 0.8 | 0.4 | 0.7 | 0.7 |

## 3 - Neural Network for content-based filtering:

- Use a Keras sequential model
  - The first layer is a dense layer with 256 units and a relu activation.
  - The second layer is a dense layer with 128 units and a relu activation.
  - The third layer is a dense layer with num_outputs units and a linear or no activation.

In [13]:

```
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(num_outputs),


])

item_NN = tf.keras.models.Sequential([

    tf.keras.layers.Dense(256, activation = 'relu'),
    tf.keras.layers.Dense(128, activation = 'relu'),
    tf.keras.layers.Dense(num_outputs),


])

# 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()
```

Loading [MathJax]/extensions/Safe.js

```
WARNING:tensorflow:From C:\Users\sanaz\anaconda3\lib\site-packages\keras\src\backend.py:
873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_gr
aph instead.

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.math.l2_normalize (TFOp  (None, 32)                   0         ['sequential[0][0]']
 Lambda)

 tf.math.l2_normalize_1 (TF  (None, 32)                   0         ['sequential_1[0]
[0]']
 OpLambda)

 dot (Dot)                   (None, 1)                    0         ['tf.math.l2_normali
ze[0][0]',
                                                                     'tf.math.l2_normali
ze_1[0][0]
                                                                    ']

==========================================================================================
==========
Total params: 82240 (321.25 KB)
Trainable params: 82240 (321.25 KB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________
__________
```

In [14]:

```
# Public tests
from public_tests_recsysNN import *
test_tower(user_NN)
test_tower(item_NN)
```

```
All tests passed!
All tests passed!
```

A mean squared error loss and an Adam optimizer are used.

In [15]:

```
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 [16]:

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

Loading [MathJax]/extensions/Safe.js

```
Epoch 1/30
WARNING:tensorflow:From C:\Users\sanaz\anaconda3\lib\site-packages\keras\src\utils\tf_ut
ils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.
ragged.RaggedTensorValue instead.

1273/1273 [==============================] - 3s 2ms/step - loss: 0.1236
Epoch 2/30
1273/1273 [==============================] - 5s 4ms/step - loss: 0.1141
Epoch 3/30
1273/1273 [==============================] - 7s 5ms/step - loss: 0.1095
Epoch 4/30
1273/1273 [==============================] - 7s 5ms/step - loss: 0.1058
Epoch 5/30
1273/1273 [==============================] - 7s 5ms/step - loss: 0.1035
Epoch 6/30
1273/1273 [==============================] - 7s 5ms/step - loss: 0.1006
Epoch 7/30
1273/1273 [==============================] - 7s 5ms/step - loss: 0.0983
Epoch 8/30
1273/1273 [==============================] - 7s 5ms/step - loss: 0.0964
Epoch 9/30
1273/1273 [==============================] - 5s 4ms/step - loss: 0.0943
Epoch 10/30
1273/1273 [==============================] - 5s 4ms/step - loss: 0.0929
Epoch 11/30
1273/1273 [==============================] - 7s 6ms/step - loss: 0.0911
Epoch 12/30
1273/1273 [==============================] - 6s 4ms/step - loss: 0.0895
Epoch 13/30
1273/1273 [==============================] - 7s 5ms/step - loss: 0.0883
Epoch 14/30
1273/1273 [==============================] - 7s 6ms/step - loss: 0.0871
Epoch 15/30
1273/1273 [==============================] - 7s 6ms/step - loss: 0.0858
Epoch 16/30
1273/1273 [==============================] - 7s 6ms/step - loss: 0.0844
Epoch 17/30
1273/1273 [==============================] - 6s 5ms/step - loss: 0.0832
Epoch 18/30
1273/1273 [==============================] - 7s 5ms/step - loss: 0.0821
Epoch 19/30
1273/1273 [==============================] - 7s 6ms/step - loss: 0.0812
Epoch 20/30
1273/1273 [==============================] - 6s 5ms/step - loss: 0.0802
Epoch 21/30
1273/1273 [==============================] - 6s 5ms/step - loss: 0.0794
Epoch 22/30
1273/1273 [==============================] - 7s 5ms/step - loss: 0.0786
Epoch 23/30
1273/1273 [==============================] - 7s 6ms/step - loss: 0.0777
Epoch 24/30
1273/1273 [==============================] - 7s 6ms/step - loss: 0.0770
Epoch 25/30
1273/1273 [==============================] - 6s 4ms/step - loss: 0.0761
Epoch 26/30
1273/1273 [==============================] - 7s 5ms/step - loss: 0.0756
Epoch 27/30
1273/1273 [==============================] - 7s 5ms/step - loss: 0.0748
Epoch 28/30
1273/1273 [==============================] - 8s 6ms/step - loss: 0.0742
Epoch 29/30
1273/1273 [==============================] - 7s 5ms/step - loss: 0.0735
Epoch 30/30
1273/1273 [==============================] - 7s 6ms/step - loss: 0.0729
```

Loading [MathJax]/extensions/Safe.js

Out[16]:
```
<keras.src.callbacks.History at 0x21075134d30>
```

Evaluating the model to determine loss on the test data:

In [17]:
```python
model.evaluate([user_test[:, u_s:], item_test[:, i_s:]], y_test)
```

```
319/319 [==============================] - 1s 2ms/step - loss: 0.0817
```

Out[17]:
```
0.08169607073068619
```

It is comparable to the training loss indicating the model has not substantially overfit the training data.

# 4 - Predictions:

Using the model to make predictions in a number of circumstances.

- Predictions for a new user:

In [18]:
```python
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]])
```

The new user enjoys movies from the adventure, fantasy genres.

The top-rated movies for the new user (using a set of movie/item vectors, item_vecs that have a vector for each movie in the training/test set. This is matched with the new user vector above and the scaled vectors are used to predict ratings for all the movies):

In [19]:
```python
# 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
Target.inverse_transform(y_p)
```

Loading [MathJax]/extensions/Safe.js

```
# sort the results, highest prediction first
sorted_index = np.argsort(-y_pu,axis=0).reshape(-1).tolist()  #negate to get largest rat
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)
```

```
27/27 [==============================] - 0s 3ms/step
```

Out[19]:

| y_p | movie id | rating ave | title | genres |
|---|---|---|---|---|
| 4.2 | 8368 | 3.9 | Harry Potter and the Prisoner of Azkaban (2004) | Adventure\|Fantasy |
| 4.2 | 5952 | 4 | Lord of the Rings: The Two Towers, The (2002) | Adventure\|Fantasy |
| 4.1 | 40815 | 3.8 | Harry Potter and the Goblet of Fire (2005) | Adventure\|Fantasy\|Thriller |
| 4.1 | 59387 | 4 | Fall, The (2006) | Adventure\|Drama\|Fantasy |
| 4.1 | 54001 | 3.9 | Harry Potter and the Order of the Phoenix (2007) | Adventure\|Drama\|Fantasy |
| 4.1 | 4993 | 4.1 | Lord of the Rings: The Fellowship of the Ring, The (2001) | Adventure\|Fantasy |
| 4.1 | 98809 | 3.8 | Hobbit: An Unexpected Journey, The (2012) | Adventure\|Fantasy |
| 4.1 | 6539 | 3.8 | Pirates of the Caribbean: The Curse of the Black Pearl (2003) | Action\|Adventure\|Comedy\|Fantasy |
| 4.1 | 7153 | 4.1 | Lord of the Rings: The Return of the King, The (2003) | Action\|Adventure\|Drama\|Fantasy |
| 4.1 | 5816 | 3.6 | Harry Potter and the Chamber of Secrets (2002) | Adventure\|Fantasy |

- Predictions for an existing user: The predictions are for "user 2", one of the users in the data set.

In [20]:

```
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 rat
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,
```

```
27/27 [==============================] - 0s 2ms/step
```

Loading [MathJax]/extensions/Safe.js

Out[20]:

| y_p | y | user | user genre ave | movie rating ave | movie id | title | genres |
|---|---|---|---|---|---|---|---|
| 4.4 | 5.0 | 2 | [4.0] | 4.3 | 80906 | Inside Job (2010) | Documentary |
| 4.3 | 3.5 | 2 | [4.0,4.0] | 3.9 | 99114 | Django Unchained (2012) | Action\|Drama |
| 4.2 | 4.5 | 2 | [4.0,4.0] | 4.1 | 68157 | Inglourious Basterds (2009) | Action\|Drama |
| 4.2 | 4.0 | 2 | [4.0,4.1,3.9] | 4.0 | 6874 | Kill Bill: Vol. 1 (2003) | Action\|Crime\|Thriller |
| 4.1 | 4.5 | 2 | [4.0,4.1,4.0] | 4.2 | 58559 | Dark Knight, The (2008) | Action\|Crime\|Drama |
| 4.1 | 3.5 | 2 | [4.0,4.2,4.1] | 4.0 | 91529 | Dark Knight Rises, The (2012) | Action\|Adventure\|Crime |
| 4.1 | 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 | 5.0 | 2 | [4.0,4.1,4.0] | 3.9 | 106782 | Wolf of Wall Street, The (2013) | Comedy\|Crime\|Drama |
| 4.0 | 5.0 | 2 | [4.0,4.2,3.9,3.9] | 3.8 | 122882 | Mad Max: Fury Road (2015) | Action\|Adventure\|Sci-Fi\|Thriller |
| 4.0 | 3.5 | 2 | [4.0,4.1,4.0,3.9] | 3.8 | 8798 | Collateral (2004) | Action\|Crime\|Drama\|Thriller |
| 4.0 | 4.5 | 2 | [4.1,4.0,3.9] | 4.0 | 80489 | Town, The (2010) | Crime\|Drama\|Thriller |
| 4.0 | 3.0 | 2 | [3.9] | 4.0 | 109487 | Interstellar (2014) | Sci-Fi |
| 3.9 | 4.0 | 2 | [4.1,4.0,3.9] | 4.3 | 48516 | Departed, The (2006) | Crime\|Drama\|Thriller |
| 3.8 | 4.0 | 2 | [4.0] | 4.0 | 112552 | Whiplash (2014) | Drama |
| 3.8 | 5.0 | 2 | [4.0] | 3.6 | 60756 | Step Brothers (2008) | Comedy |
| 3.8 | 5.0 | 2 | [4.0] | 3.7 | 89774 | Warrior (2011) | Drama |
| 3.8 | 3.5 | 2 | [4.0,3.9,3.9] | 3.9 | 115713 | Ex Machina (2015) | Drama\|Sci-Fi\|Thriller |
| 3.7 | 4.0 | 2 | [4.0,4.0,3.9] | 4.0 | 74458 | Shutter Island (2010) | Drama\|Mystery\|Thriller |
| 3.6 | 3.0 | 2 | [4.0,4.0,3.0] | 3.9 | 71535 | Zombieland (2009) | Action\|Comedy\|Horror |
| 3.6 | 2.5 | 2 | [4.0,3.9] | 3.5 | 91658 | Girl with the Dragon Tattoo, The (2011) | Drama\|Thriller |
| 3.5 | 4.0 | 2 | [4.0,4.0] | 3.2 | 46970 | Talladega Nights: The Ballad of Ricky Bobby (2006) | Action\|Comedy |
| 3.1 | 3.0 | 2 | [4.0,4.0] | 4.0 | 77455 | Exit Through the Gift Shop (2010) | Comedy\|Documentary |

The model prediction is generally within 1 of the actual rating though it is not a very accurate predictor of how a user rates specific movies. This is especially true if the user rating is significantly different than the user's genre average.

## Finding Similar Items:

The neural network above produces two feature vectors, a user feature vector $v_u$, and a movie feature vector, $v_m$. These are 32 entry vectors whose values are difficult to interpret. However, similar items will have similar vectors. This information can be used to make recommendations. For example, if a user has rated "Toy Story 3" highly, one could recommend similar movies by selecting movies with similar movie

Loading [MathJax]/extensions/Safe.js

A similarity measure is the squared distance between the two vectors $ \mathbf{v_m^{(k)}}$ and $\mathbf{v_m^{(i)}}$ : $$\left\Vert \mathbf{v_m^{(k)}} - \mathbf{v_m^{(i)}} \right\Vert^2 = \sum_{l=1}^{n} (v_{m_l}^{(k)} - v_{m_l}^{(i)})^2\tag{1}$$

A function to compute the square distance:

In [21]:
```
def sq_dist(a,b):
    """
    Returns the squared distance between two vectors
    Args:
      a (ndarray (n,)): vector with n features
      b (ndarray (n,)): vector with n features
    Returns:
      d (float) : distance
    """

    sq_dis = np.square(a - b)
    d = np.sum(sq_dis)

    return d
```

In [22]:
```
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]:
```
# Public tests
test_sq_dist(sq_dist)
```

```
All tests passed!
```

A matrix of distances between movies can be computed once when the model is trained and then reused for new recommendations without retraining. The first step, once a model is trained, is to obtain the movie feature vector, $v_m$, for each of the movies. To do this,the trained item_NN is used and build a small model to run the movie vectors through it to generate $v_m$.

In [24]:
```
input_item_m = tf.keras.layers.Input(shape=(num_item_features))    # input layer
vm_m = item_NN(input_item_m)                                       # using the trained i
vm_m = tf.linalg.l2_normalize(vm_m, axis=1)                        # incorporate normali
model_m = tf.keras.Model(input_item_m, vm_m)
model_m.summary()
```

Loading [MathJax]/extensions/Safe.js

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_3 (InputLayer)        [(None, 16)]              0         
                                                                 
 sequential_1 (Sequential)   (None, 32)                41376     
                                                                 
 tf.math.l2_normalize_2 (TF  (None, 32)                0         
 OpLambda)                                                       
                                                                 
=================================================================
Total params: 41376 (161.62 KB)
Trainable params: 41376 (161.62 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

After a movie model was built --> creating a set of movie feature vectors by using the model to predict using a set of item/movie vectors as input. item_vecs is a set of all of the movie vectors. It must be scaled to use with the trained model. The result of the prediction is a 32 entry feature vector for each movie.

In [25]:
```python
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}")
```

```
27/27 [==============================] - 0s 1ms/step
size of all predicted movie feature vectors: (847, 32)
```

Computing a matrix of the squared distance between each movie feature vector and all other movie feature vectors: numpy masked arrays is used to avoid selecting the same movie.

In [26]:
```python
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
```

Loading [MathJax]/extensions/Safe.js

Out[26]:

| 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 | Sweetest Thing, The (2002) | |
| Down to Earth (2001) | Comedy\|Fantasy\|Romance | Bewitched (2005) | Comedy |
| Mexican, The (2001) | Action\|Comedy | Rush Hour 2 (2001) | |
| 15 Minutes (2001) | Thriller | Panic Room (2002) | |
| Enemy at the Gates (2001) | Drama | Aviator, The (2004) | |
| Heartbreakers (2001) | Comedy\|Crime\|Romance | Fun with Dick and Jane (2005) | Comec |
| Spy Kids (2001) | Action\|Adventure\|Children\|Comedy | Scooby-Doo (2002) | Action\|Adventure |
| Along Came a Spider (2001) | Action\|Crime\|Mystery\|Thriller | Insomnia (2002) | Action\|Cri |
| Blow (2001) | Crime\|Drama | 25th Hour (2002) | |
| Bridget Jones's Diary (2001) | Comedy\|Drama\|Romance | Punch-Drunk Love (2002) | Comed |
| Joe Dirt (2001) | Adventure\|Comedy\|Mystery\|Romance | Elektra (2005) | Adventure\|Comedy |
| Crocodile Dundee in Los Angeles (2001) | Comedy\|Drama | Nacho Libre (2006) | |
| Mummy Returns, The (2001) | Action\|Adventure\|Comedy\|Thriller | Men in Black II (a.k.a. MIIB) (a.k.a. MIB 2) (2002) | Action\|Adventu |
| Knight's Tale, A (2001) | Action\|Comedy\|Romance | 13 Going on 30 (2004) | Action\| |
| Shrek (2001) | Adventure\|Animation\|Children\|Comedy\|Fantasy\|Romance | Enchanted (2007) | Adventure\|Animation\|Children\|Comedy |
| Moulin Rouge (2001) | Drama\|Romance | Walk to Remember, A (2002) | |

Loading [MathJax]/extensions/Safe.js

| movie1 | genres | movie2 | |
|---|---|---|---|
| Pearl Harbor (2001) | Action\|Drama\|Romance | Bridget Jones: The Edge of Reason (2004) | Actio |
| Animal, The (2001) | Comedy | Dumb and Dumberer: When Harry Met Lloyd (2003) | |
| Evolution (2001) | Comedy\|Sci-Fi | Transporter 2 (2005) | |
| Swordfish (2001) | Action\|Crime\|Drama | Spy Game (2001) | A |
| Atlantis: The Lost Empire (2001) | Adventure\|Animation\|Children\|Fantasy | Enchanted (2007) | Adventure\|Animatio |
| Lara Croft: Tomb Raider (2001) | Action\|Adventure | Jurassic Park III (2001) | |
| Dr. Dolittle 2 (2001) | Comedy | Legally Blonde 2: Red, White & Blonde (2003) | |
| Fast and the Furious, The (2001) | Action\|Crime\|Thriller | Once Upon a Time in Mexico (2003) | A |
| A.I. Artificial Intelligence (2001) | Adventure\|Drama\|Sci-Fi | Marley & Me (2008) | Adve |
| Cats & Dogs (2001) | Children\|Comedy | Shark Tale (2004) | |
| Scary Movie 2 (2001) | Comedy | Orange County (2002) | |
| Final Fantasy: The Spirits Within (2001) | Adventure\|Animation\|Fantasy\|Sci-Fi | Tropic Thunder (2008) | Adventure\|Anima |
| Legally Blonde (2001) | Comedy\|Romance | Serendipity (2001) | |
| Score, The (2001) | Action\|Drama | We Were Soldiers (2002) | |
| Jurassic Park III (2001) | Action\|Adventure\|Sci-Fi\|Thriller | Lara Croft: Tomb Raider (2001) | Action\|Adve |
| America's Sweethearts (2001) | Comedy\|Romance | Maid in Manhattan (2002) | |
| Ghost World (2001) | Comedy\|Drama | Station Agent, The (2003) | |
| Planet of the | Action\|Adventure\|Drama\|Sci-Fi | King Arthur (2004) | Action\|Adve |

Loading [MathJax]/extensions/Safe.js

| movie1 | genres | movie2 | |
|---|---|---|---|
| Princess Diaries, The (2001) | Children\|Comedy\|Romance | Monster's Ball (2001) | Children\| |
| Rush Hour 2 (2001) | Action\|Comedy | Mexican, The (2001) | |
| American Pie 2 (2001) | Comedy | Rat Race (2001) | |
| Others, The (2001) | Drama\|Horror\|Mystery\|Thriller | Dogville (2003) | Drama\|Hor |
| Rat Race (2001) | Comedy | American Pie 2 (2001) | |
| Jay and Silent Bob Strike Back (2001) | Adventure\|Comedy | EuroTrip (2004) | A |
| Training Day (2001) | Crime\|Drama\|Thriller | Frailty (2001) | Cr |
| Zoolander (2001) | Comedy | Old School (2003) | |
| Serendipity (2001) | Comedy\|Romance | Legally Blonde (2001) | |
| Mulholland Drive (2001) | Crime\|Drama\|Mystery\|Thriller | Dogville (2003) | Crime\|Dra |
| From Hell (2001) | Crime\|Horror\|Mystery\|Thriller | Identity (2003) | Crime\|Hor |
| Waking Life (2001) | Animation\|Drama\|Fantasy | Bubba Ho-tep (2002) | Animat |
| K-PAX (2001) | Drama\|Fantasy\|Mystery\|Sci-Fi | 21 Grams (2003) | Drama\|Fan |

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

```
In [ ]:
```

Loading [MathJax]/extensions/Safe.js