Content Based Filtering by hand

This lab illustrates how to implement a content based filter using low level Tensorflow operations.
The code here follows the technique explained in Module 2 of Recommendation Engines: Content Based Filtering.

In [ ]:
```
!pip install tensorflow==2.5
```

Make sure to restart your kernel to ensure this change has taken place.

In [1]:
```
import numpy as np
import tensorflow as tf

print(tf.__version__)
```

```
2021-11-04 18:23:35.530612: I tensorflow/stream_executor/platform/default/dso_lo
ader.cc:53] Successfully opened dynamic library libcudart.so.11.0
2.5.0
```

To start, we'll create our list of users, movies and features. While the users and movies represent elements in our database, for a content-based filtering method the features of the movies are likely hand-engineered and rely on domain knowledge to provide the best embedding space. Here we use the categories of Action, Sci-Fi, Comedy, Cartoon, and Drama to describe our movies (and thus our users).

In this example, we will assume our database consists of four users and six movies, listed below.

In [2]:
```
users = ['Ryan', 'Danielle',  'Vijay', 'Chris']
movies = ['Star Wars', 'The Dark Knight', 'Shrek', 'The Incredibles', 'Bleu', 'M
features = ['Action', 'Sci-Fi', 'Comedy', 'Cartoon', 'Drama']

num_users = len(users)
num_movies = len(movies)
num_feats = len(features)
num_recommendations = 2
```

Initialize our users, movie ratings and features

We'll need to enter the user's movie ratings and the k-hot encoded movie features matrix. Each row of the users_movies matrix represents a single user's rating (from 1 to 10) for each movie. A zero indicates that the user has not seen/rated that movie. The movies_feats matrix contains the features for each of the given movies. Each row represents one of the six movies, the columns represent the five categories. A one indicates that a movie fits within a given genre/category.

In [3]:
```
# each row represents a user's rating for the different movies
users_movies = tf.constant([
                [4,  6,  8,  0, 0, 0],
                [0,  0, 10,  0, 8, 3],
```

```
                [0,  6,  0,  0, 3, 7],
                [10, 9,  0,  5, 0, 2]],dtype=tf.float32)

# features of the movies one-hot encoded
# e.g. columns could represent ['Action', 'Sci-Fi', 'Comedy', 'Cartoon', 'Drama'
movies_feats = tf.constant([
                [1, 1, 0, 0, 1],
                [1, 1, 0, 0, 0],
                [0, 0, 1, 1, 0],
                [1, 0, 1, 1, 0],
                [0, 0, 0, 0, 1],
                [1, 0, 0, 0, 1]],dtype=tf.float32)
```

```
2021-11-04 18:24:47.019280: W tensorflow/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.
1: cannot open shared object file: No such file or directory; LD_LIBRARY_PATH: /
usr/local/cuda/lib64:/usr/local/nccl2/lib:/usr/local/cuda/extras/CUPTI/lib64
2021-11-04 18:24:47.019330: W tensorflow/stream_executor/cuda/cuda_driver.cc:32
6] failed call to cuInit: UNKNOWN ERROR (303)
2021-11-04 18:24:47.019359: I tensorflow/stream_executor/cuda/cuda_diagnostics.c
c:156] kernel driver does not appear to be running on this host (tensorflow-2-6-
20211104-141724): /proc/driver/nvidia/version does not exist
2021-11-04 18:24:47.021955: I tensorflow/core/platform/cpu_feature_guard.cc:142]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (one
DNN) to use the following CPU instructions in performance-critical operations:
AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate comp
iler flags.
```

Computing the user feature matrix

We will compute the user feature matrix; that is, a matrix containing each user's embedding in the five-dimensional feature space.

In [4]:
```
users_feats = tf.matmul(users_movies,movies_feats)
users_feats
```

Out[4]:
```
<tf.Tensor: shape=(4, 5), dtype=float32, numpy=
array([[10., 10.,  8.,  8.,  4.],
       [ 3.,  0., 10., 10., 11.],
       [13.,  6.,  0.,  0., 10.],
       [26., 19.,  5.,  5., 12.]], dtype=float32)>
```

Next we normalize each user feature vector to sum to 1. Normalizing isn't strictly neccesary, but it makes it so that rating magnitudes will be comparable between users.

In [5]:
```
users_feats = users_feats/tf.reduce_sum(users_feats,axis=1,keepdims=True)
users_feats
```

Out[5]:
```
<tf.Tensor: shape=(4, 5), dtype=float32, numpy=
array([[0.25      , 0.25      , 0.2       , 0.2       , 0.1       ],
       [0.0882353 , 0.        , 0.29411766, 0.29411766, 0.32352942],
       [0.44827586, 0.20689656, 0.        , 0.        , 0.3448276 ],
       [0.3880597 , 0.2835821 , 0.07462686, 0.07462686, 0.17910448]],
      dtype=float32)>
```

Ranking feature relevance for each user

We can use the users_feats computed above to represent the relative importance of each movie category for each user.

In [6]:
```
top_users_features = tf.nn.top_k(users_feats, num_feats)[1]
top_users_features
```

Out[6]:
```
<tf.Tensor: shape=(4, 5), dtype=int32, numpy=
array([[0, 1, 2, 3, 4],
       [4, 2, 3, 0, 1],
       [0, 4, 1, 2, 3],
       [0, 1, 4, 2, 3]], dtype=int32)>
```

In [7]:
```
for i in range(num_users):
    feature_names = [features[int(index)] for index in top_users_features[i]]
    print('{}: {}'.format(users[i],feature_names))
```

```
Ryan: ['Action', 'Sci-Fi', 'Comedy', 'Cartoon', 'Drama']
Danielle: ['Drama', 'Comedy', 'Cartoon', 'Action', 'Sci-Fi']
Vijay: ['Action', 'Drama', 'Sci-Fi', 'Comedy', 'Cartoon']
Chris: ['Action', 'Sci-Fi', 'Drama', 'Comedy', 'Cartoon']
```

Determining movie recommendations.

We'll now use the `users_feats` tensor we computed above to determine the movie ratings and recommendations for each user.

To compute the projected ratings for each movie, we compute the similarity measure between the user's feature vector and the corresponding movie feature vector.

We will use the dot product as our similarity measure. In essence, this is a weighted movie average for each user.

In [8]:
```
users_ratings = tf.matmul(users_feats,tf.transpose(movies_feats))
users_ratings
```

Out[8]:
```
<tf.Tensor: shape=(4, 6), dtype=float32, numpy=
array([[0.6       , 0.5       , 0.4       , 0.65      , 0.1       ,
        0.35      ],
       [0.4117647 , 0.0882353 , 0.5882353 , 0.67647064, 0.32352942,
        0.4117647 ],
       [1.        , 0.6551724 , 0.        , 0.44827586, 0.3448276 ,
        0.79310346],
       [0.8507463 , 0.6716418 , 0.14925373, 0.53731346, 0.17910448,
        0.5671642 ]], dtype=float32)>
```

The computation above finds the similarity measure between each user and each movie in our database. To focus only on the ratings for new movies, we apply a mask to the all_users_ratings matrix.

If a user has already rated a movie, we ignore that rating. This way, we only focus on ratings for previously unseen/unrated movies.

In [9]:
```
users_ratings_new = tf.where(tf.equal(users_movies, tf.zeros_like(users_movies)),
                             users_ratings,
```

```
                                    tf.zeros_like(tf.cast(users_movies, tf.float32
users_ratings_new
```

Out[9]:
```
<tf.Tensor: shape=(4, 6), dtype=float32, numpy=
array([[0.        , 0.        , 0.        , 0.65      , 0.1       ,
        0.35      ],
       [0.4117647 , 0.0882353 , 0.        , 0.67647064, 0.        ,
        0.        ],
       [1.        , 0.        , 0.        , 0.44827586, 0.        ,
        0.        ],
       [0.        , 0.        , 0.14925373, 0.        , 0.17910448,
        0.        ]], dtype=float32)>
```

Finally let's grab and print out the top 2 rated movies for each user

In [10]:
```
top_movies = tf.nn.top_k(users_ratings_new, num_recommendations)[1]
top_movies
```

Out[10]:
```
<tf.Tensor: shape=(4, 2), dtype=int32, numpy=
array([[3, 5],
       [3, 0],
       [0, 3],
       [4, 2]], dtype=int32)>
```

In [11]:
```
for i in range(num_users):
    movie_names = [movies[index] for index in top_movies[i]]
    print('{}: {}'.format(users[i],movie_names))
```

```
Ryan: ['The Incredibles', 'Memento']
Danielle: ['The Incredibles', 'Star Wars']
Vijay: ['Star Wars', 'The Incredibles']
Chris: ['Bleu', 'Shrek']
```

In [ ]:
```

```