Sistemas de recomendación

Carga de datos de Movie Lens

In [1]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import preprocessing, neighbors
import pandas as pd
```

In [5]:
```
df = pd.read_csv("../../Data-Sets/datasets/ml-100k/u.data.csv", sep = "\t", header=None)
df.head()
```

Out[5]:

	0	1	2	3
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

In [6]:
```
df.shape
```

Out[6]:
```
(100000, 4)
```

In [7]:
```
df.columns
```

Out[7]:
```
Int64Index([0, 1, 2, 3], dtype='int64')
```

In [9]:
```
df.columns = ["UserID", "ItemID", "Rating", "Time_Stamp"]
df.head()
```

Out[9]:

	UserID	ItemID	Rating	Time_Stamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

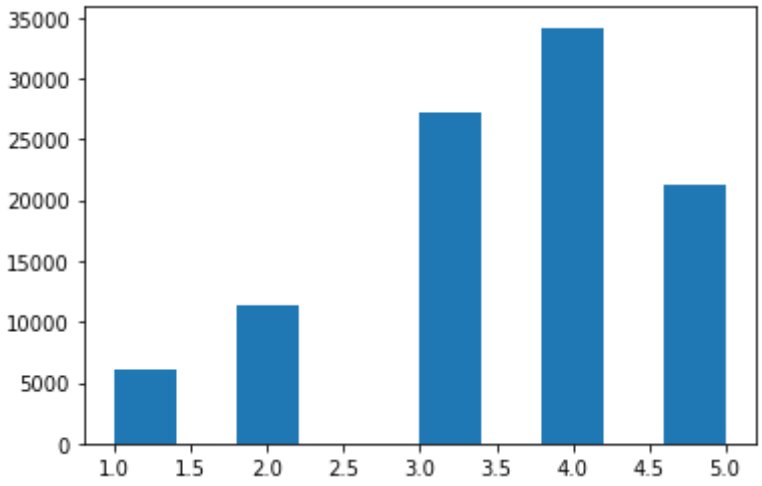
Análisis exploratorio de los ítems

In [10]:
```
import matplotlib.pyplot as plt
```

In [11]:
```
plt.hist(df.Rating)
```

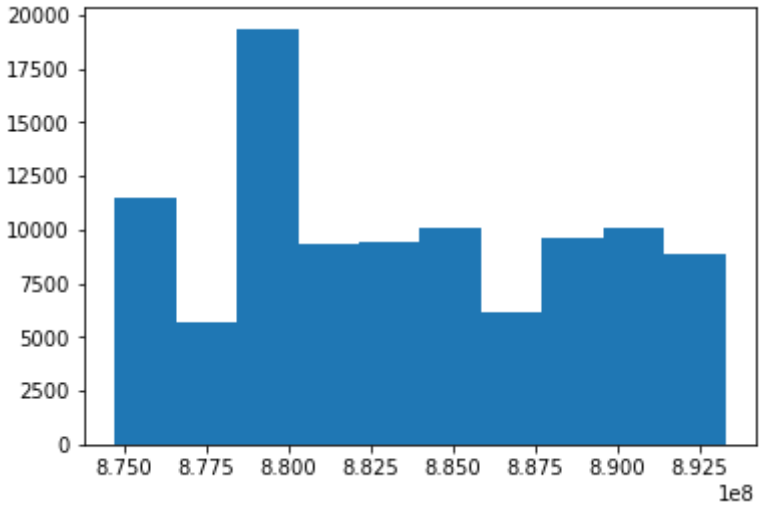
Out[11]:
```
(array([ 6110.,     0., 11370.,     0.,     0., 27145.,     0., 34174.,
```

```
           0., 21201.]),
 array([1. , 1.4, 1.8, 2.2, 2.6, 3. , 3.4, 3.8, 4.2, 4.6, 5. ]),
 <BarContainer object of 10 artists>)
```



In [13]:
```
plt.hist(df.Time_Stamp)
```

Out[13]:
```
(array([11459.,  5724., 19359.,  9315.,  9396., 10083.,  6175.,  9603.,
        10048.,  8838.]),
 array([8.74724710e+08, 8.76580903e+08, 8.78437096e+08, 8.80293288e+08,
        8.82149481e+08, 8.84005674e+08, 8.85861867e+08, 8.87718060e+08,
        8.89574252e+08, 8.91430445e+08, 8.93286638e+08]),
 <BarContainer object of 10 artists>)
```

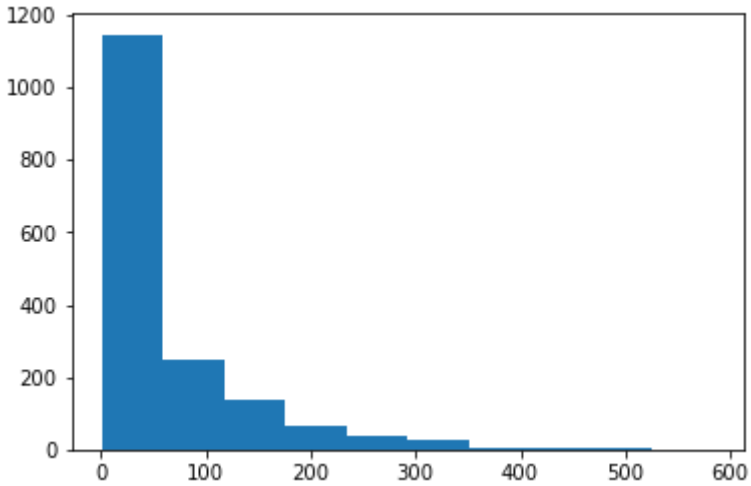


In [14]:
```
df.groupby(["Rating"])["UserID"].count()
```

Out[14]:
```
Rating
1     6110
2    11370
3    27145
4    34174
5    21201
Name: UserID, dtype: int64
```

In [26]:
```
plt.hist(df.groupby(["ItemID"])["ItemID"].count())
```

Out[26]:
```
(array([1.146e+03, 2.490e+02, 1.390e+02, 6.700e+01, 3.800e+01, 2.400e+01,
        7.000e+00, 5.000e+00, 6.000e+00, 1.000e+00]),
 array([  1. ,  59.2, 117.4, 175.6, 233.8, 292. , 350.2, 408.4, 466.6,
        524.8, 583. ]),
 <BarContainer object of 10 artists>)
```



El conjunto de películas de la 0 a la 50 se lleva la mayoría de visualizaciones.

Representación en forma matricial

In [27]:
```
n_users = df.UserID.unique().shape[0]
n_users
```

Out[27]:
```
943
```

In [29]:
```
n_items = df.ItemID.unique().shape[0]
n_items
```

Out[29]:
```
1682
```

In [30]:
```
ratings = np.zeros((n_users, n_items)) #Filas x Columnas
#Spars matrix, matriz esparceada
```

In [32]:
```
#Por cada fila del data frame original.
for row in df.itertuples():
    ratings[row[1]-1, row[2]-1] = row[3]
```

In [33]:
```
type(ratings)
```

Out[33]:
```
numpy.ndarray
```

In [35]:
```
ratings.shape #943 USUARIOS Y 1682 PELIS
```

Out[35]:
```
(943, 1682)
```

In [36]:
```
ratings
```

Out[36]:
```
array([[5., 3., 4., ..., 0., 0., 0.],
       [4., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [5., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 5., 0., ..., 0., 0., 0.]])
```

In [43]:
```
sparsity = float(len(ratings.nonzero()[0]))
```

```
sparsity /= (ratings.shape[0]*ratings.shape[1])
sparsity *= 100
print("Coeficiente de sparseidad: {:4.2f}%".format(sparsity))
```

```
Coeficiente de sparseidad: 6.30%
```

6.30% de esa matriz no está vacía. No se conoce absolutamente nada de la combinación Usuario, Película del aproximadamente el 94%, por lo que hay margen más que de sobra para recomendar películas que los usuarios no hallan visto.

Crear conjuntos de entrenamiento y validación

In [49]:
```
from sklearn.model_selection import train_test_split
```

In [50]:
```
ratings_train, ratings_test = train_test_split(ratings, test_size=0.3, random_state=42)
```

In [51]:
```
ratings_train.shape #660 usuarios y 1682 pelis
```

Out[51]:
```
(660, 1682)
```

In [52]:
```
ratings_test.shape #283 usuarios para proponer una recomendación.
```

Out[52]:
```
(283, 1682)
```

Filtro colaborativo basado en Usuarios

Mapa de Ruta

- Matriz de similaridad entre los usuarios. (distancia del coseno) **Nota**: La distancia y la similaridad son conceptos casi opuestos, porque una similaridad alta significa que son parecidos, mientras que un una distancia alta son diferentes. La distancia coseno siempre dá valores entre 0 y 1. El problema está que un coseno igual a 0 es que la distancia es nula, y la similaridad igual a 0 es que son totalmente opuestos.
- Predecir la valoración desconocida de un ítem *i* para un usuario activo *u* basándonos en la suma ponderada de todas las valoraciones del resto de usuarios para dicho ítem.
- Recomendaremos los nuevos ítems a los usuarios según lo establecido en los pasos anteriores.

In [53]:
```
import sklearn
```

In [54]:
```
sim_matrix = 1-sklearn.metrics.pairwise.cosine_distances(ratings_train)
#El uno menos está para transformar los ítems en 0. A menor distancia en unos y los ítems en
#para que sea equivalente a las similaridades.
```

In [55]:
```
type(sim_matrix)
```

Out[55]:
```
numpy.ndarray
```

In [56]:
```
sim_matrix.shape
```

Out[56]:
```
(660, 660)
```

In [57]:
```
sim_matrix
```

Out[57]:
```
array([[1.        , 0.17448349, 0.18009754, ..., 0.13331459, 0.17695593,
        0.21882481],
       [0.17448349, 1.        , 0.07993097, ..., 0.07175808, 0.09552622,
        0.05512655],
       [0.18009754, 0.07993097, 1.        , ..., 0.0191736 , 0.02233385,
        0.10310785],
       ...,
       [0.13331459, 0.07175808, 0.0191736 , ..., 1.        , 0.04853428,
        0.05142508],
       [0.17695593, 0.09552622, 0.02233385, ..., 0.04853428, 1.        ,
        0.1198022 ],
       [0.21882481, 0.05512655, 0.10310785, ..., 0.05142508, 0.1198022 ,
        1.        ]])
```

Esto crea una matriz con diagonal 1, porque un usuario siempre es similar a sí mismo, por ende la máxima similaridad es igual a 1 y luego en los otros valores tenemos las distancias de ese mismo contra el segundo, contra el tercero, contra el cuarto y así suscesivamente. **Es una matriz simétrica**. Las distancias las calcula tomando cada una de las filas creadas en _ratings*train* (o sea los 660 usuarios) y luego calcula la distancia coseno total de dicha fila, esa distancia que corresponde a cada usuario la contrasta contra los otros 659 usuarios y crea dicha matriz. Dicha matriz se debe interpretar como **porcentaje de parercido de los gustos** por ejemplo el usuario 1 y el dos tienen un parecido del 17% de gustos.

In [61]:
```
users_predictions = sim_matrix.dot(ratings_train) / np.array([np.abs(sim_matrix).sum(axis=1)
# Hacemos las predicciones, .dot es producto matricial de sim_matrix y ratings_train. O sea c
# de gustos por la matriz de entrenamiento, y el resultado dividirlo por la sumatoria de todo
# de la misma fila, por eso el axis=1 y el abs, es valor absoluto, y T es para que haga la tr
# y la división se haga de manera correcta.
```

In [62]:
```
users_predictions
```

Out[62]:
```
array([[2.10259747e+00, 5.86975978e-01, 3.40264192e-01, ...,
        0.00000000e+00, 7.33611460e-03, 6.04379414e-03],
       [1.40999723e+00, 2.91863934e-01, 2.68085289e-01, ...,
        0.00000000e+00, 3.50378592e-03, 2.32963985e-03],
       [1.69014833e+00, 3.13648440e-01, 3.26127887e-01, ...,
        0.00000000e+00, 3.25391767e-03, 1.77210119e-03],
       ...,
       [1.73393747e+00, 4.06719333e-01, 3.21166908e-01, ...,
        0.00000000e+00, 2.71269625e-03, 9.00511411e-03],
       [2.34361031e+00, 8.10544770e-01, 4.73941025e-01, ...,
        0.00000000e+00, 1.01130066e-02, 9.66427605e-03],
       [2.36796969e+00, 5.98146138e-01, 3.85569804e-01, ...,
        0.00000000e+00, 6.39996638e-03, 5.37442746e-03]])
```

In [63]:
```
users_predictions.shape
```

Out[63]:
```
(660, 1682)
```

Cuando realizamos el producto, entonces volvemos a tener las 1682 películas y la valoración que cada usuario le daría a cada una de las 1682 películas.

Comprobación del error cuadrático-performance

Predicción para el conjunto de testing

Se calculará el error cuadrático medio, sólo para los valores conocidos.

In [64]:
```
from sklearn.metrics import mean_squared_error
```

In [190…
```
def get_mse (preds, actuals):
    if preds.shape[0] != actuals.shape[0]: # Si el número de filas de las pred  es dist al n'
        actuals = actuals.T # Transpongo los datos actuales.
    preds = preds[actuals.nonzero()].flatten()#lo pasamos a vector
    actuals = actuals[actuals.nonzero()].flatten()#lo pasamos a vector
    return mean_squared_error(preds, actuals) #Calculamos el error y lo pasamos
```

In [67]:
```
get_mse(users_predictions, ratings_train)
```

Out[67]:
```
7.878218313143215
```

Un error cuadrático medio de 7.87 porciento en el conjunto de training.

Predicción para el conjunto de testing

In [70]:
```
sim_matrix_test = 1-sklearn.metrics.pairwise.cosine_distances(ratings_test)
```

In [71]:
```
users_predictions_test = sim_matrix_test.dot(ratings_test) / np.array([np.abs(sim_matrix_test
```

In [72]:
```
get_mse(users_predictions_test, ratings_test)
```

Out[72]:
```
7.51355110112698
```

Un error cuadrático medio de 7.51 porciento en el conjunto de testing. Se concluye que este filtro colaborativo basándonos en los usuarios puede ser una opción bastante buena para crear las recomendaciones.

Filtro colaborativo basado en los KNN

In [92]:
```
from sklearn.neighbors import NearestNeighbors
```

In [93]:
```
k = 10
```

In [94]:
```
#(1) Llamar correctamente a NearestNeighbors:
neighbors = NearestNeighbors(n_neighbors=k, metric = 'cosine')
```

In [95]:
```
neighbors.fit(ratings_train)
```

Out[95]:
```
NearestNeighbors(metric='cosine', n_neighbors=10)
```

In [96]:
```
top_k_distances, top_k_users = neighbors.kneighbors(ratings_train, return_distance=True)
```

In [97]:
```
#(2) Transformar las distancias en similitudes para ponderar, para ello definí top_k_sim como
top_k_sim = 1 - top_k_distances
```

In [98]:
```
#(3) Calcular las predicciones del conjunto de entrenamiento a partir de las similitudes y no
user_predicts_k = np.zeros(ratings_train.shape)
```

```
for i in range(ratings_train.shape[0]):
    user_predicts_k[i,:] = top_k_sim[i].dot(ratings_train[top_k_users[i]]) / np.sum(top_k_sir
```

In [99]:
```
#(4) Ahora el error cuadrático medio ya es más razonable:
get_mse(user_predicts_k, ratings_train)
```

Out[99]:
```
2.756446689065506
```

In [100…
```
#(5) Usar el modelo KKN entrenado con el conjunto de entrenamiento para obtener los k vecinos
top_k_distances_test, top_k_users_test = neighbors.kneighbors(ratings_test)
```

In [101…
```
#(6) Definir las similitudes asociadas
top_k_sim_test = 1 - top_k_distances_test
```

In [102…
```
#(7) Obtener las predicciones del conjunto de test:
user_predicts_test_k = np.zeros(ratings_test.shape)
for i in range(ratings_test.shape[0]):
    user_predicts_test_k[i,:] = top_k_sim_test[i].dot(ratings_train[top_k_users_test[i]]) / r
```

In [103…
```
#(8) Obtener el error del conjunto de test:
get_mse(user_predicts_test_k, ratings_test)
```

Out[103…
```
4.102396284374267
```

Filtro colaborativo basado en los KNN

Training

In [123…
```
n_movies = ratings_train.shape[1]
n_movies
```

Out[123…
```
1682
```

In [124…
```
k = n_movies # Los k vecinios serán el total de las películas.
```

In [125…
```
neighbors = NearestNeighbors(n_neighbors=k, metric = 'cosine')
```

In [126…
```
neighbors.fit(ratings_train.T)#Se transpone para que las películas queden en filas.
```

Out[126…
```
NearestNeighbors(metric='cosine', n_neighbors=1682)
```

In [127…
```
top_k_distances, top_k_items = neighbors.kneighbors(ratings_train.T, return_distance=True)
```

In [128…
```
top_k_distances.shape
```

Out[128…
```
(1682, 1682)
```

In [129…
```
top_k_items.shape
```

Out[129… (1682, 1682)

In [130…
```
item_preds = ratings_train.dot(top_k_distances) / np.array([np.abs(top_k_distances).sum(axis=
```

In [131…
```
item_preds.shape
```

Out[131… (660, 1682)

In [132…
```
item_preds
```

Out[132…
```
array([[2.12986943e-16, 1.81491881e-01, 1.84975068e-01, ...,
        3.53151011e-01, 3.61714398e-01, 3.61072723e-01],
       [3.64107784e-17, 2.76845163e-02, 2.75479698e-02, ...,
        4.81569560e-02, 4.93246906e-02, 4.92371895e-02],
       [2.07831434e-17, 2.56853867e-02, 2.68507499e-02, ...,
        4.45897741e-02, 4.56710098e-02, 4.55899903e-02],
       ...,
       [3.43969079e-17, 6.34395655e-02, 6.23832360e-02, ...,
        1.02259215e-01, 1.04738849e-01, 1.04553044e-01],
       [1.54584699e-16, 1.67722238e-01, 1.68514946e-01, ...,
        3.23424495e-01, 3.31267058e-01, 3.30679396e-01],
       [3.88274230e-17, 2.52640159e-02, 2.61251908e-02, ...,
        6.24256837e-02, 6.39394137e-02, 6.38259864e-02]])
```

In [133…
```
get_mse(item_preds, ratings_train)
```

Out[133… 11.460962134170675

Testing

In [135…
```
item_preds_test = ratings_test.dot(top_k_distances) / np.array([np.abs(top_k_distances).sum(a
```

In [138…
```
get_mse(item_preds, ratings_test)
```

Out[138… 12.127257997874693

La recomendación por semejanza entre películas no es la mejor, es muchísimo mejor la semejanza por valoración de usuario.

In [147…
```
items_df = pd.read_csv("../../Data-Sets/datasets/ml-100k/u.item.csv", sep = "\t", encoding="
items_df.head()
```

Out[147…

	0
0	1|Toy Story (1995)|01-Jan-1995||http://us.imdb...
1	2|GoldenEye (1995)|01-Jan-1995||http://us.imdb...
2	3|Four Rooms (1995)|01-Jan-1995||http://us.imd...
3	4|Get Shorty (1995)|01-Jan-1995||http://us.imd...
4	5|Copycat (1995)|01-Jan-1995||http://us.imdb.c...

Filtro colaborativo basado en los ítems KNN

In [176…
```
k = 30
```

```
neighbors = NearestNeighbors(n_neighbors=k, metric = 'cosine')

neighbors.fit(ratings_train.T)#Los ítems están en columnas
top_k_distances, top_k_items = neighbors.kneighbors(ratings_train.T, return_distance=True)
```

In [177…

```
user_predicts_k = np.zeros(ratings_train.T.shape)
```

In [183…

```
for i in range(ratings_train.T.shape[0]):
    if(i%50==0):
        print("iter"+str(i))
    den = 1
    user_predicts_k[i,:] = top_k_distances[i].dot(ratings_train.T[top_k_items[i]]) / np.array
```

```
iter0
iter50
iter100
iter150
iter200
iter250
iter300
iter350
iter400
iter450
iter500
iter550
iter600
iter650
iter700
iter750
iter800
iter850
iter900
iter950
iter1000
iter1050
iter1100
iter1150
iter1200
iter1250
iter1300
iter1350
iter1400
iter1450
iter1500
iter1550
iter1600
iter1650
```

In [194…

```
get_mse(user_predicts_k, ratings_train)
```

Out[194…

```
3.7413107324288792
```

In [195…

```
get_mse(user_predicts_k, ratings_test)
```

Out[195…

```
8.639650337967288
```