Caso Práctico de Módulo 3: Implementación de un Sistema de Recomendación de películas: MovieLens

- En este Notebook vamos a implementar y evaluar un ***Sistema de Recomendación basado en Filtrado Colaborativo con KNN*** usando la base de datos de MovieLens 100K.
- Esta base de datos contiene 100.000 votos con notas de 1 a 5 de 943 usuarios sobre 1682 películas.
- Este dataset ha sido dividido en votaciones de entrenamiento (80%) y votaciones de test (20%). Además para simplificar el problema, los códigos de usuarios e items, han sido modificados para que comience en 0 y terminen en el número de (usuarios / items) - 1.
- Para ***implementar y evaluar este Sistema de Recomendación*** realizaremos los siguientes paso:

(*): El punto del cálculo de las Recomendaciones no lo vamos a realizar ya que este punto no es relevante de cara a evaluar el sistema de recomendación implementado.

1.- Lectura del Dataset (Entrenamiento y Test)

- A continuación implementamos una función que data la ruta donde se encuentra un fichero con la estructura "ID_USER::ID_MOVIE::RATING" nos devuelva una matriz de votos:

In [1]:

```
def read_ratings_matrix(file):

    ratings = [[None for _ in range(NUM_ITEMS)] for _ in range(NUM_USERS)]

    with open(file, 'r') as reader:
        for line in reader:
            [u, i, rating] = line.split("::")
            ratings[int(u)][int(i)] = int(rating)

    return ratings


# Guardanos en variables los documentos donde tenemos los datos de los usuarios, peliculas y votacion, dejando
# una parte para entrenamiento y otra para test

TRAIN_RATINGS_FILE = "movielens_100k_training.txt"

TEST_RATINGS_FILE = "movielens_100k_test.txt"

# creamos las variables donde se recogen el total de usuarios y de items(peliculas que se valoran)

NUM_USERS = 943

NUM_ITEMS = 1682

# Creamos un objeto llamado "train_rating" que recoge la lectura del documento "TRAIN_RATINGS_FILE," mediante la funcion
# read_ratings_matrix, el cual genera unas listas encadenadas (usuario, pelicula, voto)

train_ratings = read_ratings_matrix(file=TRAIN_RATINGS_FILE)

# Hacemos los mismo con los datos de test

test_ratings = read_ratings_matrix(file=TEST_RATINGS_FILE)


import numpy as np
import pandas as pd

# Mostramos la matriz de votos de entrenamiento  a modo informativo
df_matriz = pd.DataFrame(data=np.array([np.array(xi) for xi in train_ratings]),
            index=["U{}".format(str(i)) for i in range(NUM_USERS)],
```

```
        columns=["I{}".format(str(i)) for i in range(NUM_ITEMS)])

df_matriz
```

Out[1]:

	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	...	I1672	I1673	I1674	I1675	I1676	I1677	I1678	I1679	I1680	I1681
U0	5	3	4	3	3	5	4	1	5	3	...	None	None	None	None	None	None	None	None	None	None
U1	4	None	None	None	None	None	None	None	None	2	...	None	None	None	None	None	None	None	None	None	None
U2	None	None	None	None	None	None	None	None	None	None	...	None	None	None	None	None	None	None	None	None	None
U3	None	None	None	None	None	None	None	None	None	None	...	None	None	None	None	None	None	None	None	None	None
U4	4	3	None	None	None	None	None	None	None	None	...	None	None	None	None	None	None	None	None	None	None
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
U938	None	None	None	None	None	None	None	None	5	None	...	None	None	None	None	None	None	None	None	None	None
U939	None	None	None	2	None	None	None	None	3	None	...	None	None	None	None	None	None	None	None	None	None
U940	5	None	None	None	None	None	4	None	None	None	...	None	None	None	None	None	None	None	None	None	None
U941	None	None	None	None	None	None	None	None	None	None	...	None	None	None	None	None	None	None	None	None	None
U942	None	5	None	None	None	None	None	None	3	None	...	None	None	None	None	None	None	None	None	None	None

943 rows × 1682 columns

2.- Cálculo de *Similaridades*

- A continuación implementamos una función que dada una matriz de votos y una métrica de similaridad, nos devuelva una matriz de similaridades entre cada par de usuarios.
- Tambien implemantamos las métricas de similaridad de:
  - MSD
  - Coseno
  - Correlación de Pearson
  - JMSD

In [2]:

```
def calculate_similarities(ratings_matrix, similarity_metric):

    # Creamos una matriz con valores de similaridad a -1
    similarities = [[float('-inf') for _ in range(NUM_USERS)] for _ in range(NUM_USERS)]

    # Recorremos la matriz por usuario
    for i, u in enumerate(ratings_matrix):
        if i%100==0:
            print("\tProcesandos {} Usuarios".format(i))
        for j, v in enumerate(ratings_matrix):
            if j != i: # No calculamos la similaridad para un mismo usuario
                similarities[i][j] = similarity_metric(u,v)

    return similarities


def rating_average(ratings):

    acc = 0
    count = 0
    for id_item in range(len(ratings)):
        if ratings[id_item] != None:
            acc += ratings[id_item]
            count += 1
    return acc / count


def msd_similarity(u, v):

    sum_r = 0
    count = 0

    for i in range(len(u)):
        if u[i] != None and v[i] != None:
            count += 1
            sum_r += math.pow((u[i] - v[i])/(MAX_RATING - MIN_RATING), 2)
```

```
    if count > 0:
        sim = 1-(sum_r/float(count))
        return sim
    else:
        return None


def cosine_similarity(u, v):

    numerador = 0
    denominador_u = 0
    denominador_v = 0

    count = 0

    for i in range(len(u)):
        if u[i] != None and v[i] != None:
            numerador += u[i] * v[i]
            denominador_u += math.pow(u[i], 2)
            denominador_v += math.pow(v[i], 2)
            count += 1
    if count > 0 and denominador_u != 0 and denominador_v != 0:
        cos = numerador / (math.sqrt(denominador_u) * math.sqrt(denominador_v))
        return cos
    else:
        return None


def jmsd_similarity(u, v):

    union = 0
    intersection = 0

    for i in range(len(u)):
        if u[i] != None and v[i] != None:
            intersection += 1
            union += 1
        elif u[i] != None or v[i] != None:
            union += 1

    if intersection > 0:
        jaccard = intersection / union
        return jaccard * msd_similarity(u,v)
    else:
        return None
```

1 - Vamos a calcular la metrica de similaridad MSD

In [3]:

```
import math

# Definimos dos variables, una con el valor minimo de los votos(1) y otra con el valor maximo de los votos(5)

MIN_RATING = 1

MAX_RATING = 5

# Elegimos dos usuarios, que van a ser el U100 y el U200, para comparar las votaciones que han realizado uno y otro.
# Las votaciones de U100 las vamos a meter en una lista llamada "u100" y las votaciones de U200 en una lista llamada "u200"

u100 = list(df_matriz.iloc[100])

u200 = list(df_matriz.iloc[200])

# Con la funcion msd_similarity podemos calcular la similaridad entre las votaciones de las distintas peliculas, por estos dos
# usuarios

print("Similaridad MSD entre los usuariuos U100 Y U200: ",msd_similarity(u100, u200))
```

Similaridad MSD entre los usuariuos U100 Y U200: 0.9140625

2 - Vamos a calcular la metrica de similaridad Coseno

In [4]:

```
# Con la funcion cosine_similarity podemos calcular esta metrica de similaridad, al igual que antes, con la metrica MSD
```

```
print("Similaridad COSENO entre los usuariuos U100 Y U200: ",cosine_similarity(u100, u200))
```

Similaridad COSENO entre los usuariuos U100 Y U200: 0.9543161696054487

3 - Vamos a calcular la metrica de similaridad JMSD

In [5]:

```
# Al igual que hemos hecho anteriormente, con la funcion jmsd_similarity podemos calular la similaridad JMSD

print("Similaridad JMSD entre los usuariuos U100 Y U200: ",jmsd_similarity(u100, u200))
```

Similaridad JMSD entre los usuariuos U100 Y U200: 0.05113636363636364

4 - Una vez calculadas las metricas de similariad entre dos unicos usuarios, vamos a calcular la similariad de todos. Para ello vamos a utilizar la metrica Coseno

In [6]:

```
# Para poder calcular la similaridad entre todos los usuarios, tenemos que utilizar la funcion "calculate_similarities".
# Como se ha indicado antes, se va a realizar usando el Coseno.

# Como el usuario U100 y el U200 pertenecen a los datos de entrenamiento, el argumento que le vamos a pasar a la funcion como
# ratings_matrix va a ser

similarities = calculate_similarities(train_ratings, msd_similarity)
```

```
Procesandos 0 Usuarios
Procesandos 100 Usuarios
Procesandos 200 Usuarios
Procesandos 300 Usuarios
Procesandos 400 Usuarios
Procesandos 500 Usuarios
Procesandos 600 Usuarios
Procesandos 700 Usuarios
Procesandos 800 Usuarios
Procesandos 900 Usuarios
```

In [7]:

```
# Mostramos la matriz de similaridades

matriz_similaridades = pd.DataFrame(data=np.array([np.array(xi) for xi in similarities]),
            index=["U{}".format(str(i)) for i in range(NUM_USERS)],
            columns=["U{}".format(str(i)) for i in range(NUM_USERS)])

matriz_similaridades.head(10)
```

Out[7]:

	U0	U1	U2	U3	U4	U5	U6	U7	U8	U9	...	U933	U934	U935	U936	
U0	-inf	0.90625	0.796875	0.84375	0.884375	0.894531	0.875862	0.943966	0.9375	0.926948	...	0.879058	0.775	0.917683	0.789062	0
U1	0.90625	-inf	0.859375	0.785714	0.9625	0.894531	0.934028	0.90625	0.84375	0.960938	...	0.891667	0.90625	0.941964	0.90625	0
U2	0.796875	0.859375	-inf	0.70625	0.9375	0.890625	0.799107	0.85	0.84375	0.8375	...	0.96875	0.875	0.875	0.84375	0
U3	0.84375	0.785714	0.70625	-inf	0.96875	0.6	0.7125	0.927083	0.96875	0.875	...	0.958333	0.9375	0.839286	0.635417	
U4	0.884375	0.9625	0.9375	0.96875	-inf	0.883929	0.800245	0.881579	0.578125	0.851786	...	0.862069	0.833333	0.901786	0.925	0
U5	0.894531	0.894531	0.890625	0.6	0.883929	-inf	0.898026	0.90625	0.857143	0.946875	...	0.885563	0.7125	0.897727	0.847222	0
U6	0.875862	0.934028	0.799107	0.7125	0.800245	0.898026	-inf	0.925676	0.857143	0.947154	...	0.872306	0.851562	0.889205	0.929688	0
U7	0.943966	0.90625	0.85	0.927083	0.881579	0.90625	0.925676	-inf	0.96875	0.967105	...	0.934028	0.979167	0.930556	0.775	
U8	0.9375	0.84375	0.84375	0.96875	0.578125	0.857143	0.857143	0.96875	-inf	0.971154	...	0.9375	1.0	0.958333	0.791667	
U9	0.926948	0.960938	0.8375	0.875	0.851786	0.946875	0.947154	0.967105	0.971154	-inf	...	0.897817	0.848214	0.959375	0.901786	0

10 rows × 943 columns

3 - Cálculo de los *K* Vecinos

3.- Cálculo de los *K-vecinos*

- A continuación implementamos una función que dada una matriz de similaridades entre usuarios y un valor de 'K' (número de vecinos a calcular), nos devuelva una matriz de "NUM_USERS x K_VECINOS" indicando los vecinos de cada usuario:

In [8]:

```
def calculate_neighbors(similarities_matrix, k_neighbors):

    neighbors = [None for _ in range(NUM_USERS)]

    for index, similarities in enumerate(similarities_matrix):
        i_neighbors = [i[0] for i in sorted(enumerate(similarities),
                                    key=lambda x:float('-inf') if x[1] is None else x[1],
                                    reverse=True)]
        neighbors[index] = i_neighbors[0:k_neighbors]

    return neighbors
```

In [9]:

```
# Con la funcion "calculate_neighbors" podemos obtener los k-vecinos mas cercanos de cada usuario, para ello tenemos que
# pasar como argumento la matriz de similaridades y el numero de vecinos mas cercanos que queremos que nos muestre. En este
# caso vamos a indicarle a la funcion que nos devuelva los 10 vecinos mas cercanos


neighbors = calculate_neighbors(similarities, 10)

# pasamos los resultados a un dataframe para verlos mejor

pd.DataFrame(data=np.array([np.array(xi) for xi in neighbors]),
             index=["U{}".format(str(i)) for i in range(NUM_USERS)],
             columns=["K{}".format(str(i)) for i in range(10)])

# este dataframe representa para cada usuario los k-vecinos mas cercanos en orden descedente, de tal manera, que para el U0, el
# vecino mas cercano seria el U154, seguido del U417, y asi sucesivamente.
```

Out[9]:

	K0	K1	K2	K3	K4	K5	K6	K7	K8	K9
U0	154	417	811	875	104	110	272	308	350	515
U1	50	113	166	186	368	521	699	889	349	418
U2	40	54	94	97	217	311	339	351	397	489
U3	24	35	40	67	88	95	96	113	114	138
U4	130	218	241	265	413	422	439	490	518	530
...	...	...	...	...	...	...	...	...	...	...
U938	8	18	32	38	45	46	60	87	95	96
U939	385	8	260	104	35	403	476	161	100	347
U940	8	46	50	99	110	134	146	170	186	190
U941	402	472	598	788	622	851	97	500	936	612
U942	260	442	573	812	821	45	463	208	309	375

943 rows × 10 columns

4.- Cálculo de las *Predicciones*

- A continuación implementamos un método que nos calcule las predicciones de los votos que emitirian los usuarios sobre los items con el método de agregación de "Media Ponderada":

In [13]:

```
def calculate_weighted_average_prediction(ratings_matrix, similarities_matrix, neighbors):

    # Creamos una matriz para el cálculo de predicciones
    predictions = [[None for _ in range(NUM_ITEMS)] for _ in range(NUM_USERS)]

    # Recorremos la matriz de votos
    for i, u in enumerate(ratings_matrix):
        for j, v in enumerate(ratings_matrix[0]):
            # Obtenemos las similaridades con cada vecino y si voto
```

```
        # Obtenemos las similaridades con cada vecino y si voto
        numerador = 0
        denominador = 0
        for neighbor in neighbors[i]:
            if ratings_matrix[neighbor][j] != None:
                numerador += similarities_matrix[i][neighbor] * ratings_matrix[neighbor][j]
                denominador += similarities_matrix[i][neighbor]

        predictions[i][j] = None if denominador == 0 else numerador/denominador

    return predictions

# Predecimos los votos aplicando la media ponderada, a traves, de la funcion calculate_weighted_average_prediction, que requiere
# como argumentos, la matriz, con los datos de entrenamiento, la matriz de similaridades y los vecinos

prediccion_mediaponderada = calculate_weighted_average_prediction(train_ratings, similarities, neighbors)
```

In [14]:

```
# Mostramos como quedan las predicciones de los items, usando la media ponderada

pd.DataFrame(data=np.array([np.array(xi) for xi in prediccion_mediaponderada]),
             index=["U{}".format(str(i)) for i in range(NUM_USERS)],
             columns=["I{}".format(str(i)) for i in range(NUM_ITEMS)])
```

Out[14]:

	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	...	I1672	I1673	I1674	I1675	I1676	I1677	I1678	I1679	I1680
U0	None	None	None	None	None	None	None	None	None	None	...	None	None	None	None	None	None	None	None	None
U1	4.0	None	None	None	None	None	None	5.0	None	None	...	None	None	None	None	None	None	None	None	None
U2	4.5	2.5	None	2.75	3.0	None	3.0	3.0	4.0	5.0	...	None	None	None	None	None	None	None	None	None
U3	4.6	None	None	4.0	None	None	4.4	4.666667	4.5	None	...	None	None	None	None	None	None	None	None	None
U4	4.0	None	None	4.0	None	None	3.0	None	4.666667	4.0	...	None	None	None	None	None	None	None	None	None
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
U938	4.5	None	None	4.0	None	5.0	4.333333	5.0	None	None	...	None	None	None	None	None	None	None	None	None
U939	3.666845	None	None	None	None	5.0	3.498014	None	None	None	...	None	None	None	None	None	None	None	None	None
U940	None	None	None	None	3.0	5.0	4.0	5.0	None	None	...	None	None	None	None	None	None	None	None	None
U941	3.799023	None	None	None	None	None	3.499738	None	4.498433	3.0	...	None	None	None	None	None	None	None	None	None
U942	4.496063	None	None	None	None	None	4.0	None	3.0	None	...	None	None	None	None	None	None	None	None	None

943 rows × 1682 columns

En este dataframe podemos ver, cual ha sido la votacion media ponderada que ha obtenido cada pelicula, del total de usuarios.

Por lo tanto, ya tenemos un ranking, ajustado a cada usuario, de cuales son las mejores peliculas para recomendarles.

El siguiente paso, obviamente, es proceder a recomendar a cada usuario, un determinado numero de peliculas, que no haya valorado y por tanto, que no haya visto.

Esto lo podemos hacer mediante la funcion "make_recommendations", que necesita pasarle como argumentos, el numero de peliculas que queremos recomendar, los datos de entrenamiento, y la prediccion de la media ponderada

In [15]:

```
def make_recommendations(num_recomendations, ratings_matrix, predictions_matrix):

    # Creamos una matriz para las recomendaciones
    recommendations = [[(None, None) for _ in range(num_recomendations)] for _ in range(NUM_USERS)]

    # Recorremos la matriz de votos
    for i, u in enumerate(ratings_matrix):
        for j, v in enumerate(ratings_matrix[0]):
            if ratings_matrix[i][j] == None:
                recommendations[i].append((j, predictions_matrix[i][j]))

        # Ordenamos los items a recomendar al usuario
        recommendations[i] = sorted(recommendations[i],
                                    key=lambda x:float('-inf') if x[1] is None else x[1],
                                    reverse=True)[0:num_recomendations]

    return [[x[0] for x in reco_user] for reco_user in recommendations]
```

```
num_peliculas = 4

prediccion_recomendaciones = make_recommendations(num_peliculas, train_ratings, prediccion_mediaponderada)
```

In [17]:

```
# Mostramos en un dataframe las 4 peliculas recomendadas a cada usuario

pd.DataFrame(data=np.array([np.array(xi) for xi in prediccion_recomendaciones]),
             index=["U{}".format(str(i)) for i in range(NUM_USERS)],
             columns=["Reco{}".format(str(i+1)) for i in range(num_peliculas)])
```

Out[17]:

	Reco1	Reco2	Reco3	Reco4
U0	311	317	473	510
U1	184	186	656	7
U2	9	13	56	108
U3	11	14	31	47
U4	10	11	44	56
...	...	...	...	...
U938	5	7	22	27
U939	5	21	48	87
U940	5	7	49	63
U941	407	115	147	149
U942	197	247	250	261

943 rows × 4 columns

Podemos decir, viendo este dataframe, que al U0, se le recomienda en primer lugar la pelicula 311, seguida de la 317, 473 y 510. Al U1, se le recominda la pelicula 184, seguida de la 186, 656 y 7. Y asi contodos los usuarios.

6.- Evaluación del Sistema de Recomendación (MAE)

- A continuación se implementa un método que dada la matriz de votos y la matriz de predicciones nos devuelve el MAE del Sistema de Recomendación:

In [18]:

```
def get_mae(ratings_matrix, predictions_matrix):

    mae_users = [None for _ in ratings_matrix]

    # Recorremos la matriz de votos
    for i, u in enumerate(ratings_matrix):
        # Calculamos el MAE de cada usuario
        sum_user = 0
        count = 0
        for j, v in enumerate(ratings_matrix[0]):
            if ratings_matrix[i][j] != None and predictions_matrix[i][j] != None:
                sum_user += abs(ratings_matrix[i][j] - predictions_matrix[i][j])
                count += 1
        mae_users[i] = sum_user/count if count > 0 else None

    return np.nanmean(np.array(mae_users, dtype=np.float), axis=0)


mae_media_ponderada = get_mae(train_ratings, prediccion_mediaponderada)

print("El MAE utilizando la media ponderada es de: ",mae_media_ponderada )
```

El MAE utilizando la media ponderada es de: 0.09377918951182435

```
<ipython-input-18-b2bce4900228>:16: DeprecationWarning: `np.float` is a deprecated alias for the builtin `float`. To silence this warning, use `float` b
y itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  return np.nanmean(np.array(mae_users, dtype=np.float), axis=0)
```

7.- Sistema de Recomendación y Evaluación

- A continuación se implementa el "Sistema de Recomendación", en el cual:
  - Calcularemos las similaridades para distintas métricas de similaridad
  - Realizaremos las predicciones para diferentes números de vecinos
  - Evaluaremos para cada experimento los conjuntos de entrenamiento y test.

In [23]:

```
import math
import numpy as np
import pandas as pd

# Número de usuarios e items
NUM_USERS = 943
NUM_ITEMS = 1682

# Notas máximas y mínimas dadas en la matriz de votos
MIN_RATING = 1
MAX_RATING = 5

# Ruta del los datasets de entrenamiento y test
TRAIN_RATINGS_FILE = 'movielens_100k_training.txt'
TEST_RATINGS_FILE = 'movielens_100k_test.txt'

# Lectura de los Datasets de Entrenamiento y Test
train_ratings = read_ratings_matrix(file=TRAIN_RATINGS_FILE)
test_ratings = read_ratings_matrix(file=TEST_RATINGS_FILE)

# Metricas, K vecinos y predicciones a probar
SIMILARITIES_METRICS = [('MSD', msd_similarity),
                        ('COSENO', cosine_similarity),
                        ('JMSD', jmsd_similarity)]
K_NEIGHBORS = [25, 50, 100, 150, 200, 300, 400, 500]


# Guardo en una lista los experimentos realizados, siendo un experimento una lista con 4 posiciones:
#     1.- Métrica de similaridad
#     2.- Número de vecinos
#     3.- MAE
#     4.- Entrenamiento o Test
experiments = []

for metric in SIMILARITIES_METRICS:
    print('{}: Cálculo de similatidades'.format(metric[0]))
    similarities_matrix = calculate_similarities(ratings_matrix=train_ratings,
                                                 similarity_metric=metric[1])
    for k in K_NEIGHBORS:

        print('  {} Vecinos'.format(k))
        neighbors_matrix = calculate_neighbors(k_neighbors=k,
                                               similarities_matrix=similarities_matrix) ##########

        # Calculamos las predicciones
        predictions = calculate_weighted_average_prediction(train_ratings, similarities_matrix, neighbors_matrix) ####

        # Calculamos el MAE para entrenamiento y test
        mae_train = get_mae(ratings_matrix=train_ratings,
                            predictions_matrix=predictions)
        mae_test = get_mae(ratings_matrix=test_ratings,
                           predictions_matrix=predictions)

        # Añadimos los experimentos a la lista
        experiments.append([metric[0], k, mae_train, "Train"])
        experiments.append([metric[0], k, mae_test, "Test"])
```

```
MSD: Cálculo de similatidades
  Procesandos 0 Usuarios
  Procesandos 100 Usuarios
  Procesandos 200 Usuarios
  Procesandos 300 Usuarios
  Procesandos 400 Usuarios
  Procesandos 500 Usuarios
  Procesandos 600 Usuarios
  Procesandos 700 Usuarios
```

```
 Procesandos 800 Usuarios
 Procesandos 900 Usuarios
  25 Vecinos
```

```
<ipython-input-18-b2bce4900228>:16: DeprecationWarning: `np.float` is a deprecated alias for the builtin `float`. To silence this warning, use `float` b
y itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  return np.nanmean(np.array(mae_users, dtype=np.float), axis=0)
```

```
  50 Vecinos
  100 Vecinos
  150 Vecinos
  200 Vecinos
  300 Vecinos
  400 Vecinos
  500 Vecinos
COSENO: Cálculo de similatidades
 Procesandos 0 Usuarios
 Procesandos 100 Usuarios
 Procesandos 200 Usuarios
 Procesandos 300 Usuarios
 Procesandos 400 Usuarios
 Procesandos 500 Usuarios
 Procesandos 600 Usuarios
 Procesandos 700 Usuarios
 Procesandos 800 Usuarios
 Procesandos 900 Usuarios
  25 Vecinos
  50 Vecinos
  100 Vecinos
  150 Vecinos
  200 Vecinos
  300 Vecinos
  400 Vecinos
  500 Vecinos
JMSD: Cálculo de similatidades
 Procesandos 0 Usuarios
 Procesandos 100 Usuarios
 Procesandos 200 Usuarios
 Procesandos 300 Usuarios
 Procesandos 400 Usuarios
 Procesandos 500 Usuarios
 Procesandos 600 Usuarios
 Procesandos 700 Usuarios
 Procesandos 800 Usuarios
 Procesandos 900 Usuarios
  25 Vecinos
  50 Vecinos
  100 Vecinos
  150 Vecinos
  200 Vecinos
  300 Vecinos
  400 Vecinos
  500 Vecinos
```

- Pasamos los resultados de los experimentos a un DataFrame:

In [24]:

```
# Pasamos los resultados de los experimentos a un DataFrame
df_results = pd.DataFrame.from_records(experiments,
                                       columns=['Metrica', 'K-Vecinos', 'MAE', 'Train/Test'])
```

- Mostramos los resultados de los experimentos con los datos de entrenamiento:

In [25]:

```
pd.pivot_table(df_results[df_results['Train/Test']=='Train'],
               values=['MAE'],
               index=['K-Vecinos'],
               columns=['Metrica'])
```

Out[25]:

	MAE

Metrica	MAE		
K-Vecinos	COSENO	JMSD	MSD
25	0.739705	0.789135	0.189270
50	0.714021	0.784164	0.313055
100	0.682957	0.788175	0.439108
150	0.668099	0.792791	0.499039
200	0.655862	0.795384	0.537014
300	0.667087	0.800467	0.596012
400	0.693104	0.803457	0.641578
500	0.719150	0.805755	0.681724

- Mostramos los resultados de los experimentos con los datos de test:

In [26]:

```
pd.pivot_table(df_results[df_results['Train/Test']=='Test'],
               values=['MAE'],
               index=['K-Vecinos'],
               columns=['Metrica'])
```

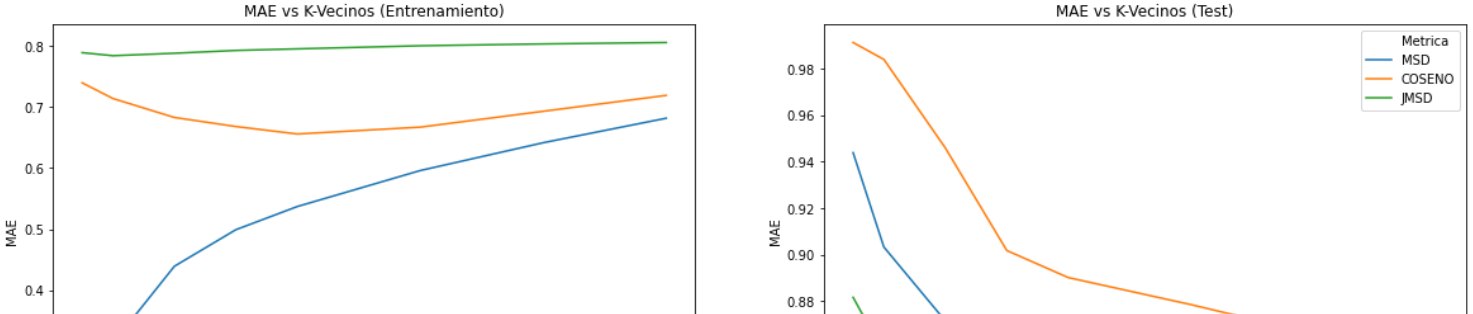
Out[26]:

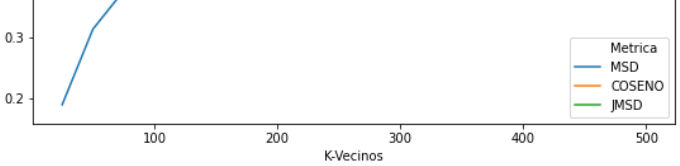
	MAE		
Metrica	COSENO	JMSD	MSD
K-Vecinos			
25	0.991211	0.881689	0.943860
50	0.983949	0.857734	0.903342
100	0.945913	0.850982	0.871570
150	0.901831	0.860059	0.845946
200	0.890210	0.859590	0.832801
300	0.878561	0.857354	0.829561
400	0.865961	0.863219	0.836281
500	0.862582	0.865219	0.839224

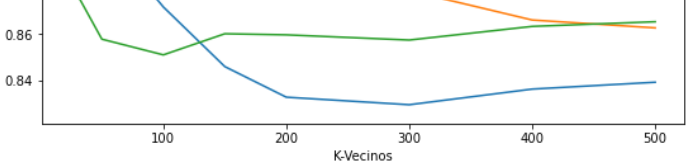
- Pintamos los resultados

In [27]:

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.subplots(figsize = (20, 6))
# Pintamos el MAE con los datos de Entrenamiento
plt.subplot(1, 2, 1)
plt.title('MAE vs K-Vecinos (Entrenamiento)')
sns.lineplot(x="K-Vecinos", y="MAE", hue="Metrica", data=df_results[df_results['Train/Test']=='Train'])

# Pintamos el MAE con los datos de Test
plt.subplot(1, 2, 2)
plt.title('MAE vs K-Vecinos (Test)')
sns.lineplot(x="K-Vecinos", y="MAE", hue="Metrica", data=df_results[df_results['Train/Test']=='Test'])

plt.show()
```







Antes que nada, decir que el MAE(error absoluto medio), es una manera de medir la precision de un modelo.

En este caso, estamos midiendo, o comparando, el calculo de similaridades, utilizando tres metricas distintas (MSD,COSENO,JMSD) y utilizando un numero k-vecinos que va desde los 25 hasta los 500.

Dicho esto, observando la grafica y teniendo en cuenta que cuanto menor es el MAE, para un modelo dado, mejor será el modelo a la hora de predecir los valores reales, podemos decir que:

- Respecto a los datos de entrenamiento, la mejor metrica para usar es la MSD CON 100 - 200 vecinos
- Respecto a los datos de test, la mejor metrica para usar es también la MSD con 200 - 300 vecinos.

En conclusion, despues de analizar estas graficas, diria que la metrica a utilizar seria la MSD con unos 200 vecinos.

In [ ]: