MINE 4201- Sistemas de Recomendación

Taller 1 - Modelos Colaborativos

Integrantes

- Cristian C. Moreno Mojica(c.morenom@uniandes.edu.co (mailto:c.morenom@uniandes.edu.co))
- Juan J. Ovalle Villamil(jj.ovalle@uniandes.edu.co (mailto:jj.ovalle@uniandes.edu.co))
- Maria C. Rodríguez Niño (mc.rodriguezn12@uniandes.edu.co (mailto:mc.rodriguezn12@uniandes.edu.co))

Librerias

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from surprise import Reader
from surprise import Dataset
from surprise.model_selection import train_test_split
from sklearn.model_selection import train_test_split  as train_test_split1
from surprise import KNNBasic
from surprise import accuracy
from surprise.model_selection import GridSearchCV
from joblib import dump, load
from sklearn.metrics import mean_squared_error
```

1) Conocimiento del dataset de trabajo

Existen dos conjuntos de datos:

1. user_info : contiene información del usuario como genero, edad, pais y fecah de registro.
2. user_interact : contiene la interaccion de los usuarios y los items (artistas y canciones).

In [2]:
```
user_info = pd.read_csv('data/lastfm-dataset-1K/userid-profile.tsv', sep = '
user_info.head()
```

Out[2]:

	#id	gender	age	country	registered
0	user_000001	m	NaN	Japan	Aug 13, 2006
1	user_000002	f	NaN	Peru	Feb 24, 2006
2	user_000003	m	22.0	United States	Oct 30, 2005
3	user_000004	f	NaN	NaN	Apr 26, 2006
4	user_000005	m	NaN	Bulgaria	Jun 29, 2006

In [3]:
```
user_interact = pd.read_csv('data/lastfm-dataset-1K/userid-timestamp-artid-a
                    sep = '\t',
                   names = ['UserId', 'TimeStamp', 'ArtId', 'ArtName', 'TraId'
user_interact.head()
```

Out[3]:

	UserId	TimeStamp	ArtId	ArtName	TraId	TraName
0	user_000001	2009-05-04T23:08:57Z	f1b1cf71-bd35-4e99-8624-24a6e15f133a	Deep Dish	NaN	Fuck Me Im Famous (Pacha Ibiza)-09-28-2007
1	user_000001	2009-05-04T13:54:10Z	a7f7df4a-77d8-4f12-8acd-5c60c93f4de8	坂本龍一	NaN	Composition 0919 (Live_2009_4_15)
2	user_000001	2009-05-04T13:52:04Z	a7f7df4a-77d8-4f12-8acd-5c60c93f4de8	坂本龍一	NaN	Mc2 (Live_2009_4_15)
3	user_000001	2009-05-04T13:42:52Z	a7f7df4a-77d8-4f12-8acd-5c60c93f4de8	坂本龍一	NaN	Hibari (Live_2009_4_15)
4	user_000001	2009-05-04T13:42:11Z	a7f7df4a-77d8-4f12-8acd-5c60c93f4de8	坂本龍一	NaN	Mc1 (Live_2009_4_15)

La base user_interact refleja las interacciones de los usuarios y los items. Dicha relación se puede ver desde los dos tipos de items disponibles, es decir, canciones y artistas. Asi, **cada registro de la tabla representa una cancion de un artista x escuchada por un usuario y**. Con esto en mente, podemos llegar a insights relevantes para entender el conjunto de datos. Por ejemplo, si se desea ver las canciones o artistas mas escuchados, basta con realizar un conteo por el id del artista o el id de la canción respectivamente.

In [4]:
```
print('----------------------- Top 10 Artistas -----------------------')
display(user_interact.groupby(['ArtId','ArtName']).size().reset_index(name =

print('----------------------- Top 10 Canciones -----------------------')
user_interact.groupby(['TraId','TraName']).size().reset_index(name = 'Frecue
```

```
----------------------- Top 10 Artistas -----------------------
```

	ArtId	ArtName	Frecuencia
70252	a74b1b7f-71a5-4011-9441-d0b5e4122711	Radiohead	115099
74262	b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d	The Beatles	100126
77135	b7ffd2af-418f-4be2-bdd1-22f8b48613da	Nine Inch Nails	84317
65734	9c9f1380-2516-4fc9-a3e6-f9f61941d090	Muse	63139
85720	cc197bad-dc9c-440d-a5b5-d52ba2e14234	Coldplay	62212
55945	8538e728-ca0b-4321-b7e5-cff6565dd4c0	Depeche Mode	59609
55359	83d91898-7763-47d7-b03b-b92132375c47	Pink Floyd	58484
99	0039c7ae-e1a7-4a7d-9b49-0cbc716821a6	Death Cab For Cutie	58046
55629	847e8284-8582-4b0e-9c26-b042a4f49e57	Placebo	53467
1610	03ad1736-b7c9-412a-b442-82536d63a5c4	Elliott Smith	50202

```
----------------------- Top 10 Canciones -----------------------
```

Out[4]:

	TraId	TraName	Frecuencia
821625	db16d0b3-b8ce-4aa8-a11a-e4d53cc7f8a6	Such Great Heights	3991
476769	7f1f45c0-0101-49e9-8d69-23951d271163	Love Will Tear Us Apart	3651
593060	9e2ad5bc-c6f9-40d2-a36f-3122ee2072a3	Karma Police	3533
293118	4e17b118-70a6-4c1f-b326-b4ce91fd3fad	Soul Meets Body	3479
957215	ff1e3e1a-f6e8-4692-b426-355880383bb6	Supermassive Black Hole	3463
822442	db4c9220-df76-4b42-b6f5-8bf52cc80f77	Heartbeats	3155
932176	f874c752-65bc-4d50-ac7e-932243ae9f02	Rebellion (Lies)	3047
363124	60e94685-0481-4d3d-bd84-11c389d9b2a5	Starlight	3040
710562	bd782340-6fa5-4b52-aa5a-ceafb9bc0340	Gimme More	3002
280955	4ad08552-6c35-49ed-bcc6-6822c8f9dfd8	When You Were Young	2997

2) Preprocesamiento de datos

a) Transforme los datos correspondientes a la interacción entre usuarios e ítems, implementando una estrategia para convertir estos datos en unos que sean compatibles con los modelos vistos en clase

datos en aras que sean compatibles con los modelos vistos en clase. Justifique en el informe sus decisiones en este paso.

Antes de establecer una forma de medir los ratings de cada usuario, se necesita establecer una definición de item, es decir, si se utilizaran canciones o artistas.

In [5]:
```
1 len(user_interact['ArtId'].unique())
```

Out[5]: 107296

In [6]:
```
1 len(user_interact['TraId'].unique())
2 len(user_interact['UserId'].unique())
```

Out[6]: 992

Existen 107296 artistas unicos y 960403 canciones unicas en el conjunto de datos. Por limitaciones de las maquinas utilizadas, se trabajaran los artistas como items.

Con base en Jawaheer, Szomszor y Kostkova (2010) (https://core.ac.uk/download/pdf/207051652.pdf (https://core.ac.uk/download/pdf/207051652.pdf)) decidimos utilizar 3 alternativas para medir los ratings de los usuarios:

1. La frecuencia total de los items por usuario.
2. El logaritmo de la frecuencia total de los items por usuario.
3. Una normalizacion de la frecuencia total de los items por usuario definida de la siguiente forma: la frecuencia sobre el numero total de artistas que un usuario ha esuchado.

In [7]:
```
1 #Se encuentra el numero total de artistas para cada usuario y se guarda en a
2 aux_artistas = user_interact.groupby('UserId').agg({'ArtId':'nunique'}).rese
3 aux_artistas.head()
```

Out[7]:

	UserId	Nart
0	user_000001	578
1	user_000002	1211
2	user_000003	840
3	user_000004	1623
4	user_000005	847

In [8]:
```
#Se encuentran cada una de las alternativas
ratings_artistas = user_interact.groupby(['UserId', 'ArtId']).size().reset_i
ratings_artistas['log_frecuencia'] = np.log10(ratings_artistas['frecuencia']
ratings_artistas_merged = ratings_artistas.merge(aux_artistas, on = 'UserId'
ratings_artistas_merged['normalizada'] = ratings_artistas_merged['frecuencia
ratings_artistas_merged.head()
```

Out[8]:

	UserId	ArtId	frecuencia	log_frecuencia	Nart	normalizada
0	user_000001	00c73a38-a449-4990-86ca-5088dde1b8df	2	0.301030	578	0.00346
1	user_000001	012a77c9-c897-494f-87d0-0a730996494d	1	0.000000	578	0.00173
2	user_000001	014ba96b-b8da-49e3-8a2b-b720ae42e84c	3	0.477121	578	0.00519
3	user_000001	01ce7548-dab4-4ca6-8dfc-8e2e4b50d461	4	0.602060	578	0.00692
4	user_000001	03282c56-8a24-42f4-8bfc-96188933aefa	4	0.602060	578	0.00692

In [9]:
```
#Se elimina numero total de artistas.
ratings_artistas_merged = ratings_artistas_merged.drop('Nart', axis = 1)
ratings_artistas_merged.head()
```

Out[9]:

	UserId	ArtId	frecuencia	log_frecuencia	normalizada
0	user_000001	00c73a38-a449-4990-86ca-5088dde1b8df	2	0.301030	0.00346
1	user_000001	012a77c9-c897-494f-87d0-0a730996494d	1	0.000000	0.00173
2	user_000001	014ba96b-b8da-49e3-8a2b-b720ae42e84c	3	0.477121	0.00519
3	user_000001	01ce7548-dab4-4ca6-8dfc-8e2e4b50d461	4	0.602060	0.00692
4	user_000001	03282c56-8a24-42f4-8bfc-96188933aefa	4	0.602060	0.00692

b) Tome los datos compatibles con modelos colaborativos y pártalos en dos conjuntos: un grupo de datos le sirve para construir el modelo y el resto para medir sus predicciones. Sepárelos en archivos distintos.

En el siguiente loop, con el fin de escoger cual metrica representa mejor los ratings de cada usuario, se realiza la separación de train y test para cada una de las alternativas. El procedimiento es:

1. Se define la escala de los ratings en el Reader.
2. Se carga el conjunto de datos a la estructura que utiliza la libreria surprise.
3. Se divide en train y test respectivamente.

In [10]:
```
for each in ['frecuencia','log_frecuencia', 'normalizada']:
    print('---------' + '  Alternativa: ' + str(each) + ' ---------')
    print('La escala para la alternativa ' + str(each) + 'va de ' + str(roun
    reader = Reader(rating_scale = (min(ratings_artistas_merged[each]), max(
    surprise_dataset = Dataset.load_from_df( ratings_artistas_merged[[ 'User
    train_set, test_set = train_test_split(surprise_dataset, test_size = 0.2
    print(str(type(surprise_dataset)))
    print(str(type(train_set)))
    print(str(type(test_set)))
    print('realizado.')
```

```
---------  Alternativa: frecuencia ---------
La escala para la alternativa frecuenciava de 1 hasta 26496
<class 'surprise.dataset.DatasetAutoFolds'>
<class 'surprise.trainset.Trainset'>
<class 'list'>
realizado.
---------  Alternativa: log_frecuencia ---------
La escala para la alternativa log_frecuenciava de 0.0 hasta 4.42
<class 'surprise.dataset.DatasetAutoFolds'>
<class 'surprise.trainset.Trainset'>
<class 'list'>
realizado.
---------  Alternativa: normalizada ---------
La escala para la alternativa normalizadava de 0.0 hasta 236.57
<class 'surprise.dataset.DatasetAutoFolds'>
<class 'surprise.trainset.Trainset'>
<class 'list'>
realizado.
```

3) Construcción de modelos colaborativos usuario-usuario

Como son 3 alternativas a probar, los procedimiento de los literales a) b) y c) son realizados en forma de loop.

In [11]:

```
alternativa = ['frecuencia','log_frecuencia', 'normalizada']
metrica_similitud = ['pearson', 'cosine']
rmse = []
for each in alternativa:
    for each_2 in metrica_similitud:
        print('---------' + '  Alternativa: ' + str(each) + ' ' + str(each_2
        reader = Reader(rating_scale = (min(ratings_artistas_merged[each]),
        surprise_dataset = Dataset.load_from_df( ratings_artistas_merged[[ '
        train_set, test_set = train_test_split(surprise_dataset, test_size =
        algoritmo = KNNBasic(k = 20, min_k = 2,
                sim_options = dict(name = each_2,
                                   user_based = True)).fit(trainset = train_s
        algoritmo.fit(trainset = train_set)
        test_predictions = algoritmo.test(test_set)
        rmse_intern = accuracy.rmse(test_predictions, verbose = True )
        print(rmse_intern)
        rmse.append(rmse_intern)
        print('realizado.')
```

```
---------  Alternativa: frecuencia pearson ---------
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
RMSE: 125.0326
125.03256479553293
realizado.
---------  Alternativa: frecuencia cosine ---------
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
RMSE: 138.8532
138.85321808069418
realizado.
---------  Alternativa: log_frecuencia pearson ---------
Computing the pearson similarity matrix...

C:\Users\mariac.rodriguez\Anaconda3\lib\site-packages\surprise\prediction_alg
```

Para evaluar las predicciones de cada uno de los modelos se utiliza el RMSE. Así, se quiere encontrar el modelo con el menor RMSE posible.

In [13]:
```
resultados_1 = pd.DataFrame(list(zip(np.repeat(alternativa, 2), metrica_simi
print('Resultados con k = 20')
resultados_1
```

```
Resultados con k = 20
```

Out[13]:

	Alternativa	Metrica Similitud	RMSE
5	normalizada	cosine	0.381981
4	normalizada	pearson	0.412365
2	log_frecuencia	pearson	0.636512
3	log_frecuencia	cosine	0.645505
0	frecuencia	pearson	125.032565
1	frecuencia	cosine	138.853218

Desde esta primera aproximación del modelo user-user, se puede evidencair que la mejor alternativa es la normalizada, luego la log_frecuencia y por ultimo la frecuencia. Estos resultados tienen logica con lo encontrado por Jawaheer, Szomszor y Kostkova (2010).

Artistas - jaccard con Normalizada modelo User-User

d) Para jaccard se construyo una fucnion especifc auwe reaclice torod el proceso para hallar la similitud entre cada usuario o item y realice la predcción de acuerdo al tipo de modelo establecido

In [14]:
```
def jaccard_user(train_pivot,item,user,k):
    train_pivot = np.array(train_pivot)
    sim = []
    for row in range(len(train_pivot)):
        a = train_pivot[[row,user]][:,~np.isnan(train_pivot[[row,user]]).any
        if a.size!=0:
            s_a = set(a[0])
            s_b = set(a[1])
            sim.append(len(s_a.intersection(s_b))/len(s_a.union(s_b)))
        else:
            sim.append(0)
    rating = [train_pivot[i][item] for i in range(train_pivot.shape[0])]
    rating = np.where(np.isnan(rating),0,rating)
    sim_k = sorted(sim)[-k:]
    rating_k = rating[[sim.index(i) for i in sim_k[-k:]] ]
    estimate = (rating_k * sim_k).sum() / sum(sim_k)
    return estimate
```

In [27]:
```
reader = Reader( rating_scale = ( min(ratings_artistas_merged['normalizada']
ratings_artistas_frec = ratings_artistas_merged[['UserId', 'ArtId','normaliz
user = ratings_artistas_frec.groupby('UserId').agg({'ArtId':'count'}).reset_
user = user[user.ArtId>1]
ratings_artistas_frec = ratings_artistas_frec[ratings_artistas_frec.UserId.i
train_set, test_set = train_test_split1(ratings_artistas_frec, test_size=.2,
```

In [43]:
```
train_pivot=train_set.pivot_table(index=['UserId'], columns=['ArtId'], value
```

In [52]:
```
%%time
predict = []
lista=[]
k=20
while (k <100):
    for i in range(40):
        UserId , ArtId =  test_set.iloc[i,:][['UserId','ArtId']]
        item = train_pivot.columns.get_loc(ArtId)
        user = train_pivot.index.get_loc(UserId)
        predict.append([UserId,ArtId, jaccard_user(train_pivot,item,user,k)]
    predict=pd.DataFrame(predict, columns=['UserId','ArtId','Jaccard'])
    predict=pd.merge(test_set, predict)
    rms = np.sqrt(mean_squared_error(predict['normalizada'], predict['Jaccar
    k=k+20
    print(rms)
    lista.append([rms,k])
display(lista)

```

```
0.036225525766172294
0.036225525766172294
0.036225525766172294
0.036225525766172294

[[0.036225525766172294, 40],
 [0.036225525766172294, 60],
 [0.036225525766172294, 80],
 [0.036225525766172294, 100]]

Wall time: 1min 17s
```

Artistas Jaccard con Normalizada modelo item-item

In [42]:
```
rating = ratings_artistas_merged[['UserId', 'ArtId','normalizada']].copy()
lista_items =  rating.groupby('ArtId').agg({'UserId':'count'}).reset_index()
lista_items = lista_items [lista_items.UserId > np.percentile(lista_items.Us
ratings_artistas_frec = rating[rating.ArtId.isin(lista_items.ArtId)]
train_set, test_set = train_test_split1(ratings_artistas_frec, test_size=.2,
```

In [50]:
```
train_pivot_item=train_set.pivot_table(index=['ArtId'], columns=['UserId'],
```

In [*]:
```
%%time
predict_item = []
ista=[]
k=20
while (k <100):
    for i in range(150):
        UserId , ArtId =  test_set.iloc[i,:][['UserId','ArtId']]
        item = train_pivot_item.columns.get_loc(UserId)
        user = train_pivot_item.index.get_loc(ArtId)
        predict_item.append([UserId,ArtId, jaccard_user(train_pivot_item,ite
    predict_item=pd.DataFrame(predict_item, columns=['UserId','ArtId','Jacca
    predict_item=pd.merge(test_set, predict_item)
    rms = np.sqrt(mean_squared_error(predict_item['normalizada'], predict_it
    k=k+20
    print(rms)
    lista.append([rms,k])
display(lista)

```

```
0.09362647634834854
0.09362647634834854
0.09362647634834854
```

d) Varíe la estrategia de selección de vecinos por umbral de similitud y por número de vecinos. Revise cuál es el impacto al variar estos parámetros.

Se define una busqueda de grilla con 5 folds de validacion cruzada que va a recorrer las siguientes opciones :

1. Todos los k's dentro de [20,99]
2. metrica de similitud coseno y pearson.

In [ ]:
```
#Se establece el diccionario para pasar a la busqueda de grilla.
grilla = dict(k = range(20, 100),
              sim_options = dict(name = ['cosine', 'pearson'],
                                 user_based = [True]))
```

In [ ]:
```
#Se crean las bases para cada una de las alternativas
data = Dataset.load_from_df( ratings_artistas_merged[[ 'UserId', 'ArtId', 'f
                            Reader(rating_scale = (min(ratings_artistas_merg
data2 = Dataset.load_from_df( ratings_artistas_merged[[ 'UserId', 'ArtId', '
                             Reader(rating_scale = (min(ratings_artistas_mer
data3 = Dataset.load_from_df( ratings_artistas_merged[[ 'UserId', 'ArtId', '
                             Reader(rating_scale = (min(ratings_artistas_mer

```

Los parametros del mejor modelo son guardados mediante la libreria joblib dado que cada busqueda de grilla demora 6 horas aproximadamente. Ademas, se pueden guardar los resultados en un conjunto de datos.

In [ ]:
```
#%%time
#gs = GridSearchCV(KNNBasic, grilla, measures = ['RMSE','MAE'], cv = 5, jobl
#gs.fit(data)
#dump(gs.best_params, 'best_model_frencuencia.joblib')
```

In [ ]:
```
#%%time
#gs2 = GridSearchCV(KNNBasic, grilla, measures = ['RMSE','MAE'], cv = 5, job
#gs2.fit(data2)
#dump(gs2.best_params, 'best_model_log_frencuencia.joblib')
```

In [ ]:
```
#%%time
#gs3 = GridSearchCV(KNNBasic, grilla, measures = ['RMSE','MAE'], cv = 5, job
#gs3.fit(data3)
#dump(gs3.best_params, 'best_model_normalizado.joblib')
```

In [ ]:
```
#Guardar resultados
#best_results_freq = pd.DataFrame(gs.cv_results)
#best_results_freq['alternativa'] = 'frencuencia'

#best_results_log = pd.DataFrame(gs2.cv_results)
#best_results_log['alternativa'] = 'log_frecuencia'

#best_results_norm = pd.DataFrame(gs3.cv_results)
#best_results_norm['alternativa'] = 'normalizada'

#best_results = best_results_freq.append(best_results_norm).append(best_resu
#best_results.to_csv('best_results_user_user.csv')
```

In [82]:
```
#Resultados
freq_user_based = load('best_model_frencuencia.joblib')
display(freq_user_based)

logfreq_user_based = load('best_model_log_frencuencia.joblib')
display(logfreq_user_based)

norm_user_based = load('best_model_normalizado.joblib')
norm_user_based
```

```
{'rmse': {'k': 99, 'sim_options': {'name': 'cosine', 'user_based': True}},
 'mae': {'k': 34, 'sim_options': {'name': 'cosine', 'user_based': True}}}

{'rmse': {'k': 37, 'sim_options': {'name': 'pearson', 'user_based': True}},
 'mae': {'k': 25, 'sim_options': {'name': 'pearson', 'user_based': True}}}
```

Out[82]:
```
{'rmse': {'k': 99, 'sim_options': {'name': 'cosine', 'user_based': True}},
 'mae': {'k': 99, 'sim_options': {'name': 'cosine', 'user_based': True}}}
```

In [83]:
```
#Resultados user user
best_user_user = pd.read_csv('best_results_user_user.csv')
display(best_user_user.shape)
best_user_user.head()
```

```
(480, 24)
```

Out[83]:

	split0_test_rmse	split1_test_rmse	split2_test_rmse	split3_test_rmse	split4_test_rmse	mean_te
0	147.958025	109.591278	130.751027	116.724733	114.641542	12
1	151.052021	114.798900	132.632230	118.841519	117.316292	12
2	147.805276	109.460653	130.682048	116.534074	114.511492	12
3	150.862114	114.666573	132.554724	118.662178	117.200316	12

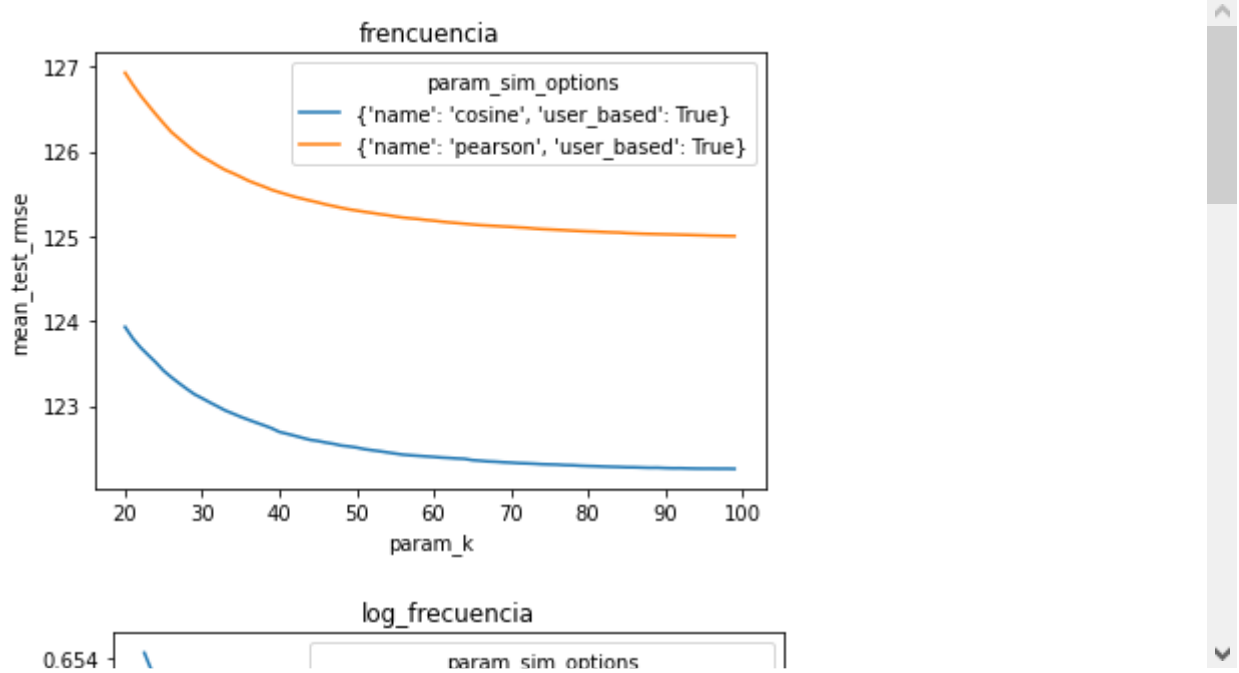
In [106]:
```
#Una vez mas los mejores modelos fueron
best_user_user_sum = best_user_user[['alternativa','param_k','param_sim_opti
print('------------------------ Mejores modelos user-based -----------------
best_user_user_sum

```

```
------------------------ Mejores modelos user-based -----------------
```

Out[106]:

	alternativa	param_k	param_sim_options	mean_test_rmse
318	normalizada	99	{'name': 'cosine', 'user_based': True}	0.383043
355	log_frecuencia	37	{'name': 'pearson', 'user_based': True}	0.646242
158	frencuencia	99	{'name': 'cosine', 'user_based': True}	122.261087

In [108]:
```
for each in ['frencuencia','log_frecuencia','normalizada']:
    temp = best_user_user[['param_k','param_sim_options','mean_test_rmse','a
    sns.lineplot(data = temp, x = 'param_k', y = 'mean_test_rmse', hue = 'pa
    plt.title(each)
    plt.show()
```



e) Revise la estrategia de ponderación por significancia de McLaughlin's [1] (McLaughlin's significance weighting) y revise cuál es el impacto al variar los parámetros de esta estrategia.

```
Respecto a la estrategia de ponderacion menciondad por McLaughlin's Nos
pareció útil ajustar las ponderaciones de similitud según la cantidad de
usuarios en común, si la cantidad de usuarios comunes estaba por debajo de
un cierto umbral y e ubral definido fue los de percentil 90 adicional por
recursos tencologico de las maquinas  donde se ejecutaba el modelo.
```

4) Construcción de modelos colaborativos ítem - ítem

Antes de realizar el mismo proceso, es importante tener en cuenta que:

1. Por limitaciones de las maquinas solo se podra utilizar una parte de los items para realizar los mdoelos.
2. Esta seleccion se hara apartir del conteo de ratings de cada item

In [111]:
```
print('Como se puede ver, para usuarios si era posible utilizar toda la base

print(len(ratings_artistas_merged.UserId.unique()))
len(ratings_artistas_merged.ArtId.unique())
```

```
Como se puede ver, para usuarios si era posible utilizar toda la base ya que er
an unicamente 992 usuarios, por ende, solo tenia que calcular 992 similitudes.
Sin embargo, para items este proceso se tendria que hacer 107295 veces.
992
```

Out[111]: 107295

In [112]:
```
ratings_artistas_merged.groupby('ArtId').size().reset_index(name = 'Frecuenc
```

Out[112]:

ArtId	Frecuencia
a74b1b7f-71a5-4011-9441-d0b5e4122711	710
cc197bad-dc9c-440d-a5b5-d52ba2e14234	636
b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d	615
9c9f1380-2516-4fc9-a3e6-f9f61941d090	594
69ee3720-a7cb-4402-b48d-a02c366f2bcf	577
...	...
74c67807-de42-4558-8943-5dee420a79b2	1
74c7da0f-0c22-4159-a621-d982b8267d67	1
74c86936-f54b-43cd-bc22-5fbc9e70b8a9	1
74ca90c6-2f9f-4c99-bdf9-734e4ab571d6	1
ffff44bd-e5a5-4e87-8700-35481264e37d	1

107295 rows × 1 columns

In [125]:
```
#Se crea una lista con los id de los items que interesan
temp_1 = ratings_artistas_merged.groupby('ArtId').size().reset_index(name =
items_disponibles = list(temp_1['ArtId'][temp_1['Frecuencia'] > np.percentil
print('---Se filtra el cojunto de datos a el 10% mas calificado --')
print(str( np.percentile(temp_1['Frecuencia'] , 90)) + ' es el percentil 90'
```

```
---Se filtra el cojunto de datos a el 10% mas calificado --
13.0 es el percentil 90
```

In [126]:
```
#Se filtra  el conjuto de datos que se utilizaran para los modelos item-item
ratings_artistas_merged_item = ratings_artistas_merged[ratings_artistas_merg
ratings_artistas_merged_item.shape
```

Out[126]: (566888, 5)

In [127]:

```
alternativa = ['frecuencia','log_frecuencia', 'normalizada']
metrica_similitud = ['pearson', 'cosine']
rmse_item = []
for each in alternativa:
    for each_2 in metrica_similitud:
        print('---------' + '  Alternativa: ' + str(each) + ' ' + str(each_2
        reader = Reader(rating_scale = (min(ratings_artistas_merged_item[eac
        surprise_dataset = Dataset.load_from_df( ratings_artistas_merged_ite
        train_set, test_set = train_test_split(surprise_dataset, test_size =
        algoritmo = KNNBasic(k = 20, min_k = 2,
                sim_options = dict(name = each_2,
                                   user_based = False)).fit(trainset = train_
        algoritmo.fit(trainset = train_set)
        test_predictions = algoritmo.test(test_set)
        rmse_intern = accuracy.rmse(test_predictions, verbose = True )
        print(rmse_intern)
        rmse_item.append(rmse_intern)
        print('realizado.')
```

```
---------  Alternativa: frecuencia pearson ---------
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
RMSE: 129.9694
129.96944911935984
realizado.
---------  Alternativa: frecuencia cosine ---------
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
RMSE: 140.5441
140.54408095130304
realizado.
---------  Alternativa: log_frecuencia pearson ---------
Computing the pearson similarity matrix...

C:\ProgramData\Anaconda3\lib\site-packages\surprise\prediction_algorithms\alg
```

In [128]:
```
resultados_2 = pd.DataFrame(list(zip(np.repeat(alternativa, 2), metrica_simi
print('Con K fijo a 20')
resultados_2
```

```
Con K fijo a 20
```

Out[128]:

	Alternativa	Metrica Similitud	RMSE
5	normalizada	cosine	0.252194
4	normalizada	pearson	0.297666
3	log_frecuencia	cosine	0.676680
2	log_frecuencia	pearson	0.685043
0	frecuencia	pearson	129.969449
1	frecuencia	cosine	140.544081

In [130]:
```
#Se establece el diccionario para pasar a la busqueda de grilla.
grilla_item = dict(k = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
             sim_options = dict(name = ['cosine', 'pearson'],
                                user_based = [False]))
```

In [131]:
```
#Se crean las bases para cada una de las alternativas
data_item = Dataset.load_from_df( ratings_artistas_merged_item[[ 'UserId', '
                            Reader(rating_scale = (min(ratings_artistas_merg
data_item2 = Dataset.load_from_df( ratings_artistas_merged_item[[ 'UserId',
                            Reader(rating_scale = (min(ratings_artistas_mer
data_item3 = Dataset.load_from_df( ratings_artistas_merged_item[[ 'UserId',
                            Reader(rating_scale = (min(ratings_artistas_mer


```

In [132]:
```
#%%time
#gs_item = GridSearchCV(KNNBasic, grilla_item, measures = ['RMSE','MAE'], cv
#gs_item.fit(data_item)
#dump(gs_item.best_params, 'best_model_item_frencuencia.joblib')

```

In [133]:
```
#%%time
#gs2_item = GridSearchCV(KNNBasic, grilla_item, measures = ['RMSE','MAE'], c
#gs2_item.fit(data_item2)
#dump(gs2_item.best_params, 'best_model_item_logfrencuencia.joblib')

```

In [134]:
```
#%%time
#gs3_item = GridSearchCV(KNNBasic, grilla_item, measures = ['RMSE','MAE'], c
#gs3_item.fit(data_item3)
#dump(gs3_item.best_params, 'best_model_item_normalizado.joblib')

```

In [135]:
```
#Guardar resultados

#best_results_freq_item = pd.DataFrame(gs_item.cv_results)
#best_results_freq_item['alternativa'] = 'frencuencia'

#best_results_log_item = pd.DataFrame(gs2_item.cv_results)
#best_results_log_item['alternativa'] = 'log_frecuencia'

#best_results_norm_item = pd.DataFrame(gs3_item.cv_results)
#best_results_norm_item['alternativa'] = 'normalizada'

#best_results_item = best_results_freq_item.append(best_results_norm_item).a
#best_results_item.to_csv('best_results_item_item.csv')
```

In [137]:
```
#Resultados
freq_item_based = load('best_model_item_frencuencia.joblib')
display(freq_item_based)

logfreq_item_based = load('best_model_item_logfrencuencia.joblib')
display(logfreq_item_based)

norm_item_based = load('best_model_item_normalizado.joblib')
norm_item_based
```

```
{'rmse': {'k': 100, 'sim_options': {'name': 'pearson', 'user_based': False}},
 'mae': {'k': 40, 'sim_options': {'name': 'cosine', 'user_based': False}}}

{'rmse': {'k': 100, 'sim_options': {'name': 'cosine', 'user_based': False}},
 'mae': {'k': 100, 'sim_options': {'name': 'cosine', 'user_based': False}}}
```

Out[137]:
```
{'rmse': {'k': 100, 'sim_options': {'name': 'cosine', 'user_based': False}},
 'mae': {'k': 20, 'sim_options': {'name': 'cosine', 'user_based': False}}}
```

In [139]:
```
best_item_item = pd.read_csv('best_results_item_item.csv').drop('Unnamed: 0'
display(best_item_item.shape)
best_item_item.head()
```

(60, 24)

Out[139]:

	split0_test_rmse	split1_test_rmse	split2_test_rmse	split3_test_rmse	split4_test_rmse	mean_test
0	153.460801	141.117203	124.856203	145.767490	142.148596	141.4
1	153.700671	141.793363	125.005914	146.700420	142.318011	141.9
2	152.619995	139.234467	122.650332	144.350983	140.120442	139.7
3	152.436374	139.293941	122.379086	144.149287	140.142718	139.6
4	152.059237	138.784611	121.896382	143.429006	139.360574	139.1

5 rows × 24 columns

In [140]:
```
#Una vez mas los mejores modelos fueron
best_item_item_sum = best_item_item[['alternativa','param_k','param_sim_opti
print('----------------------- Mejores modelos item-based ----------------
best_item_item_sum

```

----------------------- Mejores modelos item-based ----------------

Out[140]:

	alternativa	param_k	param_sim_options	mean_test_rmse
38	normalizada	100	{'name': 'cosine', 'user_based': False}	0.439530
58	log_frecuencia	100	{'name': 'cosine', 'user_based': False}	0.640830
19	frencuencia	100	{'name': 'pearson', 'user_based': False}	137.335848

In [141]:

```
for each in ['frencuencia','log_frecuencia','normalizada']:
    temp = best_item_item[['param_k','param_sim_options','mean_test_rmse','a
    sns.lineplot(data = temp, x = 'param_k', y = 'mean_test_rmse', hue = 'pa
    plt.title(each)
    plt.show()
```