

Experimentación, análisis e implementación de sistemas de recomendación basado en filtrado colaborativo

Santiago Cortés Fernández, Rogelio García Escallón, Nicolás M. Hernández Rojas

Hoja de Trabajo – Reporte Técnico

Universidad de los Andes, Bogotá, Colombia

{s.cortes, r.garcia11, nm.hernandez10}@uniandes.edu.co

Fecha de presentación: enero 31 de 2019

Tabla de contenido

- 1 Introducción
- 2 Conocimiento del *dataset* de trabajo
- 3 Pre-procesamiento de datos
- 4 Construcción de modelos colaborativos usuario-usuario
- 5 Construcción de modelos colaborativos ítem-ítem
- 6 Análisis de resultados

1. Introducción

En este taller se va a experimentar con modelos colaborativos para sistemas de recomendación utilizando un *dataset* de recomendación de música obtenido en <https://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/lastfm-1K.html>.

El siguiente informe incluye la explicación de las decisiones tomadas en cuanto a qué técnicas utilizar, comparación entre ellas, y decisiones de diseño, e incluye también el análisis de los resultados obtenidos. Los ítems a recomendar son **artistas**.

El proyecto se organiza en las carpetas “preprocessing”, “experiments”, “results” y “webpage”. Adicional a estas carpetas está este documento y un archivo “requirements.txt” con los requerimientos para Python. Por último está el archivo README.md que puede ser mejor visualizado en el repositorio de github <https://github.com/los-de-pregrado/CFRecommenderSystems>.

2. Conocimiento del *dataset* de trabajo

El *dataset* incluye la información de los hábitos de reproducción de los usuarios de *Last.fm*^[1] de cerca de 1000 usuarios hasta el 5 de mayo del 2009, e información sobre los

usuarios en dos tablas separadas: *userid-profile* y *userid-timestamp-artid-artname-traid-traname.tsv*.

Las entradas de datos de *userid-timestamp-artid-artname-traid-traname.tsv* relacionan un usuario con una fecha y hora (una marca temporal o un *timestamp*), un artista y una canción. El formato en el que están los datos es .tsv. La organización por columnas es <id del usuario, *timestamp*, id del artista, id de la canción>, donde el identificador tanto del artista como de la canción es un MBID. Un MBID significa *MusicBrainz Identifiers* y cada MBID es único para una canción o un artista. La siguiente tabla muestra un breve perfilamiento de los datos:

| Columna | ¿Contiene nulos? | Tipo de valores |
|------------|------------------|-----------------|
| userId | No | Cadena de texto |
| timestamp | No | Cadena de texto |
| artistId | Sí | Cadena de texto |
| artistName | Sí | Cadena de texto |
| trackId | Sí | Cadena de texto |
| trackName | Sí | Cadena de texto |

Tabla 1: Breve perfilamiento de datos

La primera decisión que se tomó al preprocesar los datos es eliminar las filas que contengan nulos en alguno de los identificadores, ya que es lo que se utiliza para generar el sistema de filtro colaborativo. En total se eliminaron XXXX filas por este motivo. También se eliminaron las filas que presentaran problemas de formato (más separadores de los necesarios, etc.). 9 filas se eliminaron de esta manera.

Esta tabla no incluye un *rating* del usuario a la canción o al artista, es únicamente información de hábitos de reproducción; un ejemplo se muestra a continuación.

```
userid-timestamp-artid-artname-traid-traname.tsv:
user_000639 2009-04-08T01:57:47Z 15676fc4-ba0b-4871-ac8d-ef058895b075 The Dogs D'Amour
6cc252d0-3f42-4fd3-a70f-c8ff8b693aa4 How Do You Fall in Love Again
user_000639 2009-04-08T01:53:56Z 15676fc4-ba0b-4871-ac8d-ef058895b075 The Dogs D'Amour
aa7dbea2-a0c0-4d0a-9241-5bb98a372b11 Wait Until I'm Dead
...
```

Figura 1. Ejemplo de fila de los datos en *userid-timestamp-artid-artname-traid-traname.tsv*.

Hay 83905 artistas diferentes que para efectos del modelo de sistemas de recomendación, son los ítems a recomendar.

La información de la tabla en *userid-profile* incluye información de los usuarios y un breve perfilamiento de ellos. Incluye el *id* del usuario, género, edad, país y fecha de registro. A continuación, se muestra el perfilamiento de estos datos y un ejemplo:

| Columna | ¿Contiene nulos? | Número de valores no nulos | Número de valores únicos | Tipo de valores |
|---------|------------------|----------------------------|--------------------------|---|
| #id | No | 992 | 992 | Valores consecutivos desde user_000001 hasta user_001000, con algunos faltantes |

| | | | | |
|------------|----|-----|-----|--|
| gender | Sí | 884 | 2 | M o F (o vacío) |
| age | Sí | 286 | 34 | Número entre 3 y 103, con algunos faltantes y vacíos |
| country | Sí | 907 | 66 | Nombre del país. Hay varios países que solo tienen un usuario, 25% de los usuarios son de Estados Unidos, 14% del Reino Unido, 5,5% de Polonia. El resto están todos debajo de 4%. |
| registered | Sí | 984 | 625 | Fecha entre el 29 de octubre del 2002 y el 5 de noviembre del 2009, con algunos faltantes. |

```
userid-profile.tsv:
  user_000639      m           Mexico    Apr 27, 2005
  ...
```

Figura 2. Ejemplo de fila en los datos en *userid-profile*

3. Pre-procesamiento de datos

El preprocesamiento de datos se realizó con un archivo llamado *data_preprocessing.py*, y el resultado de este preprocesamiento se guardó en el archivo *list.csv*. También se utilizaron subsets de esto para evaluar el funcionamiento general de la aplicación y realizar comparaciones pequeñas, en los archivos *list_01.csv*, *list_1.csv* y *list_1000a.csv*.

Como no se tienen *ratings*, se van a definir según la interacción que tenga un usuario con los artistas. Para ello, se definió utilizar una función logarítmica que incluye el artista que más fue escuchado, de la forma: $\text{RatingArtista} = \text{LogMaxReproducido}(\text{ReproduccionesArtista}) * 4 + 1$, si *ReproduccionesArtista* no es 0; en caso contrario, dejar este valor como 0 (esto ocurre cuando el usuario nunca ha escuchado al artista).

Esta fórmula devuelve un número entre 1.0 y 5.0. Se decidió así para no tener un comportamiento lineal que se vea sesgado a un artista con muchas reproducciones. Por ejemplo, usuario a escuchó el artista x un total de 1000 veces, y escuchó al grupo y un total de 50 veces. Si bien 50 reproducciones son relevantes, en una escala lineal se le daría un rating muy bajo.

Luego se formó una matriz donde cada columna es un usuario y cada fila un artista, y el contenido de la tabla es el rating que le da un usuario a un artista dado. Dado que esta matriz contiene muchos valores en cero, se formó una nueva matriz de tres columnas *userId*, *artistId*, *rating*, que reduce considerablemente el espacio necesario para almacenarla y sirve como entrada para un sistema de recomendación en la librería *Surprise*.

Los datos se separaron en dos conjuntos luego del pre-procesamiento, 25% de los datos se reservaron como *dataset* de pruebas. Esta separación se realiza con las siguientes instrucciones en *Surprise*, específicamente *surprise.model_selection.train_test_split*.

```
import pandas as pd

from surprise.model_selection import train_test_split
from surprise import Dataset
from surprise import Reader

df = pd.read_csv("./list.csv", delimiter=",", encoding="utf-8", error_bad_lines=False)
reader = Reader(rating_scale=(1.0, 5.0))
data = Dataset.load_from_df(df[['userId', 'artistId', 'rating']], reader)
train_set, test_set = train_test_split(data, test_size=.25)
```

4. Construcción de modelos colaborativos usuario usuario

Los experimentos mostrados a continuación parten de los archivos .py en la carpeta “experiments”. Los archivos “jaccard_sim.py”, “mcloughlin_sim.py” son las funciones para calcular las matrices de similitud de estos dos métodos. “models.py” incluye la implementación del algoritmo KNNBasic de Surprise. El resto de archivos son, según su nombre, experimentos que retornan una gráfica. Las gráficas están guardadas en la carpeta “results”.

a.

El modelo se construyó utilizando la librería surprise de Python. Para construir el modelo, se debe definir un algoritmo de predicción, en nuestro caso KNNBasic. KNNBasic recibe como parámetros el número máximo de vecinos, el nombre de una función de similitud y si es usuario-usuario o ítem-ítem.

```
from surprise import KNNBasic
sim_options = {
    'name': 'pearson',
    'user_based': True,
}
algorithm = KNNBasic(k=kValue, sim_options=sim_options)
algorithm.fit(train_set)
```

b.

Para predecir la relevancia de los ítems se utiliza la función *test*.

```
predictions = algorithm.test(test_set)
```

Esta función arroja predicciones con el identificador de usuarios *uid*, identificador del ítem *iid*, el *rating* real y el estimado, y otras informaciones. También incluye un campo para saber si fue posible predecir el rating para el ítem dado.

```
Prediction(uid='user_000500', iid='2870dffbf336-4a7b-96c0-
eecf417c7fed', r_ui=1.0, est=1.8971555182565383, details={'was_impossible': True, 'reason':
'User and/or item is unknown.'})
```

```
Prediction(uid='user_000722', iid='d4a1404d-e00c-4bac-b3ba-
e3557f6468d6', r_ui=3.049423276345537, est=1.886442260279148, details={'actual_k': 5,
'was_impossible': False})
```

Adicionalmente, se definió una función que recibe las predicciones y un número entero *n* y retorna un diccionario que contiene una lista con las primeras *n* predicciones para cada usuario en orden del rating predicho, en forma de tupla (*itemId*, *rating*).

```
'user_000463': [(('32b90c92-9978-4a07-90eb-caa4b22f4907', 3.5106869703792136),
('d4d17620-fd97-4574-92a8-a2cb7e72ce42', 3.464256641763165), ('9de8f66e-3cd1-4f11-
8328-38200f0612b0', 3.1446878961364217), ('cc0b7089-c08d-4c10-b6b0-873582c17fd6',
2.9901818013363446), ('f181961b-20f7-459e-89de-920ef03c7ed0', 2.728503442821299),
('8c538f11-c141-4588-8ecb-931083524186', 2.6670920880573963), ('970fb29f-e288-403e-
a388-d2a7889bfa47', 2.643524734521892), ('8434409e-baa9-4e12-b4aa-566a91c7d7cf',
2.480210102296422), ('4efa55ba-93cf-497f-baf3-2ca9da7e193e', 2.4473736580596714),
('77f049ad-f469-4ad1-8283-7a2606a6722e', 2.436338420952059)]
```

Para implementar los tres índices de similitud, y para algunas partes de este taller, se volvió a implementar el modelo *KNNBasic*. La clase *KNNBasic* hereda de *SymmetricAlgoJaccard*, que a su vez hereda de *AlgoBase*. Para poder solucionar varios puntos del taller, se implementó *KNNBasic* con una ligera modificación para retornar vecinos según un umbral, y se implementó *SymmetricAlgoJaccard* con una modificación que anula o sobrescribe la función *compute_similarities*, que retorna una matriz de similitud usuario-usuario (o ítem-ítem).

Todos los modelos utilizados son creados con la clase *KNNBasic* con un parámetro *sim_options* diferente, que se traduce a una implementación diferente de *compute_similarities*. Esta función utiliza otra función que calcula la matriz de similitud entre cada usuario (o cada ítem). Coseno y Pearson son importados de *similarities*, mientras que jaccard fue implementado en *jaccard_sim.py*.

```
construction_func = {'cosine': sims.cosine,
                      'pearson': sims.pearson,
                      'jaccard': jaccard}
```

- i. Para el índice de Jaccard fue necesario definir una nueva función de similitud. Esta similitud se calcula como:

$$sim(a, b) = \frac{A \cap B}{A \cup B} = \frac{A \cap B}{|A| + |B| - A \cap B}$$

De esta manera, durante el cálculo solo se mantiene una cuenta del número de artistas a los que ha calificado el usuario y la cuenta del número de artistas que los dos usuarios.

Al crear el modelo, se utiliza el siguiente parámetro:

```
sim_options_jaccard = {  
    'name': 'jaccard',  
    'user_based': True  
}
```

- ii. Un modelo basado en distancias coseno es fácil de lograr con *Surprise*. El parámetro *sim_options* que entra como parámetro en *KNNBasic* lo define con la opción name.

```
sim_options = {  
    'name': 'cosine',  
    'user_based': True,  
}
```

- iii. Un modelo basado en correlación de Pearson también es fácil de lograr con *Surprise*:

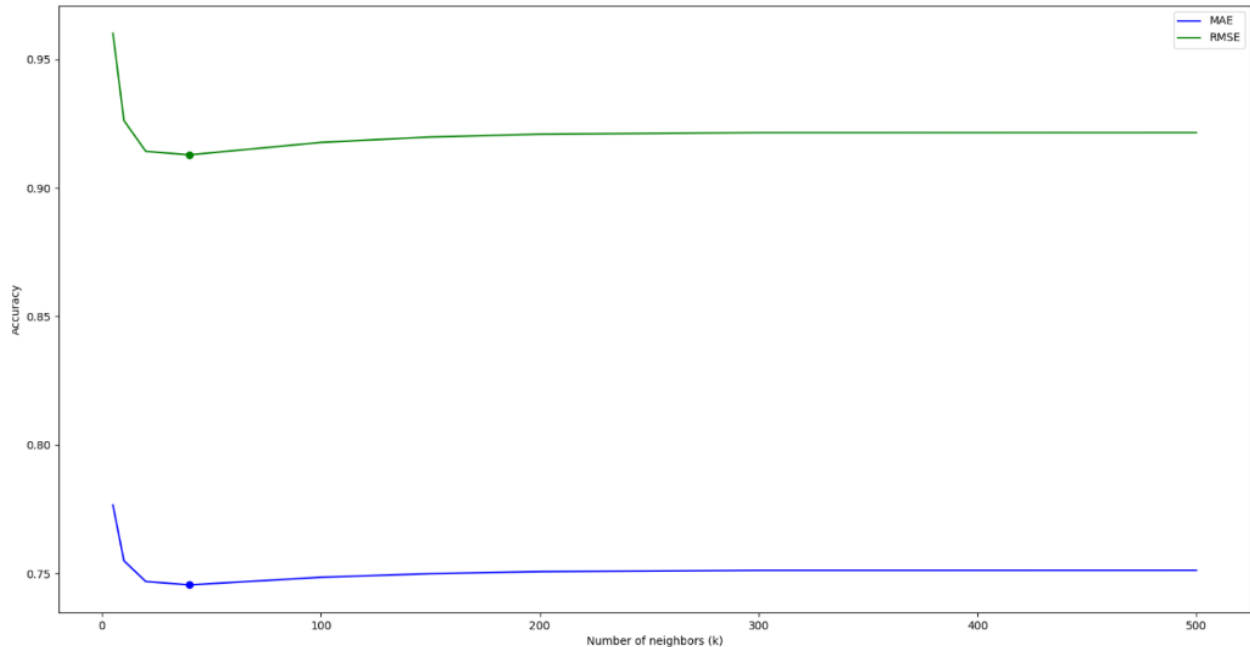
```
sim_options = {  
    'name': 'pearson',  
    'user_based': True,  
}
```

- c. Para evaluar el comportamiento del modelo y cómo se está ajustando a los datos, se utilizaron las métricas RMSE (raíz del error cuadrático medio) y MAE (error absoluto medio).

$$\text{MAE} = \frac{1}{|\hat{R}|} \sum_{\hat{r}_{ui} \in \hat{R}} |r_{ui} - \hat{r}_{ui}|$$

$$\text{RMSE} = \sqrt{\frac{1}{|\hat{R}|} \sum_{\hat{r}_{ui} \in \hat{R}} (r_{ui} - \hat{r}_{ui})^2}.$$

Experimentalmente se encontró que utilizando cualquiera de estas métricas los valores óptimos no varían demasiado. La siguiente figura muestra la determinación del número de vecinos óptima para el caso de la función de similitud de Pearson; varía en menos de 5 vecinos el valor óptimo.



Conceptualmente, RMSE castiga más a un modelo por contener predicciones muy alejadas de los valores reales. Como no se quiere recomendar un ítem demasiado alejado de los gustos de la persona, se optó por utilizar RMSE para evaluar los diferentes modelos colaborativos.

En *Surprise* se pueden calcular tres métricas diferentes: FCP (fracción de parejas concordantes), MAE y RMSE. Cada una se puede calcular así:

```
fcp = accuracy.fcp(predictions)
mae = accuracy.mae(predictions)
rmse = accuracy.rmse(predictions)
```

d.

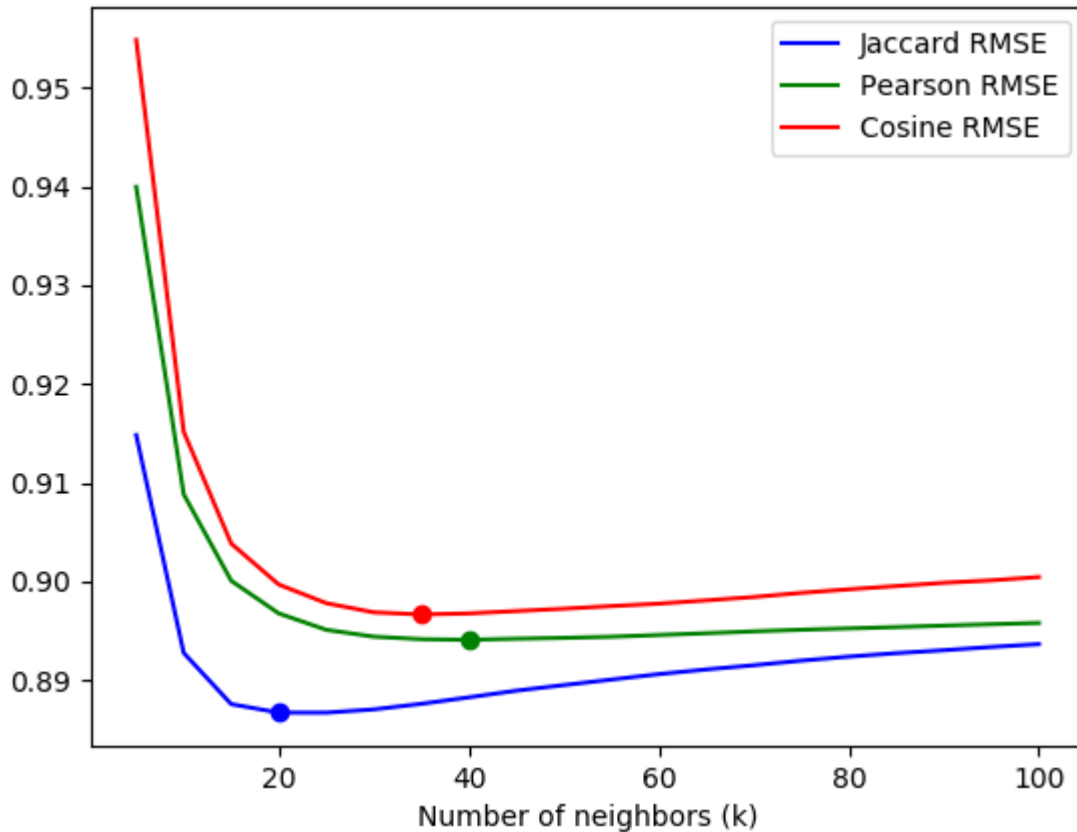
Se diseñó una estrategia para variar el parámetro del número de vecinos al definir el algoritmo a utilizar. En este caso es el valor *kValue*. Se itera sobre este valor y se analizan los resultados cada vez, guardando el valor para *kValue* que menor error obtuvo, por cada algoritmo analizado.

```
from surprise import KNNBasic
sim_options = {
    'name': 'pearson',
    'user_based': True,
}
algorithm = KNNBasic(k=kValue, sim_options=sim_options)
algorithm.fit(train_set)
```

En la función *estimate*, de la clase *KNNBasic*, se forma un montículo con todos los vecinos y se retorna los k primeros, de la siguiente manera:

```
neighbors = [(self.sim[x, x2], r) for (x2, r) in self.yr[y]]
k_neighbors = heapq.nlargest(self.k, neighbors, key=lambda t: t[0])
```

Se experimentó con un arreglo de k_values de 5 a 100, en intervalos de 5. Para los tres índices de similitud utilizados hay un valor óptimo para el número de vecinos en el que el error es mínimo. A mayor o menor número de vecinos, el error aumenta. Graficamos el número óptimo de vecinos, y para los tres algoritmos está entre 20 y 40, con el menor error siendo de Jaccard.



En la siguiente table se resumen los datos de varias ejecuciones y sus valores. En cada ejecución el test set cambia, por lo que los resultados varían. Sin embargo, no son muy diferentes.

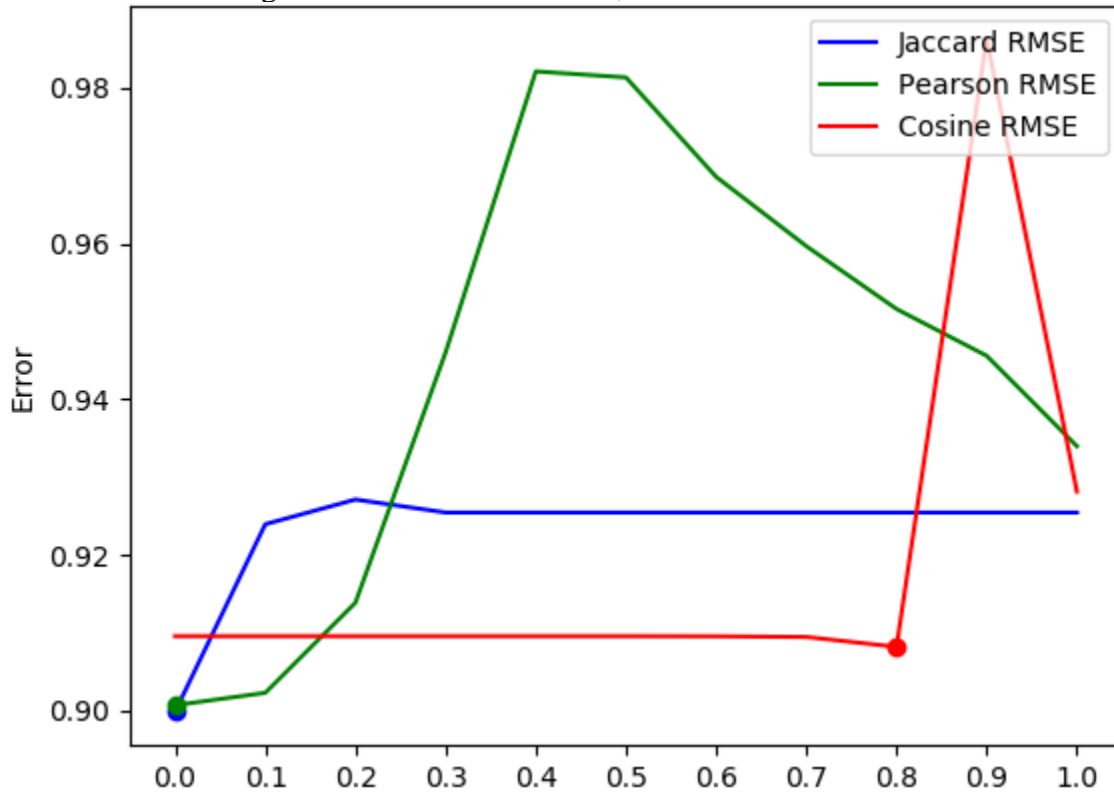
| | K-vecinos óptimos | | | | | | | | | | | |
|---------|-------------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|--------------|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Promedio | Aproximación |
| Cosine | 35 | 30 | 30 | 30 | 35 | 35 | 35 | 30 | 40 | 35 | 33,5 | 35 |
| Pearson | 35 | 35 | 40 | 40 | 35 | 40 | 40 | 40 | 35 | 40 | 38 | 40 |
| Jaccard | 25 | 20 | 25 | 20 | 20 | 20 | 20 | 20 | 25 | 25 | 22 | 20 |
| | Error | | | | | | | | | | | |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Promedio | Aproximación |
| Cosine | 0,899 | 0,903 | 0,896 | 0,905 | 0,9 | 0,901 | 0,902 | 0,905 | 0,905 | 0,903 | 0,9019 | 0,900 |
| Pearson | 0,894 | 0,89 | 0,894 | 0,894 | 0,899 | 0,896 | 0,89 | 0,894 | 0,89 | 0,891 | 0,8932 | 0,895 |
| Jaccard | 0,894 | 0,888 | 0,889 | 0,885 | 0,894 | 0,888 | 0,893 | 0,889 | 0,892 | 0,886 | 0,8898 | 0,890 |

También se definió una estrategia para analizar los resultados por umbral de similitud. Para ello, en *KNNBasic* se definió un nuevo parámetro opcional *threshold* con valor de 0 por defecto, y al seleccionar los vecinos de un usuario en la función *estimate*, utilizará solo la lista de los vecinos cuya similitud supere un cierto umbral:

```
for threshold in thresholds:
    jaccard_algorithm = KNNBasic(k=k_value, threshold=threshold, sim_options=sim_options_jaccard)
    jaccard_algorithm.fit(train_set)
    jaccard_predictions = jaccard_algorithm.test(test_set)
```

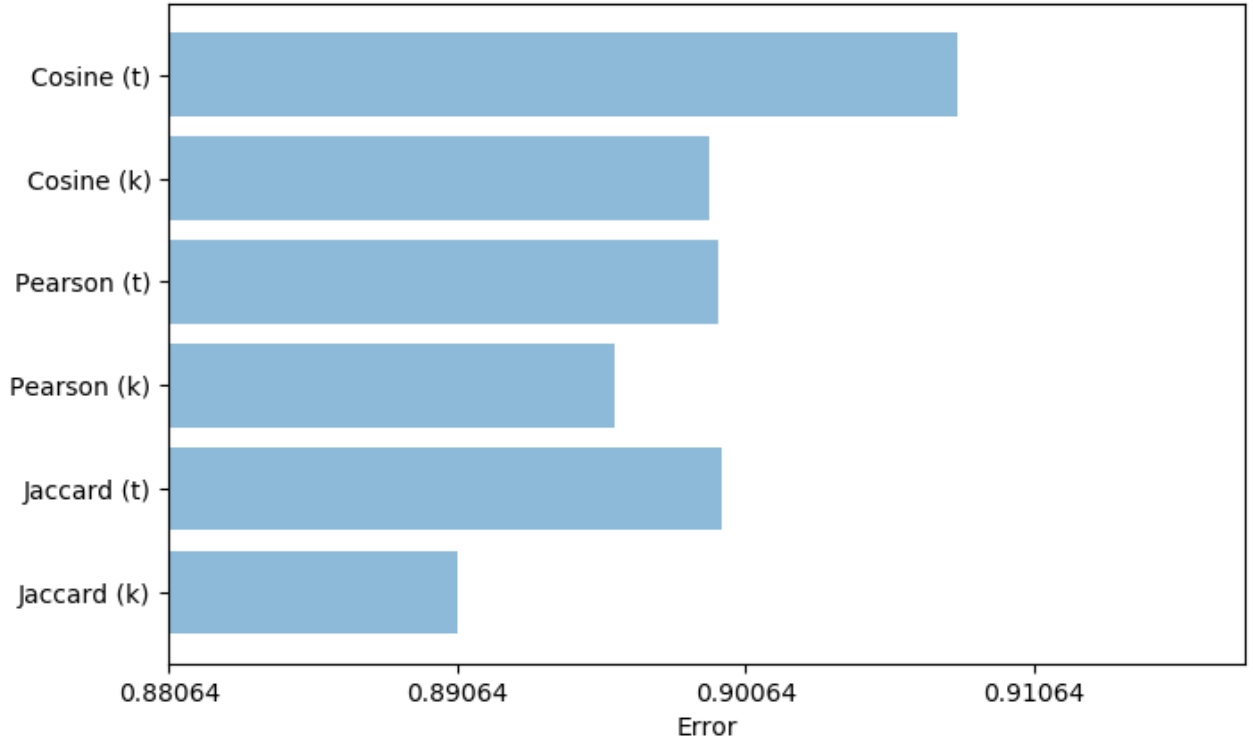
```
neighbors = [(self.sim[x, x2], r) for (x2, r) in self.yr[y]] if self.sim[x, x2] > self.threshold]
```

Se experimentó con un arreglo de thresholds entre 0 y 1 en intervalos de 0.1 y se graficó el resultado. El resultado se graficó de la misma manera, mostrando a Jaccard con el menor error:



Como se encontró que teniendo un umbral de similitud entre usuarios de 0.0, es decir, aceptando todos los vecinos, se tuvo mejor precisión en el modelo, se realizó un experimento entre los mejores parámetros encontrados para número de vecinos y para umbral.

```
k_value_jaccard = 20
k_value_pearson = 40
k_value_cosine = 35
threshold_jaccard = 0.0
threshold_pearson = 0.0
threshold_cosine = 0.8
```



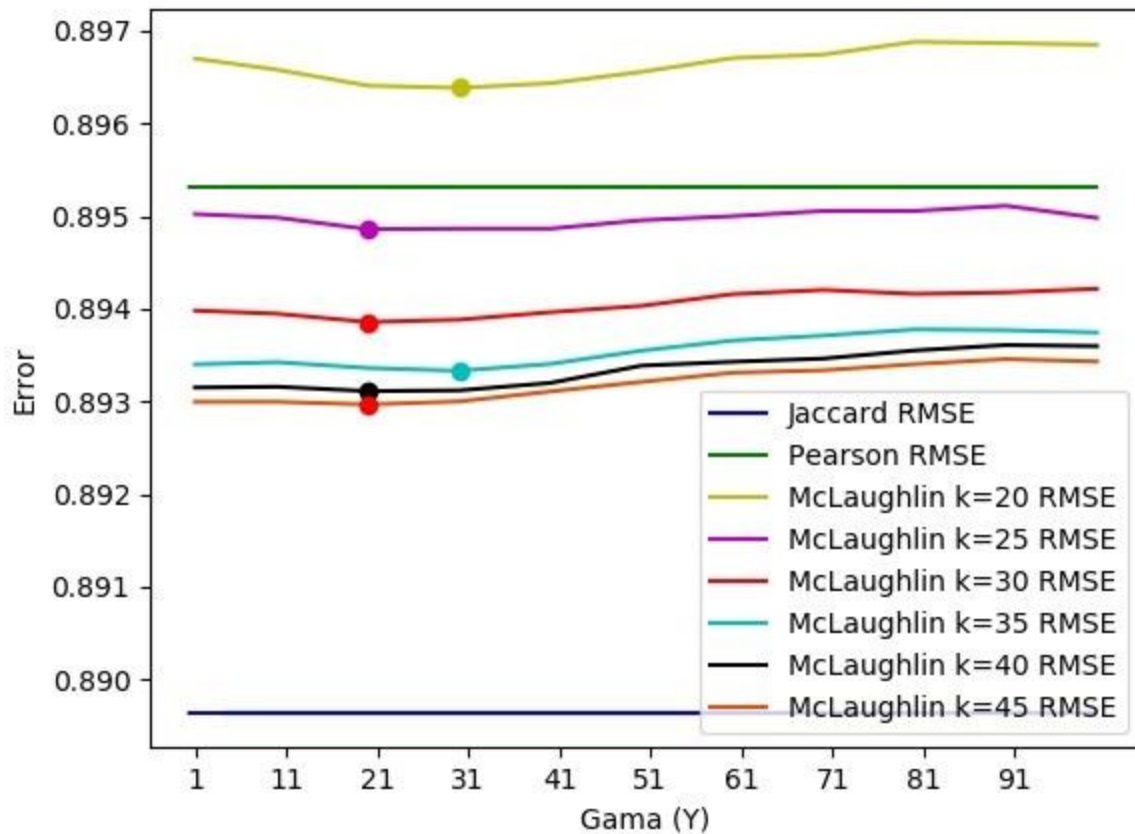
En esta gráfica se compararon los modelos con los mejores parámetros para número de vecinos y para umbral de similitud. El experimento fue realizado reiteradamente y de manera consistente, Jaccard con $k=20$ fue el modelo con menor error.

La fórmula de McLaughlin fue implementada de la misma manera en que se implementó Jaccard. Se definió una nueva forma de calcular la similitud, agregando la corrección de Herlocker y colaboradores,

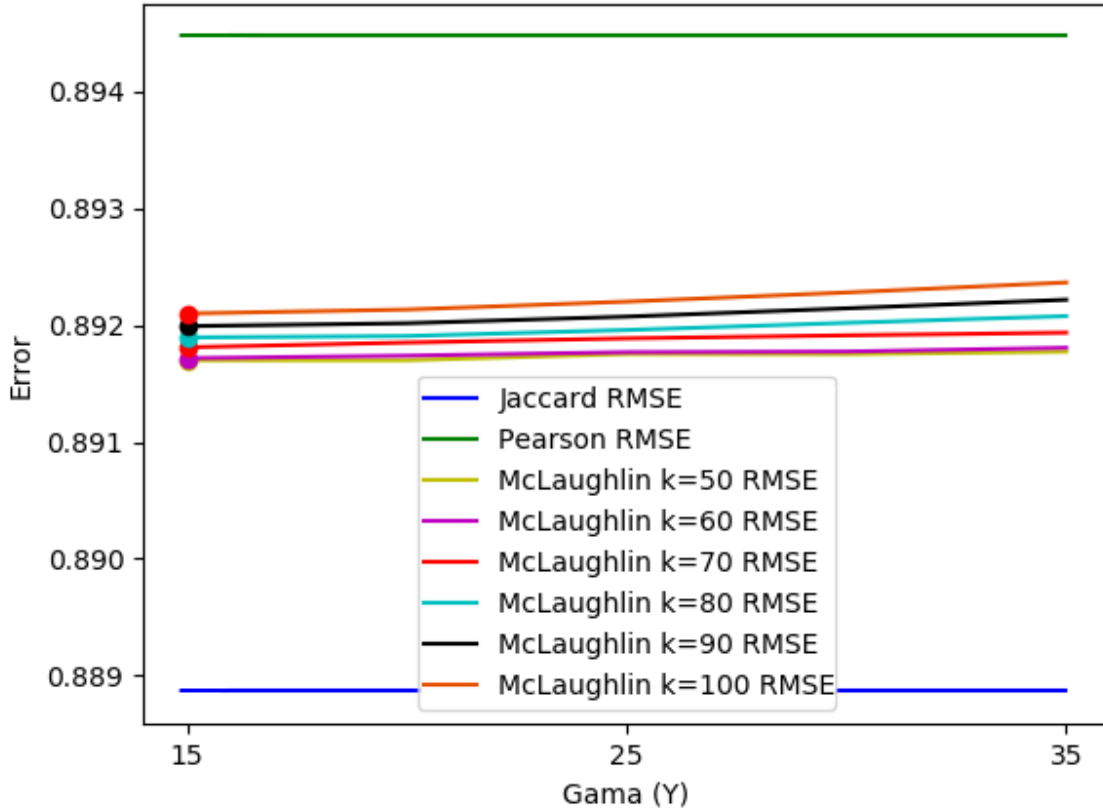
$$sim'(u, v) = \frac{\max(|I_u \cap I_v|, \gamma)}{\gamma} sim(u, v)$$

que incluye una corrección cuando los ítems que han calificado dos usuarios en común es mayor a un valor gama definido. Esto hace que estos usuarios vean su similitud aumentada por el hecho de haber calificado ítems en común. El valor de gama óptimo se debe definir mediante experimentación, y como McLaughlin sigue siendo un algoritmo que hereda de KNNBasic, también es necesario definir un valor óptimo para k . Se optó por experimentar únicamente con el número de vecinos y no con un umbral dado que en el experimento anterior se observó que variar el número de vecinos logra un error menor.

El primer experimento fue variando el valor de k entre 20 y 45 en intervalos de 5, y gamma entre 1 y 100 en intervalos de 10. Jaccard y Pearson también se graficaron para $k=20$ y $k=35$ respectivamente, que son los valores óptimos encontrados en el punto 3.d.



Se puede observar que el error disminuye a medida que aumenta el valor de k. Para k=45, el gamma óptimo está entre 20 y 30. Se repitió con valores de más de 45 para descartar que siguiera disminuyendo el error. En este caso solo se graficó McLaughlin para valores de 50 hasta 100 en intervalos de 10.



Se puede observar que el mínimo error lo tiene McLaughlin con $k=50$, y a partir de este punto aumenta.

El paper de McLaughlin también habla de que utilizar la métrica MAE no es óptimo dado que puede que algunos errores grandes se escondan con la métrica, resultando en un MAE pequeño pero errores graves. En este caso se está utilizando RMSE, que es similar a MAE pero castiga precisamente los valores muy alejados del real. La métrica que se propone es tomar en cuenta la fracción de ítems bien catalogados. En *Surprise* se incluye una métrica, llamada FCP — *Fraction of Concordant Pairs*, que toma en cuenta si un ítem se catalogó bien **en relación** a los demás ítems. Se utilizó FCP para comprar los resultados dado que esta métrica busca evitar tener en cuenta únicamente la distancia del valor predicho al valor real, como lo hacen MAE y RMSE.

Primero se calcula el número de parejas que concuerdan en el ranking real, n_c para un usuario u , n_c^u , para todo ítem i y j , tomando los valores predichos para el usuario u y los ítems i, j , p_{ui} y p_{uj} , respectivamente, y de forma similar, los valores reales r_{ui} y r_{uj} .

$$n_c^u = |\{(i, j) \mid p_{ui} > p_{uj} \wedge r_{ui} > r_{uj}\}|$$

De la misma forma se calculan las parejas que no concuerdan, n_d^u :

$$n_d^u = |\{i, j\} \mid \neg(p_{ui} > p_{uj} \wedge r_{ui} > r_{uj})|$$

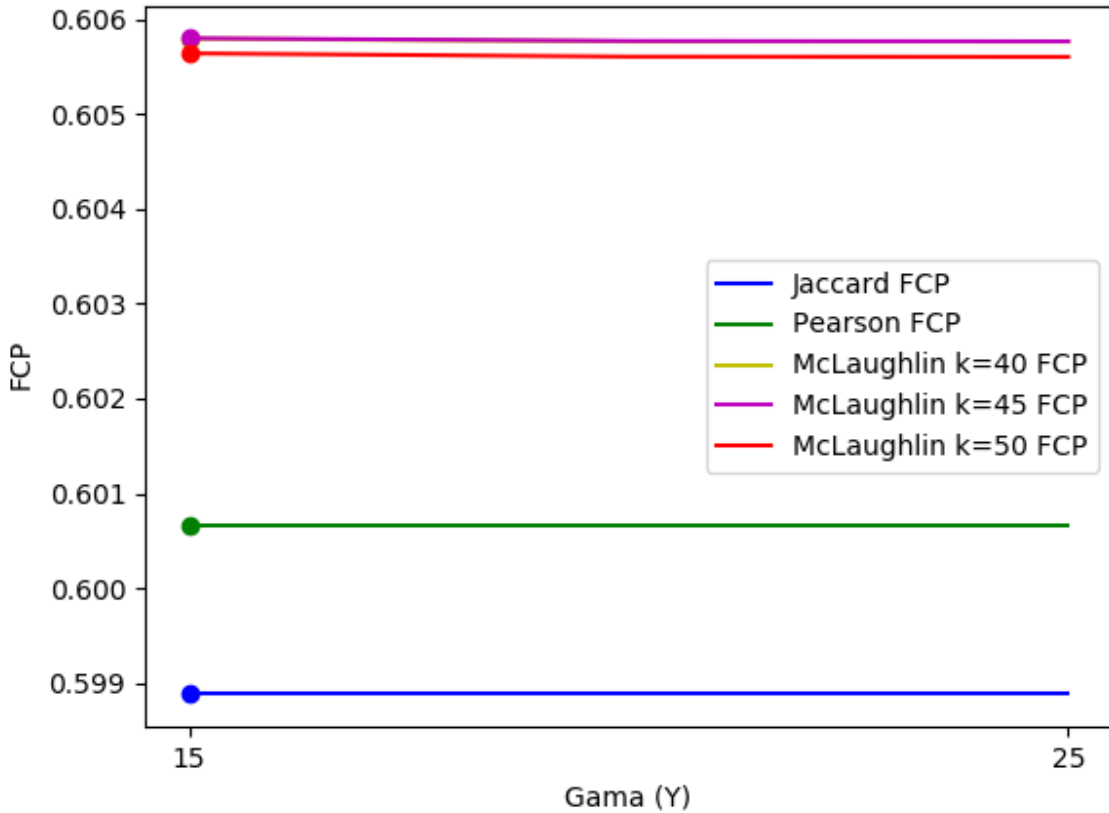
Con estos dos valores, se suma sobre todos los usuarios.

$$n_c = \sum_u n_c^u, n_d = \sum_u n_d^u$$

Por último, se calcula la métrica,

$$FCP = \frac{n_c}{n_c + n_d}$$

El experimento se realizó con los valores que tuvieron menor error según RMSE, k entre 40 y 50 en intervalos de 5 y gammas entre 15 y 25 en intervalos de 5. Jaccard y Pearson también se incluyeron en la gráfica. Se debe tener en cuenta que FCP, al contrario de RMSE y MAE, se leen al contrario; un valor mayor de FCP significa una mayor precisión.



Efectivamente se invierte el orden, y en este caso Jaccard, que había tenido el menor error consistentemente en todos los experimentos, tiene la menor FCP. McLaughlin con k=40 y k=45 tuvieron la misma precisión, y el gamma entre estos valores no afectó la FCP.

5. Construcción de modelos colaborativos ítem ítem

Dado que el cambio para realizar pruebas con la técnica de filtrado colaborativo ítem-ítem es muy pequeño, se utilizaron los mismos archivos. El otro cambio es que se utilizó una submuestra del archivo completo de calificaciones dado que la matriz completa no podía caber en memoria.

a.

Para construir el modelo, se utilizó la misma librería que en el caso de filtrado colaborativo usuario usuario. El parámetro al definir el algoritmo KNNBasic llamado `sim_options` puede tomar un valor llamado `user_based`, de la siguiente manera:

```
sim_options_jaccard = {  
    'name': 'jaccard',  
    'user_based': False,  
}
```

De manera similar, se definió un parámetro `sim_options_cosine` y `sim_options_pearson`.

b.

Predecir la relevancia de los ítems para cada usuario funciona de igual manera para filtrado ítem-ítem que para usuario-usuario y se obtiene el mismo output de la función.

```
predictions = algorithm.test(test_set)
```

No hubo modificaciones en cuanto a la manera de implementar las tres métricas de similitud, lo único que cambia es el parámetro mencionado en el punto anterior. El parámetro `sim_options` se define con la opción `user_based` con valor `False`. Luego se inicializa el algoritmo así (tomando Jaccard como ejemplo):

```
jaccard_algorithm = KNNBasic(k=k_value, sim_options=sim_options_jaccard)
```

Luego se llama la función `fit`.

```
jaccard_algorithm.fit(train_set)
```

La función `fit` de `KNNBasic` llama la función `fit` de `SymmetricAlgo` define el “x” que utiliza el cálculo de similitudes. Esto hace el papel de transponer la matriz para hacer filtrado por usuarios o por ítems.

```
ub = self.sim_options['user_based']
self.n_x = self.trainset.n_users if ub else self.trainset.n_items
self.n_y = self.trainset.n_items if ub else self.trainset.n_users
self.xr = self.trainset.ur if ub else self.trainset.ir
self.yr = self.trainset.ir if ub else self.trainset.ur
```

El atributo trainset es definido en AlgoBase, y solamente toma el valor del train_set que fue pasado como parámetro a la función fit.

También, al llamar la función estimate se utiliza una función switch de la clase SymmetricAlgo, que intercambia la x y la y para efectos de estimar las calificaciones.

```
x, y = self.switch(u, i)
```

Estas son las únicas diferencias entre la implementación de filtrado por usuarios y por ítems, dado que son simétricos.

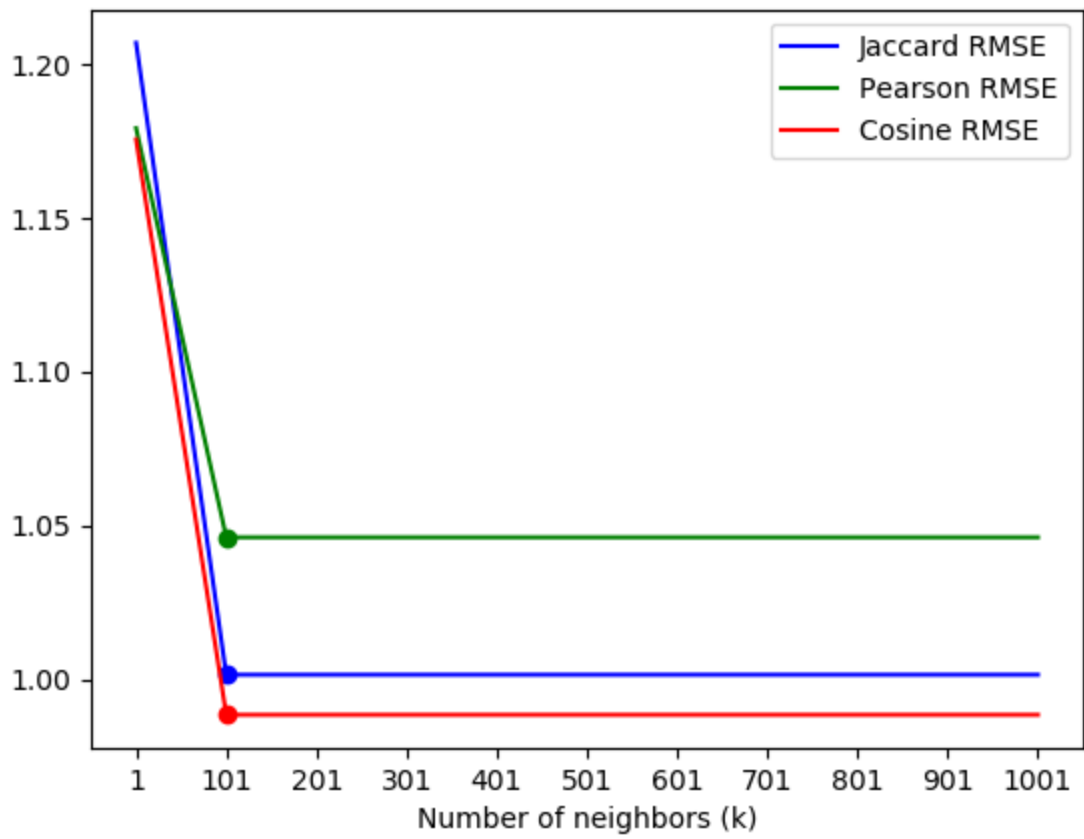
c.

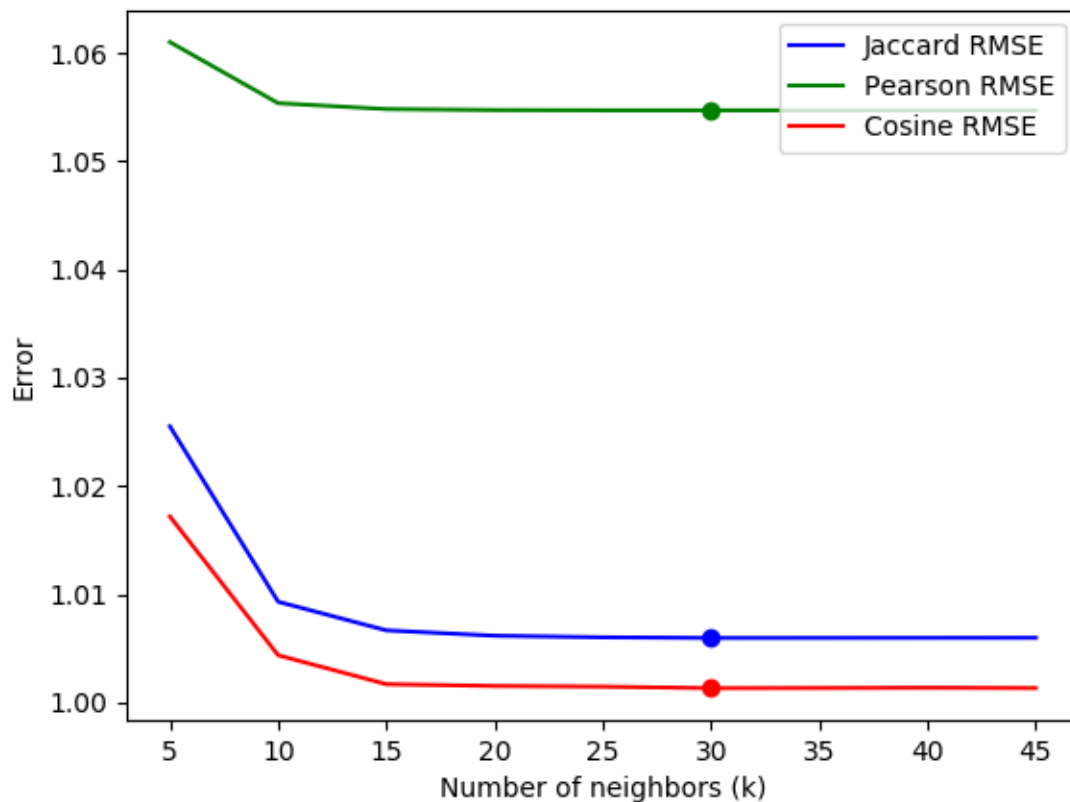
Para ser consistentes con la experimentación, también se utilizará RMSE.

d.

La estrategia para variar los parámetros es la misma, sin embargo, los valores de k sí cambiaron.

En la siguiente figura se muestran valores para k de entre 1 y 1000. Luego se intentó determinar en qué punto el error deja de variar; este punto cambia mucho cada vez que se ejecuta el script. En la tabla se resumen los valores encontrados tras 10 intentos.



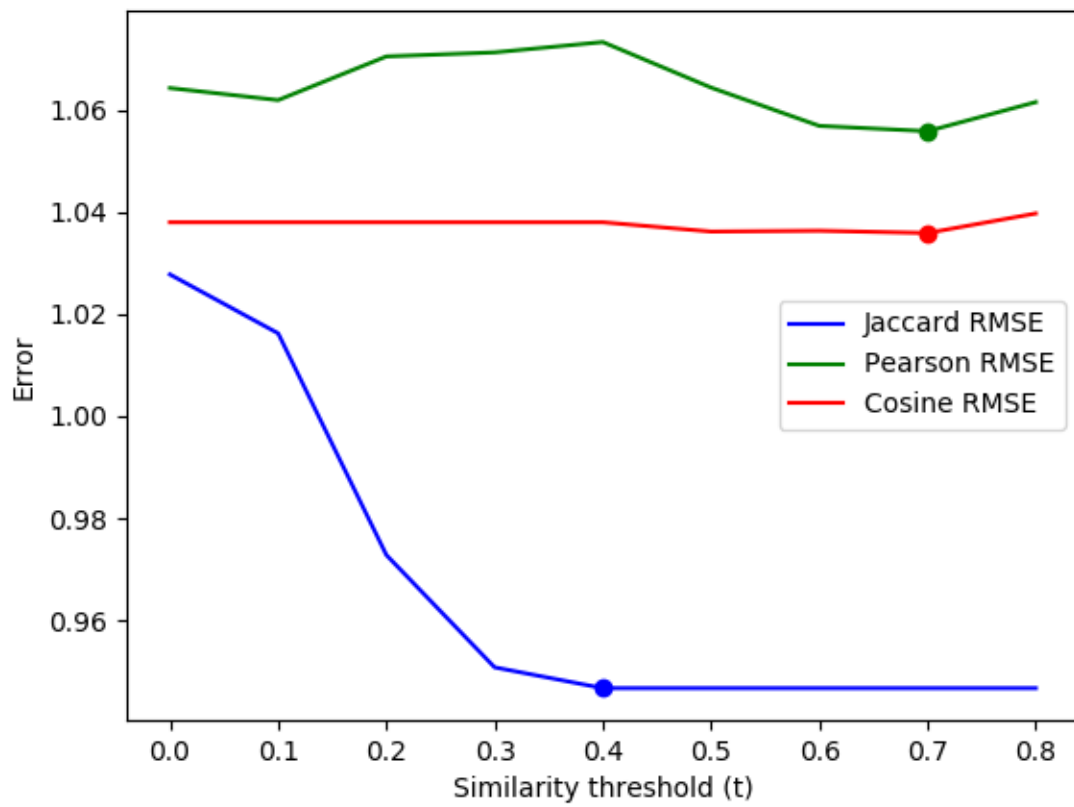


Luego de experimentar con los valores óptimos, se llegó a este resumen:

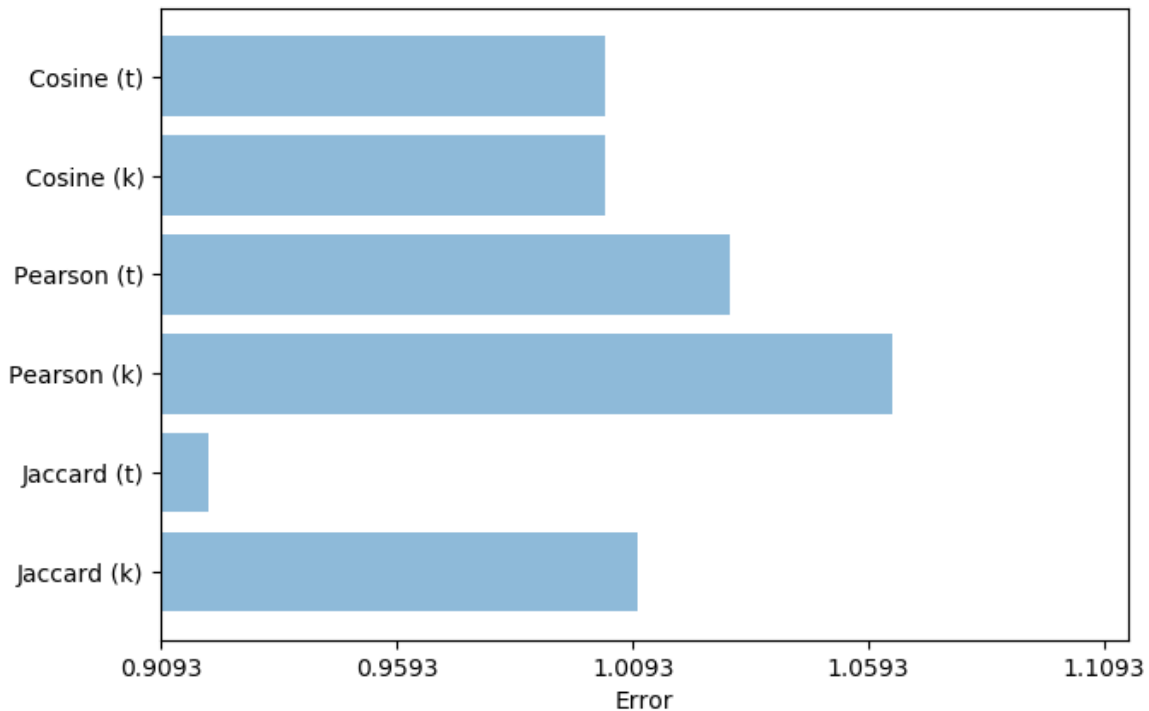
| | K-vecinos óptimos | | | | | | | | | | | |
|---------|-------------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|----------|--------------|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Promedio | Aproximación |
| Cosine | 35 | 45 | 40 | 45 | 35 | 40 | 40 | 35 | 40 | 45 | 40 | 40 |
| Pearson | 20 | 30 | 30 | 25 | 30 | 35 | 30 | 25 | 30 | 20 | 27,5 | 30 |
| Jaccard | 15 | 15 | 25 | 25 | 20 | 20 | 15 | 20 | 25 | 15 | 19,5 | 20 |
| | Error | | | | | | | | | | | |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Promedio | Aproximación |
| Cosine | 0,9921 | 0,9938 | 1,0113 | 0,9929 | 0,9875 | 0,9994 | 0,9934 | 0,9895 | 1,0013 | 0,9954 | 0,9956 | 0,996 |
| Pearson | 1,0467 | 1,0515 | 1,0542 | 1,0491 | 1,0558 | 1,0550 | 1,0060 | 1,0559 | 1,0554 | 1,0547 | 1,0484 | 1,050 |
| Jaccard | 0,9913 | 0,9929 | 1,0199 | 0,9921 | 0,9989 | 0,9892 | 0,9899 | 0,9896 | 0,9894 | 0,9893 | 0,9942 | 0,994 |

En general, Jaccard sigue teniendo menos error, aunque la diferencia no es muy grande con coseno. Pearson, que en el filtrado usuario-usuario tuvo una mejor precisión que coseno, en este caso está por debajo. El número óptimo de vecinos para Jaccard, Pearson y coseno son 20, 30 y 40, respectivamente, sin embargo, como siempre después de este número óptimo el error permanece igual, se va a optar por utilizar 30, 40 y 50 como valores óptimos de k.

Con la estrategia de cambiar un threshold, se logró tener el umbral óptimo para cada algoritmo:



Luego de obtener los parámetros óptimos, se realiza una comparación entre thresholding y los k vecinos más similares.



En esta ocasión, el algoritmo con menor error es Jaccard con un umbral de similitud de 0,4. Las dos funciones de coseno son muy similares entre ellas, y en general son la segunda mejor opción según error, y Pearson por número de vecinos es menos preciso que Pearson con un umbral de 0,7, pero en general Pearson es la peor opción.

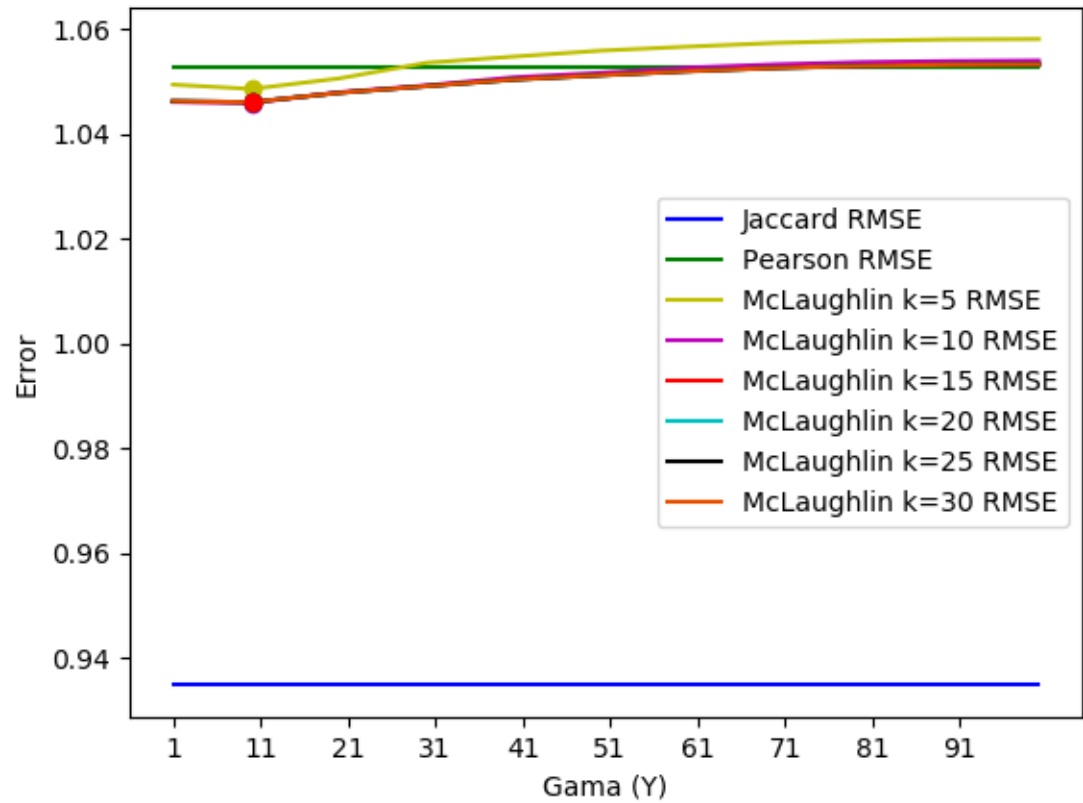
e.

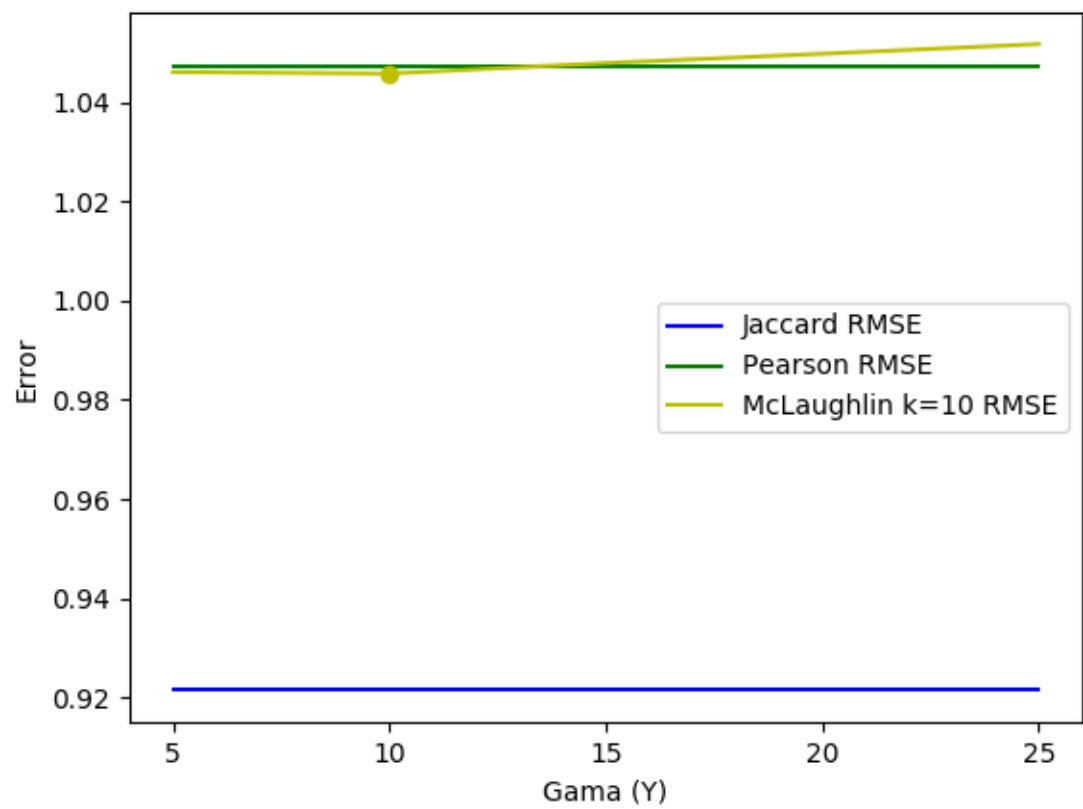
McLaughlin, al igual que los otros tres modelos, es un algoritmo KNNBasic, y por tanto es simétrico. Por eso, se puede cambiar el mismo valor `user_based` en el parámetro `sim_options`. Este parámetro también incluye el `gamma`, llamado `min_support`, que será variado para encontrar el `gamma` óptimo:

```
for gamma in gammas:
    sim_options_mclaughlin = {
        'name': 'mclaughlin',
        'user_based': True,
        'min_support': gamma
    }
```

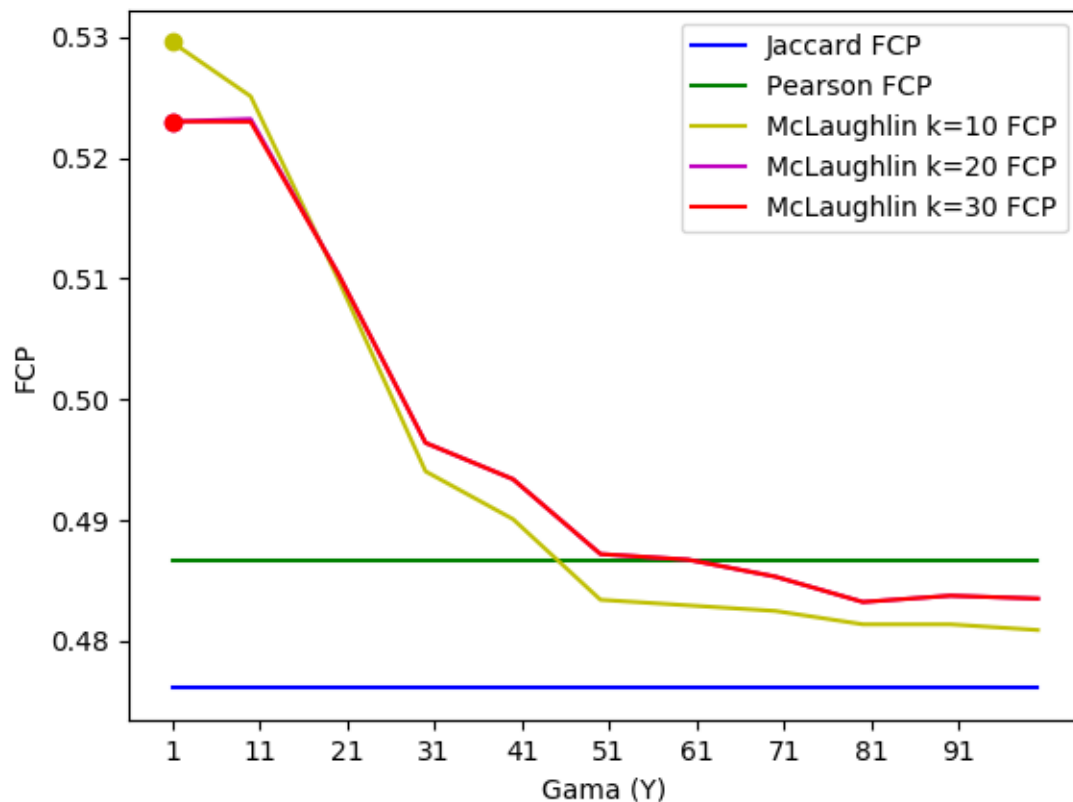
El `gamma` se varió entre 1 y 100, y se graficaron valores de `k` para McLaughlin de 5 a 30 en intervalos de 5. Consistentemente el valor óptimo de `gamma` fue 10, y en casi todos los casos

McLaughlin sí es una mejora de Pearson, sin embargo, están muy cercanos los valores de sus errores. A partir de $k=10$, McLaughlin no presenta un cambio en la precisión.





A continuación se muestra una comparación bajo FCP y no RMSE.



Es interesante observar que el valor óptimo para Gama es 1, y que McLaughlin a menor k es más preciso. La interpretación de que gama sea 1 es que siempre se va a tener en cuenta el número de usuarios en común que clasificaron un artista dado, y ese solo hecho hace a McLaughlin más preciso que Jaccard y Pearson, según FCP.

6. Construcción de la aplicación web

Todos los archivos pertinentes a la página están en la carpeta “webpage”.

El proyecto web incluye una base de datos con tres tablas, una para usuarios, otra para artistas y otra para calificaciones. Se exponen tres APIs de la base de datos de la página web, a través del puerto 8082 y la dirección /api/artist, /api/user y /api/rating.

También hay un API realizado en Python que está en el archivo batch.py en la carpeta webpage/artistfm/src/public/batch. Es un servidor Flask que corre en el puerto 8081. Expone tres direcciones.

/model actualiza la matriz de similitud. Si el servidor recibe una petición /model cuando está calculando una matriz, encola el cálculo de otra matriz. Esto ocurre una vez durante un cálculo de una matriz de similitud, por lo tanto, si termina de calcular la matriz y no ha habido nuevas

peticiones, deja de calcular la matriz. De igual forma, el servidor sigue contestando las demás peticiones aun cuando está ocupado calculando la matriz.

/predict/<user_id> realiza una petición de las 10 predicciones con mejor calificación predicha para el usuario. Las retorna como una lista de ids de artistas.

/ranking hace una petición de los 10 mejores artistas según la fórmula utilizada en el laboratorio 1 (Miller 2009). La fórmula es el “límite inferior del puntaje del intervalo de confianza de Wilson para un parámetro de Bernoulli”, mejor explicada en la página. Está dada por:

$$\frac{\left(\hat{p} + \frac{z^2 \alpha / 2}{2n} \pm z_{\alpha/2} \sqrt{\left[\hat{p}(1 - \hat{p}) + z_{\alpha/2}^2 / 4n \right] / n} \right)}{\left(1 + z_{\alpha/2}^2 / n \right)}$$

En la carpeta webpage/artistfm/src/public/batch se encuentran los archivos batch.py, que contiene el servidor, y el archivo model_builder.py, que contiene una clase ModelBuilder con todas las funciones necesarias para responder las tres solicitudes. Adicionalmente están todas las clases necesarias para construir el modelo.

a.

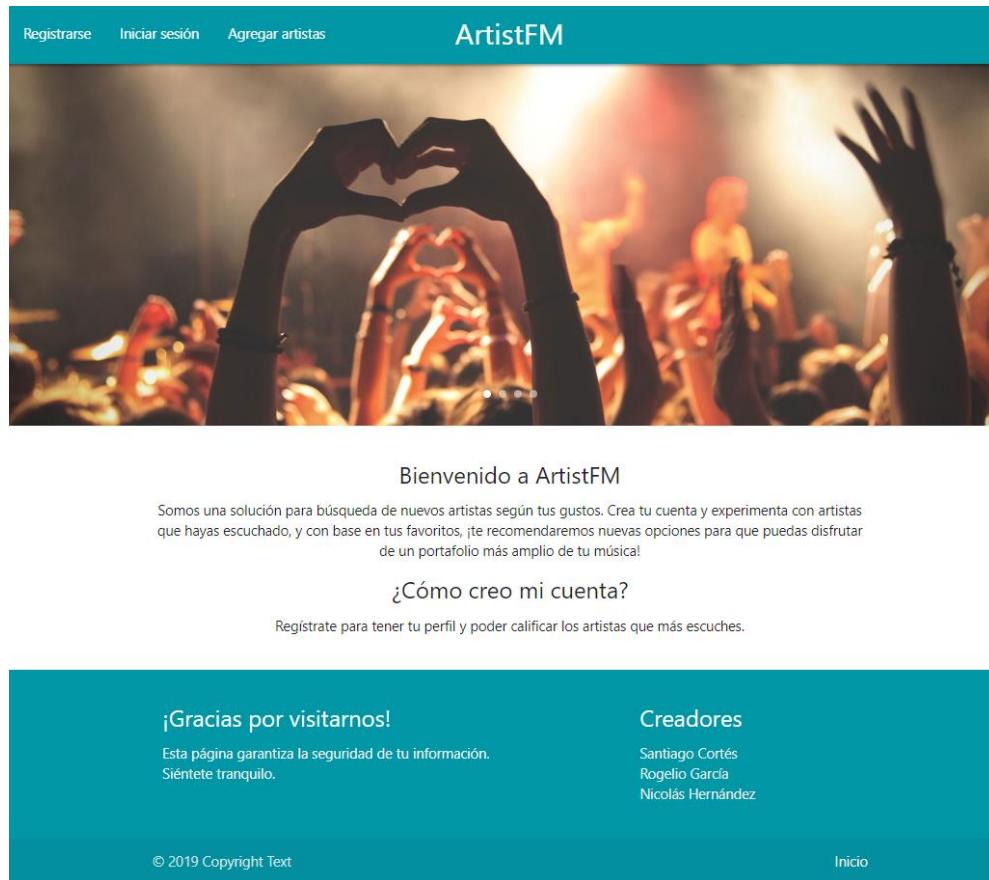
El sistema de recomendación implementado sigue la técnica usuario-usuario, dado su eficiencia en memoria; no es plausible mantener en memoria dos matrices de similitud de más de 80’000 ítems.

Luego de decidir la técnica a utilizar, para definir la métrica de similitud se realizó un cálculo del tiempo que tarda el algoritmo en generar la matriz de similitud. A continuación, se muestra el resultado de un experimento que busca medir el tiempo de ejecución; experimentos siguientes tuvieron un desempeño comparativamente igual.

| Métrica de similitud | Time elapsed (seconds) |
|----------------------|------------------------|
| McLaughlin | 311.41 |
| Jaccard | 111.74 |
| Cosine | 77.64 |
| Pearson | 79.12 |

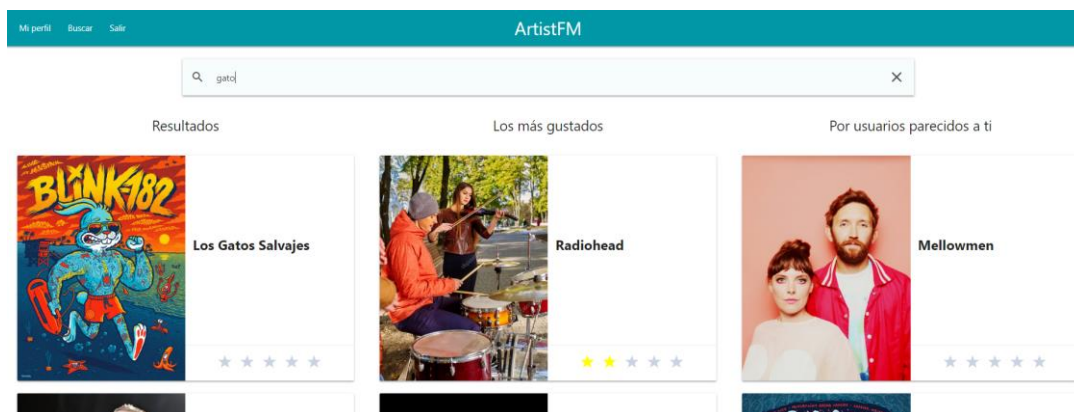
La métrica de similitud utilizada es Jaccard, dado que en experimentos para analizar el error tuvo un desempeño alto, es un cálculo sencillo y su eficiencia es aceptable. En la comparación de diferentes métricas según el criterio FCP con la técnica usuario-usuario, Pearson y Jaccard tenían un desempeño muy similar. Dada su diferencia en desempeño en los experimentos con RMSE, se decidió utilizar Jaccard.

Se desarrolló una página utilizando React, que se encuentra en el directorio webpage.



Luego de ingresar, el botón de buscar se desactiva durante el tiempo que dure haciendo la petición de predicciones para ese usuario, que puede tardar hasta dos segundos. Luego se visualiza una barra de búsqueda para encontrar artistas por nombre, el ranking de los 10 artistas mejor calificados.

También, si el usuario ha realizado calificaciones previamente, se puede calcular su similitud a otros usuarios y, por lo tanto, generar recomendaciones.




Cada ítem visualizado permite realizar una calificación, ver una calificación realizada o editar el puntaje dado.

Para almacenar usuarios, artistas y calificaciones, se tiene una base de datos con una tabla para artistas, otra para usuarios, y una tabla de calificaciones que tiene una referencia a la tabla artistas, una a la tabla usuarios y un valor para el puntaje.

Las recomendaciones se generan con un backend hecho en Python en el archivo batch.py en la carpeta webpage/artistfm/src/public/batch.

El perfil del usuario muestra la información que se proporcionó al registro. En el caso de usuarios que ya estaban en la base de datos, el correo es el id que tenían.

ArtistFM



Mi perfil

| | |
|--------------------|--------------------------------------|
| Nombre | Nicolás |
| Apellidos | Mateo |
| Correo electrónico | nicolas_hero@hotmail.com |
| Fecha de creación | Monday, March 4th, 2019, 10:01:15 PM |

b.

La página permite hacer login o registrar un usuario nuevo mediante el botón ‘iniciar sesión’. Este usuario es agregado a la tabla de usuarios en la base de datos, y no recibirá ninguna recomendación hasta calificar artistas, solamente podría ver la búsqueda por nombre o el ranking general.

ArtistFM

Regístrate en ArtistFM. Recuerda que los datos ingresados son los que necesitarás para iniciar sesión una vez te registre

Ingresa tus datos

Nombres

Apellidos

Imagen de perfil

Correo Electrónico

Escribe tu correo...

Contraseña

Debe tener mínimo 8 caracteres

Confirmación contraseña

Vuelve a escribir tu contraseña

REGISTRARSE

ArtistFM



Resultados

Los más gustados

Por usuarios parecidos a ti



The Beatles


No has calificado ningún artista, aún no sabemos cuáles son tus gustos.


Las preferencias son guardadas a manera de ratings. Por ejemplo, se generan 10 ratings a los 10 ítems del ranking.


Cada vez que se genera un rating, la página web hace un request al backend para pedirle que actualice el modelo. La actualización del modelo se demora hasta 130 segundos. Si hay más ratings durante la actualización, se pone en cola otra actualización.


Resultados

Los más gustados



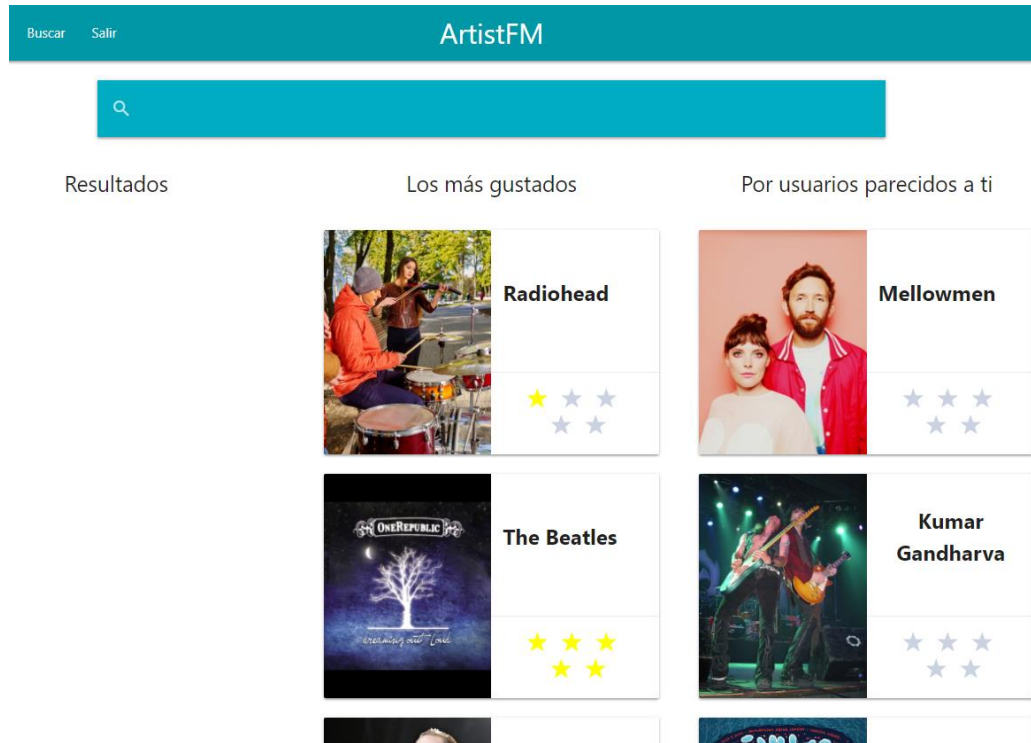
The Beatles




Mgmt




Al seleccionar buscar nuevamente, sí aparecen recomendaciones para el nuevo usuario.




Al hacer click sobre el botón agregar artistas, permite ingresar la información del artista a agregar.

The screenshot shows the 'Agregar artistas' form on the ArtistFM website. The form has a teal header with 'Agregar artistas' on the left and 'ArtistFM' in the center. Below the header is a teal bar with the text 'Crea nuevos artistas para aumentar la red de músicos en nuestra página.' The main section is titled 'Ingresa los datos del músico'. It contains three input fields: 'Nombre' with the value 'rex orange county', 'Musicbrainz' with the value '883cbf1f-dcaa-4d17-b9ed-394e1fc', and 'Imagen' with the value 'https://media.timeout.com/images/105151161/630/472/image.jpg'. At the bottom of the form is a teal button labeled 'CREAR'.

Al crear el artista, ya aparece en la búsqueda y si se hace login, se puede calificar.

🔍 rex orange

Resultados









Rex Orange County

★
★
★
★
★

c.

Los ratings históricos de un usuario se pueden ver al final de la página buscar.

Calificaciones históricas

| | | | | | |
|---|--|---|--|---|--|
|  | <p>The Beatles</p> <div> ★ ★ ★ ★ ★ </div> |  | <p>Radiohead</p> <div> ★ ★ ★ ★ ★ </div> |  | <p>Coldplay</p> <div> ★ ★ ★ ★ </div> |
|  | |  | |  | |

d.

La aplicación está desplegada en su totalidad en la máquina virtual del equipo. La IP de la máquina virtual es 172.24.101.30. La página corre en el puerto 8082 y el backend en el puerto 8081:

<http://172.24.101.30:8082/>

<http://172.24.101.30:8081/>

7. Análisis de resultados

El ranking general tiene sentido, según la importancia de las bandas. The Beatles, Mgmt, Radiohead, Coldplay, Interpol, The Strokes, Muse, Franz Ferdinand, entre otras. Es necesario tener en cuenta que son los hábitos de escucha hasta 5 de mayo de 2009.

Las recomendaciones realizadas tienen sentido; se escogió el modelo que menos error tuvo en experimentos realizados.

Para las métricas de similitud utilizadas, excepto Jaccard, se está asumiendo que un usuario calificó a un artista, sin embargo, esto no es cierto porque la calificación que se dio de un usuario para un artista está basada en las reproducciones que ese usuario haya tenido para una canción de ese artista. Por tanto, también se está asumiendo que, si un usuario escucha muchas veces una canción específica de un artista, le gusta el artista. Por ello, dado que Jaccard no tiene en cuenta los ratings sino únicamente los ítems con los que se ha interactuado alguna vez, es decir, los hábitos de escucha, puede ser una métrica más precisa para este caso en particular dado que no está asumiendo la calificación. Por eso, además, se asume que tras la corrección de McLaughlin se obtiene un error menor, dado que se tienen en cuenta las interacciones en común entre dos usuarios.

Durante los experimentos para decidir el modelo más preciso, un limitante era el tiempo de ejecución de los scripts. Al ser tantos datos y tantas pruebas independientes cambiando un parámetro para poder generar la gráfica resultante, no se pudo experimentar de forma exhaustiva. Por ejemplo, idealmente, todos los experimentos se hubieran repetido con la métrica FCP. Adicionalmente, se evaluarían combinaciones de k número de vecinos y de threshold, dado que hacer thresholding de todas maneras toma un k como *input*, y viceversa. Adicionalmente, se pudo haber hecho la experimentación con MAE también. Dado que solo se tienen valores sintéticos y no precisamente calificaciones, puede que MAE haya sido una métrica más adecuada.

Por otro lado, con McLaughlin se propone una corrección para Pearson. En comparaciones según RMSE, igualmente tiene un desempeño muy inferior a Jaccard. Sin embargo, en el paper se propone una nueva forma de evaluar dos algoritmos. Al escoger una métrica similar en *Surprise*, McLaughlin tiene un desempeño superior al de los demás modelos. Es, entonces, una cuestión de decidir qué métrica utilizar. En el paper de McLaughlin se menciona que RMSE y MAE dejan muchos errores en las predicciones “escondidos”, dado que algunos de los resultados son “oscuros”. Con esto se refiere a que pueden ser ítems difíciles o imposibles de conseguir, o, por ejemplo, en un idioma que no es relevante para el usuario. En este caso, en last.fm, el acceso a un artista específico o la reproducción de una canción no es un problema. Por estas razones, se escogió tener en cuenta los resultados de los experimentos con RMSE.

Para incorporar los nuevos ítems y los nuevos usuarios, se agregar a la tabla correspondiente en la base de datos al agregarlos con el formulario en la página. Para un nuevo usuario no se pueden generar recomendaciones dado que no ha realizado calificaciones. Una vez realiza calificaciones, ya aparecen recomendaciones para el usuario dados sus vecinos. Un nuevo ítem, de manera similar, no va a aparecer en recomendaciones si no se le ha asignado una calificación. Al asignarle una calificación, ya aparece en la tabla de calificaciones, que es la que se utiliza para actualizar el sistema de recomendación.

8. Bibliografía

Miller, E. (2009, February 6). How Not To Sort By Average Rating. Retrieved March 3, 2019, from <http://www.evanmiller.org/how-not-to-sort-by-average-rating.html>

Mclaughlin, M. R., & Herlocker, J. L. (2004). A collaborative filtering algorithm and evaluation metric that accurately model the user experience. Proceedings of the 27th Annual International Conference on Research and Development in Information Retrieval - SIGIR 04. doi:10.1145/1008992.1009050