Obligatorio 2: Sistemas de Recomendación

Estudiantes: Santiago Esquerré, Candela Giménez
Docente: Dr. Mario González

Introducción

Los sistemas de recomendación se han convertido en herramientas fundamentales en diversas aplicaciones comerciales, como plataformas de streaming (Netflix, Spotify, YouTube) y comercio electrónico (Amazon, Mercado Libre). Su objetivo principal es predecir las preferencias o puntajes que un usuario asignaría a un ítem específico, como películas, canciones o productos, basándose en información histórica.

Existen dos enfoques principales para construir sistemas de recomendación:

- **Filtrado basado en contenido:** Este modelo utiliza características predefinidas de los ítems, como el género, director o elenco de una película, para recomendar elementos similares que coincidan con las preferencias individuales de un usuario.
- **Filtrado colaborativo:** Este enfoque se basa en el comportamiento colectivo de los usuarios, asumiendo que aquellos con preferencias similares en el pasado compartirán intereses futuros. A diferencia del filtrado basado en contenido, este modelo no requiere características predefinidas de los ítems.

En este trabajo, se exploran estos dos métodos aplicados al conjunto de datos 'MovieLens small', que contiene más de 100,000 puntuaciones realizadas por 610 usuarios sobre 9,742 películas. El problema se aborda mediante el uso de mínimos cuadrados y regularización para minimizar el error cuadrático medio entre los puntajes estimados y los reales, siguiendo las ideas del sistema ganador del Netflix Prize. También se evalúa el impacto de parámetros clave, como la regularización y la dimensionalidad de las representaciones, en la calidad de las recomendaciones.

Modelado del Problema

La esencia de cualquier sistema de recomendación se centra en estimar el puntaje r_{ui} que un usuario u, le asigna a un ítem i (en este caso una película), basándose en la información que los usuarios generan a medida que interactúan con el sistema. Este problema puede ser modelado como una aproximación por mínimos cuadrados, modelando los datos disponibles (tanto de los usuarios como de los ítem) en lo que se llama un espacio latente.[1]

Recordemos que un problema de aproximación por mínimos cuadrados consiste en que dado un conjunto de m datos $\{(t_i, y_i)\}_{i\in\{1,\dots,m\}} \subset \mathbb{R}^2$, buscamos un 'ajuste' para estos datos a partir de una combinación lineal de funciones linealmente independientes $\phi_1, \dots, \phi_n$. Es decir, buscamos:

$$f(t) = \sum_{j=1}^{n} x_j \phi_j(t) \quad \text{tal que} \quad y_i \approx f(t_i)$$

[1] Un espacio multidimensional abstracto que contiene valores de características que no podemos interpretar directamente, pero que codifica una representación interna significativa de los eventos observados externamente.

Lo que se traduce en hallar un vector $\mathbf{x} \in \mathbb{R}^n$ que cumpla:

$$\mathbf{x} = \arg\min_{\mathbf{v}\in\mathbb{R}^n} \left[\sum_{k=1}^{m} (\langle \mathbf{v}, \Phi(t_k)\rangle - y_k)^2\right] \quad \text{donde} \quad \Phi(t) = (\phi_1(t), \ldots, \phi_n(t))$$

Filtrado en base a contenidos

En el caso del filtrado basado en contenidos, recordamos que nuestro dataset nos proporciona 100.836 puntuaciones realizadas por 610 usuarios sobre 9742 películas, además, para cada una de estas, se tiene una clasificación en al menos uno de los géneros: *Action, Adventure, Animation, Children, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western.*

De esta forma nuestro conjunto de datos es $\{(m_i, r_i)\}_{i\in\{1,\ldots,100{,}836\}}$, donde r_i es el puntaje que un cierto usuario le asignó a la película m_i.

Predefinimos las funciones ϕ_k, de la siguiente forma:

- Enumeramos los géneros de películas del 1 al 18.
- Enumeramos las películas del dataset del 1 al 9742.
- Definimos $\phi : \{1 \ldots 9742\} \to \{0, 1\}$ tal que $\phi_k(t) = 1 \Leftrightarrow$ la película t es del género k, y 0 en otro caso.

Entonces, para hacer la recomendación basada en contenidos, tomamos un cierto usuario u y definimos el conjunto $I_u = \{k \in \mathbb{N} : \text{el par } (m_k, r_k) \text{ corresponde al usuario } u\}$, luego, necesitamos encontrar el vector $\mathbf{p}_u^*$ de puntajes estimados para el usuario u, el cual debe cumplir que:

$$\mathbf{p}_u^* = \arg\min_{\mathbf{p}_u\in\mathbb{R}^{18}} \left[\sum_{k\in I_u} (\langle \mathbf{p}_u, \Phi(m_k)\rangle - r_k)^2\right] \tag{1}$$

Una vez se obtiene $\mathbf{p}_u^*$, podemos generar una lista de preferencia de películas para el usuario u mediante el siguiente procedimiento:

- Consideramos la matriz $Q \in \mathcal{M}_{9742\times 18}(\mathbb{R})$ cuya fila k-ésima es el vector $\Phi(m_k)$.
- Hallamos $\mathbf{p}_u^*$ que cumpla con (1).
- Hacemos $\mathbf{r}_u^* = Q(\mathbf{p}_u^*)^T$.
- Consideramos el vector $\mathbf{l}_u$ resultado de aplicar un ordenamiento descendente en los valores de las componentes de $\mathbf{r}_u^*$ y posteriormente tomar los índices originales de las componentes ordenadas, es decir, si $\mathbf{r}_u^* = (7, 9, 22, 3, 47)$, entonces $\mathbf{l}_u = (5, 3, 2, 1, 4)$.

De esta forma, $\mathbf{l}_u$ tiene los n° de película ordenados por preferencia descendente para el usuario u.

Filtrado colaborativo

Con el fin de mejorar el rendimiento del método anterior, queremos dejar que el algoritmo decida como definir las funciones ϕ_k con el fin de que el algoritmo identifique patrones en la relación película-usuario y genere una función Φ que extraiga una cantidad $f \in \mathbb{N}$ de características (variables latentes) de la misma.

Entonces, nuestro problema ahora consiste en estimar $\mathbf{p}_u$ para cada usuario, y $\Phi(m_k)$ (que a partir de ahora llamaremos $\mathbf{q}_k$) para cada película.

Esto se traduce en encontrar $P^* \in \mathcal{M}_{610\times 18}(\mathbb{R})$ cuya fila n-ésima es el vector $\mathbf{p}_k^*$ y $Q^* \in \mathcal{M}_{9742\times 18}(\mathbb{R})$ cuya fila k-ésima es el vector $\mathbf{q}_k^*$ que cumplan con:

$$(P^*, Q^*) = \arg\min_{P,Q} \left[\sum_{u=1}^{610} \sum_{k \in I_u} (\langle \mathbf{p}_u, \mathbf{q}_k \rangle - r_{uk})^2 \right]$$

Para esto vamos a aplicar un método iterativo de *minimización alternada* con los siguientes pasos:

- Elegimos un valor de $f \in \mathbb{N}$
- Inicializar matrices $P^{(0)} \in \mathcal{M}_{610\times f}(\mathbb{R})$ y $Q^{(0)} \in \mathcal{M}_{9742\times f}(\mathbb{R})$ (en la implementación se ve que las hacemos aleatorias en una distribución normal con media 0 y varianza 1).
- Para $k \geq 0$:
  - $p_u^{(k+1)} = \arg\min_{X\in\mathbb{R}^f} \|Q_u^{(k)} X - r_u\|^2 \quad \forall u \in \{1, \ldots, 610\}$
  - $q_i^{(k+1)} = \arg\min_{Y\in\mathbb{R}^f} \|P_i^{(k+1)} Y - r_i\|^2 \quad \forall i \in \{1, \ldots, 9742\}$

Una vez terminadas las iteraciones, dado un usuario u, se puede obtener su lista de preferencia, computando el vector $\mathbf{r}_u^* = Q(\mathbf{p}_u^*)^T$, y considerando $\mathbf{l}_u$ como en el filtrado en base a contenidos.

Si bien este modelo debería funcionar a nivel teórico, la realidad es que este esquema iterativo tiene una probabilidad realmente grande de generar un *sobreajuste*[2] en los datos de entrada y por lo tanto, si bien el error en los datos de entrada es realmente bajo, el error en los datos de prueba se dispara (como veremos en la sección de resultados).

Para mitigar el *sobreajuste*, utilizamos la regularización de Tychonoff. Este método introduce un término adicional en la función que buscamos minimizar, el cual penaliza soluciones con normas grandes.

Entonces, las matrices P^* y Q^* que buscamos en este caso, son:

$$(P^*, Q^*) = \arg\min_{P,Q} \left[\sum_{u=1}^{610} \sum_{k \in I_u} (\langle \mathbf{p}_u, \mathbf{q}_k \rangle - r_{uk})^2 + \lambda(\|\mathbf{p}_u\|^2 + \|\mathbf{q}_k\|^2) \right]$$

Se puede modificar el algoritmo de la parte anterior para que contemple este nuevo término:

- $p_u^{(k+1)} = \arg\min_{X\in\mathbb{R}^f} \|\tilde{Q}_u^{(k)} X - \tilde{r}_u\|^2$
- $q_i^{(k+1)} = \arg\min_{Y\in\mathbb{R}^f} \|\tilde{P}_i^{(k+1)} Y - \tilde{r}_i\|^2$

donde aquí:

[2]El modelo aprende no solo los patrones generales y útiles que son representativos del problema, sino también las particularidades, ruidos o irregularidades presentes en los datos de entrenamiento que no son generalizables a nuevos datos.

- $\tilde{Q}_u^{(k)} = \begin{pmatrix} Q_u^{(k)} \\ \sqrt{\lambda} I_f \end{pmatrix} \in \mathbb{R}^{(9742+f)\times f}, \quad \tilde{r}_u = \begin{pmatrix} r_u \\ \mathbf{0}_f \end{pmatrix} \in \mathbb{R}^{9742+f}$
- $\tilde{P}_i^{(k+1)} = \begin{pmatrix} P_i^{(k+1)} \\ \sqrt{\lambda} I_f \end{pmatrix} \in \mathbb{R}^{(610+f)\times f}, \quad \tilde{r}_i = \begin{pmatrix} r_i \\ \mathbf{0}_f \end{pmatrix} \in \mathbb{R}^{610+f}$

(Aquí I_f es la matriz identidad $f \times f$ y $\mathbf{0}_f$ es el vector nulo de $\mathbb{R}^f$.)

Llegado este punto se puede obtener $\mathbf{l}_u$ como antes, pero en este caso obtenemos que esta solución mitiga considerablemente el *sobreajuste*.

Implementación

Una vez modelados los problemas pasamos a mostrar una implementación en `Octave` de los solver para cada uno de ellos.

Filtrado basado en contenidos

Recordando el procedimiento planteado en la sección de modelado tenemos:

1. Computamos la matriz Q usando la función `createMovieGenderMatrix`:

```
    % -*- functionInfo -*-
    %  PRE:  Existe un archivo CSV en path con la matriz Q, cuyas fila i-ésima es un vector qi
          de 0s y 1s donde en la posición k ponemos 1 si dicha película es del género k, y 0 en
          caso contrario.
    %  POS: Se ha construido la matriz Q con las filas del csv, en formato sparse.
    function Q = createMovieGenderMatrix (path)

    if (nargin != 1)
    path = '../data/clasificacion_peliculas.csv';
    end

    % Leer el archivo
    data = csvread(path, 1, 1); % Saltar la cabecera

    % Extraer las dimensiones
    num_peliculas = size(data, 1); % Número de filas (películas)
    num_generos = size(data, 2); % Número de géneros (sin la columna de ID)

    % Extraer las posiciones de los valores no nulos (1s)
    [row_indices, col_indices] = find(data == 1);

    % Crear la matriz sparse
    Q = sparse(row_indices, col_indices, 1, num_peliculas, num_generos);

    end
```

2. Computamos el conjunto de datos de puntajes asignados por el usuario u con la función `createUserRatingsMatrix`:

```
% -*- functionInfo -*-
%  PRE:  Existe un archivo CSV en path con las ternas (usuario_Id, pelicula_Id, puntaje) y un
     usario con la id usuario_id.
%  POST: Se construye la matriz de duplas (id película y puntaje) para un usuario.
function r_u = createUserRatingsMatrix(usuario_id, path)

    if (nargin == 1)
        path = '../data/puntajes_ajuste.csv';
    end

    % Leer los datos del archivo
    data = csvread(path, 1, 0);

    % Filtrar las filas correspondientes al usuario_id
    filas_usuario = data((data(:, 1) == usuario_id), :);

    % Crear la matriz de duplas (id película y puntaje)
```

```
    r_u = filas_usuario(:, 2:3); % Segunda columna: id película, tercera columna: puntaje
end
```

3. Encontramos el vector de gustos $\mathbf{p}_u^*$ resolviendo el sistema $Q_u(\mathbf{p}_u^*)^T = \mathbf{r}_u$ en el sentido de mínimos cuadrados, donde Q_u es la matriz Q considerando solo las filas correspondientes a las películas puntuadas por el usuario u, y $\mathbf{r}_u$ es el vector de puntajes asignados por el usuario u. Para esto, usamos la función `createTasteVector`:

```
% -*- functionInfo -*-
% PRE: Existe un usuario con el id usuario_id.
% POST: Se ha construido el vector p_u con los puntajes estimados.
function p_u = createTasteVector (usuario_id, Q, r_u)
    % Cargar las matrices Q y R
    if (nargin == 1)
        Q = createMovieGenderMatrix();
        r_u = createUserRatingsMatrix(usuario_id);
    else if (nargin == 2)
        r_u = createUserRatingsMatrix(usuario_id);
    end

    p_u = Q(r_u(:, 1), :) \ r_u(:, 2);

end
```

4. Computamos los vectores $\mathbf{r}_u^*$ $\mathbf{l}_u$ como explicamos en la parte de modelado usando la función `generateRecommendationList`:

```
% -*- functionInfo -*-
% PRE: Existe un usuario con el id usuario_id.
% POST: Se construye la lista de recomendación de películas L para el usuario con el id
    usuario_id usando filtrado en base a contenidos, es decir, donde se ordenan los id de las
    películas en orden de preferencia decreciente, además se retorna LRatings, que es el
    vector de puntajes de las películas, que en la posición i-ésima tiene el puntaje de la
    película i.
function [L, LRatings] = generateRecommendationList (usuario_id)
    Qu = createMovieGenderMatrix();
    r_u = createUserRatingsMatrix(usuario_id);
    p_u = createTasteVector(usuario_id, Qu, r_u);
    LRatings = Qu * p_u;

    [~, L] = sort(abs(LRatings), 'descend');
end
```

Filtrado colaborativo

Minimización alternada sin regularización

Para la minimización alternada sin regularización, se implementó la función `alternatingMinimization` que resuelve el problema de minimización alternada de la siguiente forma:

```
% -*- functionInfo -*-
% PRE: Existen los archivos de datos puntajes_ajuste.csv y puntajes_test.csv. Se tiene un valor f de
    la dimensión de las variables latentes, un valor maxiter de la cantidad de iteraciones y un
    valor saveError que indica si se quiere guardar el error cuadrático medio en cada iteración.
% POST: Se intenta minimizar el error cuadrático medio de los puntajes calculados a través del
    método de minimización alternada, efectivo para datos de ajuste, no tanto para test.
function [P, Q, errAjuste, errTest] = alternatingMinimization(path, maxiter, f, saveError)
    % Verificación de argumentos
    if (nargin < 4)
        saveError = 0;
    end

    if (nargin < 3)
        f = 15;
    end

    if (nargin < 2)
        maxiter = 50;
```

```
    end

    if (nargin < 1)
        path = "../data/puntajes_ajuste.csv";
    end

    % Carga el conjunto de datos
    addpath("../data");
    data = csvread(path, 1, 0);
    data_test = csvread("../data/puntajes_test.csv", 1, 0);

    % Definir número de usuarios y películas
    num_pelis = max(data(:, 2));
    num_usuarios = max(data(:, 1));

    % Inicialización aleatoria de las matrices P y Q
    P = randn(num_usuarios, f);
    Q = randn(num_pelis, f);

    % Inicialización de vectores de error
    errAjuste = zeros(maxiter, 1);
    errTest = zeros(maxiter, 1);

    % Preprocesar índices de usuarios y películas
    indices_usuarios = cell(num_usuarios, 1);
    indices_peliculas = cell(num_pelis, 1);

    for u = 1:num_usuarios
        indices_usuarios{u} = find(data(:, 1) == u);
    end

    for i = 1:num_pelis
        indices_peliculas{i} = find(data(:, 2) == i);
    end

    % Iteraciones de minimización alternada
    for k = 1:maxiter
        % Actualización de P
        for u = 1:num_usuarios
            filas_usuario = data(indices_usuarios{u}, :);
            r_u = filas_usuario(:, 2:3); % id película y puntaje
            Q_u = Q(r_u(:, 1), :); % Subconjunto de Q
            P(u, :) = Q_u \ r_u(:, 2); % Resolver sistema
        end

        % Actualización de Q
        for i = 1:num_pelis
            filas_pelicula = data(indices_peliculas{i}, :);
            r_i = filas_pelicula(:, [1, 3]); % id usuario y puntaje
            P_i = P(r_i(:, 1), :); % Subconjunto de P
            Q(i, :) = P_i \ r_i(:, 2); % Resolver sistema
        end

        if (saveError)
            errAjuste(k) = error_cuadratico_medio(P, Q, data);
            errTest(k) = error_cuadratico_medio(P, Q, data_test);
        end

    end

end
```

Donde la función `error_cuadratico_medio` es la dada en los datos del obligatorio.

Minimización alternada con regularización de Tychonoff

Se hizo una implementación similar a la anterior, pero considerando la regularización de Tychonoff. Se obtiene la función `tychonoffRegularization` con el siguiente código:

```
% -*- functionInfo -*-
% PRE: Existen los archivos de datos puntajes_ajuste.csv y puntajes_test.csv. Se tiene un valor f de
     la dimensión de las variables latentes, un valor maxiter de la cantidad de iteraciones, un
    valor lambda de la regularización y un valor saveError que indica si se quiere guardar el error
    cuadrático medio en cada iteración.
```

```
% POST: Se minimiza el error cuadrático medio de los puntajes calculados a través del método de
    minimización alternada con regularización de Tychonoff.
function [P, Q, errAjuste, errTest] = tychonoffRegularization(path, maxiter, f, lambda, saveError)
    % Verificación de argumentos
    if (nargin < 5)
        saveError = 0;
    end

    if (nargin < 4)
        lambda = 1.2;
    end

    if (nargin < 3)
        f = 15;
    end

    if (nargin < 2)
        maxiter = 50;
    end

    if (nargin < 1)
        path = "../data/puntajes_ajuste.csv";
    end

    % Carga el conjunto de datos
    addpath("../data");
    data = csvread(path, 1, 0);
    data_test = csvread("../data/puntajes_test.csv", 1, 0);

    % Definir número de usuarios y películas
    num_pelis = max(data(:, 2));
    num_usuarios = max(data(:, 1));

    % Inicialización aleatoria de las matrices P y Q
    P = 0.2 * randn(num_usuarios, f);
    Q = 0.2 * randn(num_pelis, f);
    sqrt_lambda_I = sqrt(lambda) * eye(f);
    zeros_f = zeros(f, 1);

    % Inicialización de vectores de error
    errAjuste = zeros(maxiter, 1);
    errTest = zeros(maxiter, 1);

    % Preprocesar índices de usuarios y películas
    indices_usuarios = cell(num_usuarios, 1);
    indices_peliculas = cell(num_pelis, 1);

    for u = 1:num_usuarios
        indices_usuarios{u} = find(data(:, 1) == u);
    end

    for i = 1:num_pelis
        indices_peliculas{i} = find(data(:, 2) == i);
    end

    % Iteraciones de minimización alternada
    for k = 1:maxiter
        % Actualización de P
        for u = 1:num_usuarios
            filas_usuario = data(indices_usuarios{u}, :);
            r_u = filas_usuario(:, 2:3); % id película y puntaje
            Q_u = Q(r_u(:, 1), :); % Subconjunto de Q
            P(u, :) = [Q_u; sqrt_lambda_I] \ [r_u(:, 2); zeros_f]; % Resolver sistema
        end

        % Actualización de Q
        for i = 1:num_pelis
            filas_pelicula = data(indices_peliculas{i}, :);
            r_i = filas_pelicula(:, [1, 3]); % id usuario y puntaje
            P_i = P(r_i(:, 1), :); % Subconjunto de P
            Q(i, :) = [P_i; sqrt_lambda_I] \ [r_i(:, 2); zeros_f]; % Resolver sistema
        end

        if (saveError)
            errAjuste(k) = error_cuadratico_medio(P, Q, data);
            errTest(k) = error_cuadratico_medio(P, Q, data_test);
```

```
        end

    end

end
```

Resultados

Métricas de Evaluación

Para evaluar el desempeño de los sistemas implementados en este trabajo, se utilizó el error cuadrático medio (ECM) como métrica principal. El ECM mide la discrepancia entre los puntajes estimados con $\langle \mathbf{p}_u, \mathbf{q}_i \rangle$ y los puntajes reales r_{ui}, y se define como:

$$\text{ECM} = \frac{1}{N} \sum_{u=1}^{610} \sum_{i \in I_u} (\langle \mathbf{p}_u, \mathbf{q}_i \rangle - r_{ui})^2 \tag{2}$$

donde:

- N: número total de puntuaciones consideradas (tanto en los datos de ajuste como de prueba).
- M: número total de usuarios.
- I_u: conjunto de ítems que el usuario u puntuó.
- $\langle p_u, q_i \rangle$: puntaje estimado por el modelo.
- r_{ui}: puntaje real otorgado por el usuario u al ítem i.

El ECM es adecuado como métrica de calidad para este problema porque:

- **Penaliza grandes errores:** Al elevar al cuadrado las diferencias, se asigna mayor peso a las predicciones con mayores desviaciones, lo que ayuda a identificar casos extremos donde el modelo no está funcionando bien.
- **Promueve un ajuste generalizado:** Minimizar el ECM garantiza que las predicciones estén, en promedio, lo más cerca posible de los valores reales, logrando un equilibrio entre precisión y generalización.
- **Comparabilidad:** El ECM permite comparar directamente la calidad de las predicciones en diferentes configuraciones del modelo, como el impacto del filtrado basado en contenido frente al colaborativo o el efecto de la regularización.

Por estas razones, el ECM es una métrica estándar en el desarrollo y evaluación de sistemas de recomendación. Además, proporciona una base sólida para analizar el desempeño de los métodos implementados, tanto en los datos de ajuste como en los de prueba, ayudando a identificar fenómenos como el sobreajuste y a ajustar los parámetros del modelo. Se utilizó la función `error_cuadratico_medio` dada en los datos de este obligatorio, para calcular el ECM en cada caso.

Resultados de filtrado basado en contenidos

Para evaluar el rendimiento del filtrado basado en contenidos, se implementó la función `compareResults` que toma un id de usuario y calcula el ECM en los datos de ajuste y de prueba para el usuario dado:

```
% -*- functionInfo -*-
% PRE: Existe un usuario con el id user_id.
% POST: Se compara la lista de recomendación de películas generada por el sistema de recomendación
    en base a contenidos con los puntajes disponibles en el dataset de test y de ajuste;
function [errAjuste, errTest] = compareResults (user_id, test_data_path, ajuste_data_path)

    if (nargin < 3)
        ajuste_data_path = "../data/puntajes_ajuste.csv";
    end

    if (nargin < 2)
        test_data_path = "../data/puntajes_test.csv";
    end

    [L, LRatings] = generateRecommendationList(user_id);

    data_ajuste = csvread(ajuste_data_path, 1, 0);
    test_data = csvread(test_data_path, 1, 0);

    available_ratings_ajuste = data_ajuste(data_ajuste(:, 1) == user_id, [2, 3]);
    available_ratings_test = test_data(test_data(:, 1) == user_id, [2, 3]);

    errAjuste = (norm(LRatings(available_ratings_ajuste(:, 1)) - available_ratings_ajuste(:, 2), 2)
        ^2) / size(available_ratings_ajuste, 1);
    errTest = (norm(LRatings(available_ratings_test(:, 1)) - available_ratings_test(:, 2), 2)^2) /
        size(available_ratings_test, 1);

endfunction
```

Al utilizarla para el usuario 4, obtenemos los siguientes resultados:

```
octave:1> [errAjuste, errTest] = compareResults(4);
octave:2> errAjuste
errAjuste = 3.2757
octave:3> errTest
errTest = 2.2139
```

Los datos que se obtienen pueden hacer un poco de ruido a primera vista, pensando que el error en los datos de ajuste es mayor que en los de prueba, pero esto se debe a que la cantidad de datos de prueba es mucho menor que la de los de ajuste, esto ocasiona que al haber más datos, la probabilidad de acumular un error mayor crece.

Resultados de filtrado colaborativo

Para el caso de filtrado basado en contenidos, vimos como en las implementaciones de `alternatingMinimization` y `tychonoffRegularization` se guardaba el error cuadrático medio en cada iteración, esto nos permite graficar la evolución del error en función de las iteraciones, lo que nos permite ver si el modelo converge y si lo hace, en cuántas iteraciones lo hace.

A modo de no saturar con gráficas el informe, adjuntamos en [1] en la ruta `Código/scripts_parte2/figuras` diferentes gráficas que muestran la evolución del error cuadrático medio en función de las iteraciones para diferentes valores de f y λ.

Lo primero que observamos en las gráficas para el caso de `alternatingMinimization` (sin regularización) es que, en los datos de ajuste, el error cuadrático medio disminuye a medida que aumentamos el número de iteraciones, lo que indica que el modelo converge y es capaz de ajustarse a dichos datos. Sin embargo, este comportamiento no se mantiene para los datos de prueba, pues vemos que el error se dispara con el transcurso de las iteraciones, esto claramente es efecto del fenómeno de *sobreajuste* que habíamos comentado anteriormente, el modelo se

ajusta demasiado bien a los datos de ajuste, tomando en cuenta también ciertos patrones que no son generalizables para otros datos.

Para el caso de `tychonoffRegularization`, lo que observamos en dichas figuras es que a medida que aumentamos la dimensión de las variables latentes f, el error cuadrático medio disminuye, lo que indica que el modelo es capaz de ajustarse mejor a los datos. Sin embargo, este efecto no es lineal, y a partir de cierto valor de f el error comienza a estabilizarse, lo que sugiere que agregar más variables latentes no necesariamente mejora la calidad de las recomendaciones.

En cuanto a los valores de λ, observamos que la regularización de Tychonoff permite controlar el sobreajuste y mejorar la generalización del modelo. A medida que aumentamos λ, el error cuadrático medio en los datos de prueba disminuye, lo que indica que el modelo es capaz de generalizar mejor a nuevos datos y evitar el sobreajuste.

Conclusión

Los resultados obtenidos en este trabajo evidencian la efectividad de los métodos de recomendación basados en mínimos cuadrados y regularización. El filtrado colaborativo, a través de la minimización alternada, mostró ser capaz de capturar relaciones complejas entre usuarios e ítems, mientras que el filtrado basado en contenidos demostró ser útil en escenarios donde la información explícita de los ítems está disponible. Sin embargo, cada enfoque presenta limitaciones, como el riesgo de sobreajuste en modelos no regularizados y la dependencia de información contextual en el caso del filtrado basado en contenidos.

El uso de la regularización de Tychonoff permitió mitigar el sobreajuste, logrando un balance entre precisión y generalización, especialmente en los datos de prueba. Los experimentos mostraron que aumentar la dimensionalidad del espacio latente mejora el ajuste del modelo hasta un punto de saturación, lo que resalta la importancia de ajustar cuidadosamente los parámetros f y λ para maximizar el rendimiento sin comprometer la eficiencia computacional.

En conclusión, este estudio proporciona un análisis detallado de dos enfoques fundamentales para sistemas de recomendación, destacando la importancia de métricas como el error cuadrático medio para evaluar su desempeño. Las implementaciones realizadas no solo permiten explorar el impacto de diversos parámetros, sino que también sientan una base sólida para futuras mejoras y adaptaciones en contextos reales, como aplicaciones comerciales o plataformas digitales.

Referencias

[1] Repositorio de GitHub con código fuente y datos adjuntos, `https://github.com/santi-esquerre/MetNum--Obligatorio2--SistemasDeRecomendacion`