LARGE-SCALE BANDIT RECOMMENDER SYSTEM [1]

IIC3633 - Sistemas Recomendadores

Saúl Langarica Chavira

19 de Octubre, 2017

Pontificia Universidad Católica de Chile

INTRODUCCIÓN

INTRODUCCIÓN

Debido a que la actualización de los modelos de usuarios puede ser bastante costosa computacionalmente, la mayoría de los sistemas recomendadores lo hacen con baja frecuencia (Una vez al día, una vez a la semana, etc).

- Esto puede causar inconvenientes en sistemas que deben actualizarse rápidamente por gustos cambiantes del usuario o por cambio de contexto.

- Por lo tanto, en ocasiones se hace necesario contar con un sistema recomendador que sea capaz de aplicar acciones y/o correcciones inmediatas una vez que recibe el feedback del usuario.

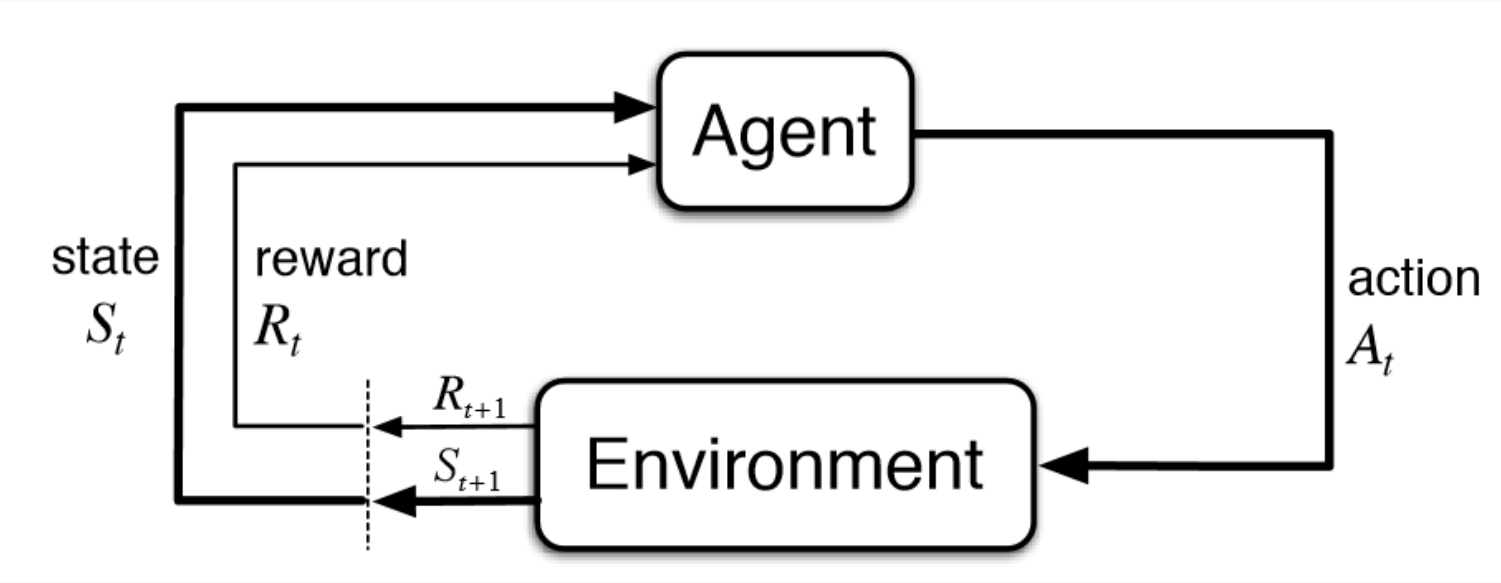
- Una solución para esto es utilizar algoritmos de **Reinforcement learning**.

- En el presente paper se muestra la implementación de un filtro colaborativo basado en matrices de factorización y algoritmos de Reinforcement learning.

REINFORCEMENT LEARNING

Los algoritmos de Reinforcement learning están inspirados en la forma en que aprenden los animales. Se busca tomar acciones en cada paso de tiempo que maximicen una recompensa a largo plazo (o equivalentemente, que minimicen el castigo).

El esquema típico de estos algoritmos se presenta a continuación:



Estos algoritmos son de carácter secuencial pues en cada paso de tiempo:

1. El agente, en base a un modelo interno, escoge una acción
2. El ambiente (que puede ser desconocido) responde con una recompensa y una transición de estado
3. El agente modifica su modelo interno de acuerdo a la recompensa recibida y escoge otra acción

APLICACIÓN A SISTEMAS RECOMENDADORES

Dado que el problema de recomendar un item a un usuario se puede ver como un problema secuencial en que en cada paso de tiempo:

1. El sistema, en base a un modelo interno del usuario, escoge un item para recomendarle
2. El usuario entrega un feedback al sistema respecto a la recomendación (recompensa)
3. El sistema actualiza su modelo interno del usuario en base a este feedback

Se puede concluir entonces que el problema de realizar recomendaciones en tiempo real se puede abordar con los algoritmos de Reinforcement learning.

BLOQUES DEL ALGORITMO

MATRICES DE FACTORIZACIÓN

Dada una matriz **R*** con ratings con items en sus columnas y usuarios en sus filas, en la que la gran mayoría de las entradas son desconocidas y son estas entradas desconocidas las que se buscan predecir. Se puede encontrar una factorización matricial de la forma:

$$R^* = UV^T \tag{1}$$

En donde U es de N x k con características del usuario y V es de M x k con características de los itemes. Un estimador de R* se puede obtener al resolver el problema de optimización dado por:

$$\widehat{\mathbf{R}}^* \stackrel{def}{=} \widehat{\mathbf{U}}\widehat{\mathbf{V}}^T, \text{ s.t. } (\widehat{\mathbf{U}}, \widehat{\mathbf{V}}) = \underset{\mathbf{U},\mathbf{V}}{\operatorname{argmin}} \sum_{\forall (i,j) \in \mathcal{S}} \left(r_{i,j} - \mathbf{U}_i \mathbf{V}_j^T\right)^2 + \lambda \cdot \Omega(\mathbf{U}, \mathbf{V})$$

MATRICES DE FACTORIZACIÓN

Donde $r_{i,j}$ es el rating del usuario i al item j, λ es el factor de regularización y $\Omega(U, V)$ es el término de regularización que en este caso está dado por:

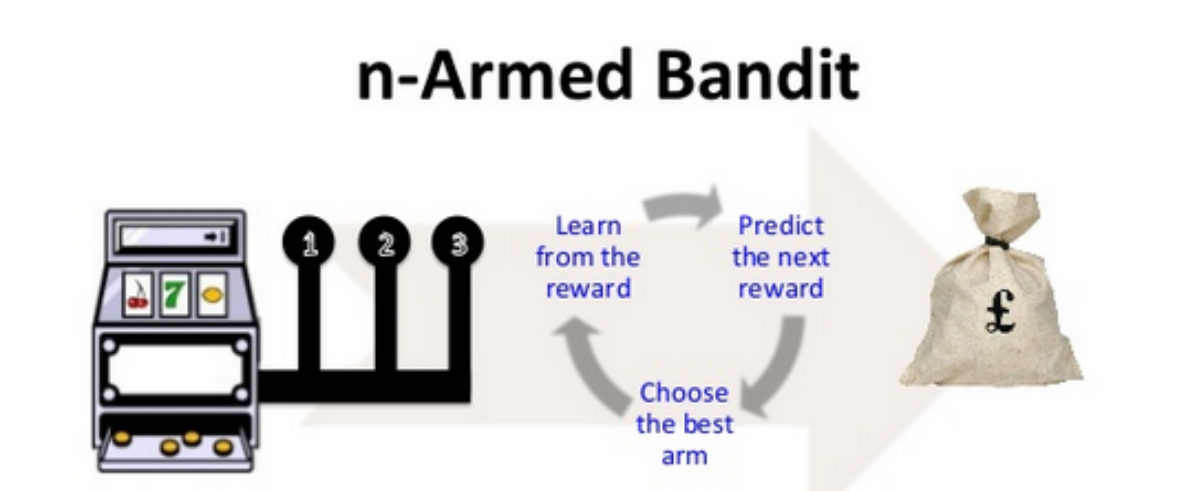
$$\sum_i \#\mathcal{I}(i)||\mathbf{U}_i||^2 + \sum_j \#\mathcal{J}(j)||\mathbf{V}_j||^2.$$

Que pondera cada ítem y usuario por su importancia relativa (#I(i) = cantidad de apariciones del usuario i, #J(j) cantidad de apariciones del item j). En esta implementación se utiliza ALS (alternating least squares) para la optimización.

MULTI-ARMED BANDITS

- Como ya se mencionó, los sistemas recomendadores que reaccionan en tiempo real tiene muchas similitudes con los problemas de Reinforcement learning.

- Un problema clásico de esta área es el de los Multi-armed Bandits (máquinas de casino), en el que se presenta una máquina de casino con múltiples brazos y en cada paso de tiempo t se puede tirar una brazo de la máquina y se recibe una recompensa por ello. Se puede jugar T veces y se busca maximizar la recompensa a corto y largo plazo.



MULTI-ARMED BANDITS

- Debido a que el problema de los sistemas recomendadores tiene finitas acciones posibles (cantidad finita de ítemes para recomendar), se trabaja en tiempo que se podría considerar discreto y se busca maximizar (minimizar) un índice a corto y a largo plazo, el problema se podría modelar como una de Multi-armed bandits.

- Solo que en este caso, en vez de maximizar una recompensa, se busca minimizar lo que los autores llaman ”Pseudo-regret”que mide cuanto pierde el sistema en promedio al recomendar un item sub-óptimo.

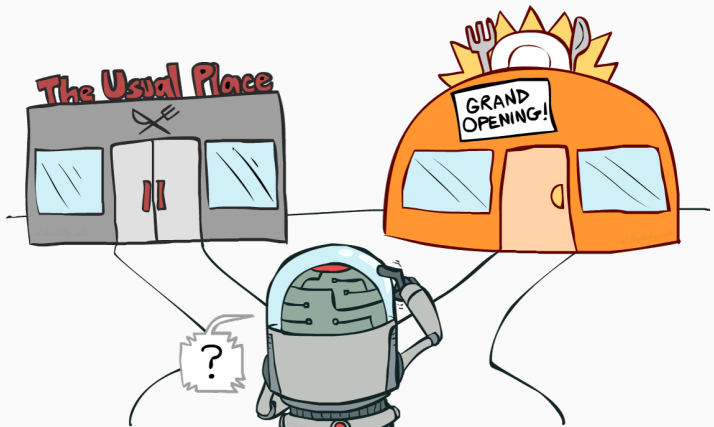
$$\mathcal{R}_T = \sum_{t=1}^{T} \max_j r^*_{i_t,j} - \mathbb{E}[r_t] = \sum_{t=1}^{T} \max_j r^*_{i_t,j} - r^*_{i_t,j_t}.$$

EXPLORACIÓN VS EXPLOTACIÓN

- Como sería intuitivo pensar, mejorar la recompensa a corto plazo, no siempre es lo mejor para el largo plazo o viceversa.

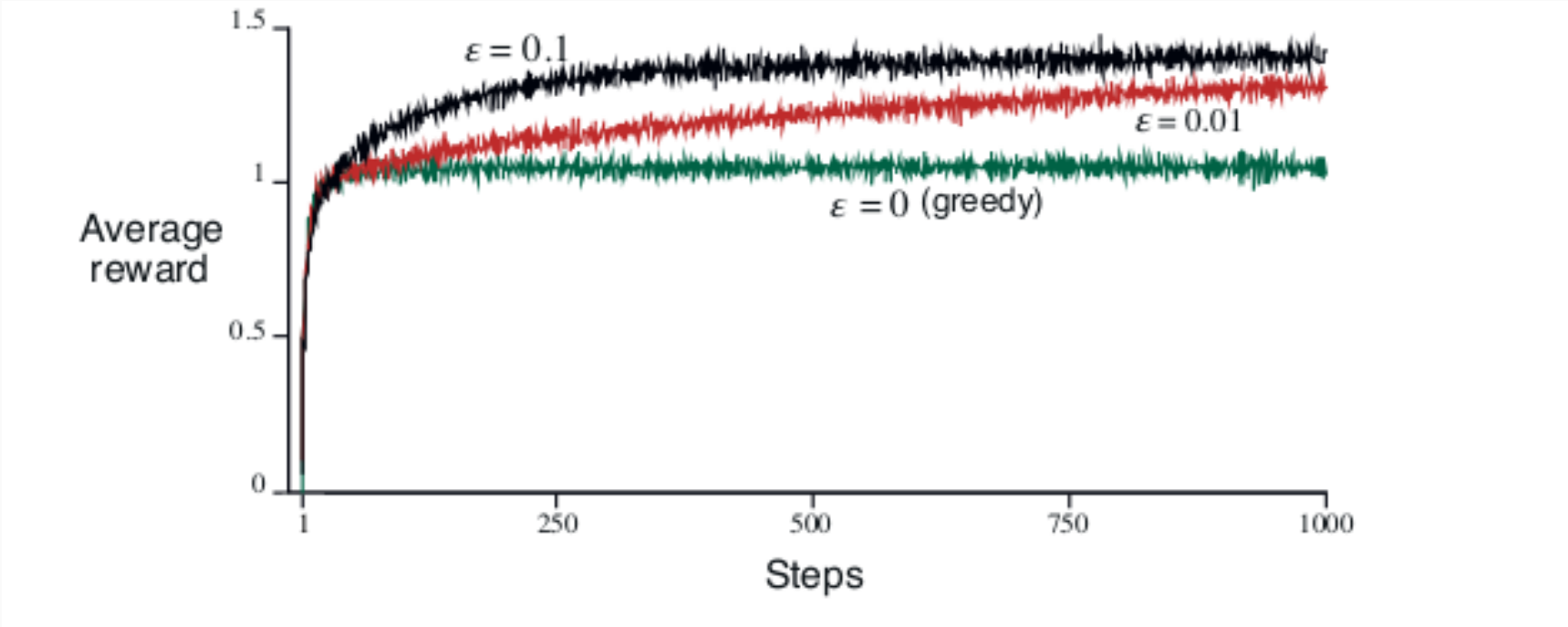
- Explorar nuevas acciones puede llevarnos a conocer mejores opciones de las ya conocidas y descartar otras, pero podría llevar a que la recompensa a corto plazo caiga.

- Este trade-off entre exploración de nuevas alternativas vs explotación de las ya conocidas es un problema típico de los problemas de Multi-armed Bandits.



EXPLORACIÓN VS EXPLOTACIÓN

La forma en que se aborda el problema en este paper y en el contexto de los sistemas recomendadores es con la estrategia ϵ-greedy. Que consiste en escoger la mejor opción que se tenga hasta el momento (explotar) con probabilidad 1 - ϵ y explorar nuevas alternativas con probabilidad ϵ.



ALGORITMO

ALGORITMO

- Una vez ya introducidos los bloques que conforman el algoritmo, se procede a presentar el algoritmo como tal.

- El algoritmo inicial que se presenta lleva por nombre **SeALS** (Sequential ALS).

- En cada paso de tiempo, dado un usuario i el algoritmo asocia una recompensa a cada item j, y el item a recomendar se escoge con la estrategia ϵ- greedy y a partir de la recompensa se actualiza la matriz $R = UV^T$ y la matriz S que contiene solo los elementos conocidos.

ALGORITMO - VERSIÓN 1

Algorithm 1. SeALS: recommend in a sequential context

Input: T_u, p, λ, α **Input/Output:** $\mathbf{R}$, $\mathcal{S}$

for $t = 1, 2, \ldots$ **do**

$(\widehat{\mathbf{U}}, \widehat{\mathbf{V}}) \leftarrow \text{ALS-WR}(\mathbf{R}, \mathcal{S}, \lambda)$

get user i_t and set $\mathcal{A}_t$ of allowed items

$$j_t \leftarrow \begin{cases} \text{argmax}_{j \in \mathcal{J}_t} \widehat{\mathbf{U}}_{i_t} \widehat{\mathbf{V}}_j^T & \text{, with probability } 1 - \min(\alpha/t, 1) \\ \text{random}(j \in \mathcal{A}_t) & \text{, with probability } \min(\alpha/t, 1) \end{cases}$$

recommend item j_t and receive rating $r_t = r_{i_t, j_t}$

update $\mathbf{R}$ and $\mathcal{S}$

end for

ALGORITMO - VERSIÓN 2

- El problema con esta implementación es que ALS-WR requiere de mucho cómputo al correrse en cada paso de tiempo.

- Luego, la solución encontrada fue solo correr el ALS-WR cada T_u pasos pero con una versión basado en mini-batches. El nuevo algoritmo recibe el nombre de **mBALS-WR** en el que en vez procesar todos los elementos, solo toma algunos al azar. Además esto permite actualizar las matrices más seguido pues requiere de menos cómputo.

Algorithm 2. mBALS-WR: mini-batch version of ALS-WR

Input: $\mathbf{R}$, $\mathcal{S}$, λ, p, **Input/Output:** $\widehat{\mathbf{U}}$, $\widehat{\mathbf{V}}$

Sample randomly $p\%$ of all users in a list l_{users}

Sample randomly $p\%$ of all items in a list l_{items}

$$\forall i \in l_{users},\ \widehat{\mathbf{U}}_i \leftarrow \text{argmin}_{\mathbf{U}} \sum_{j \in \mathcal{J}_t(i)} \left(r_{i,j} - \mathbf{U}\widehat{\mathbf{V}}_j^T \right)^2 + \lambda \cdot \#\mathcal{I}_t(i) \|\mathbf{U}\|$$

$$\forall j \in l_{items},\ \widehat{\mathbf{V}}_j \leftarrow \text{argmin}_{\mathbf{V}} \sum_{i \in \mathcal{I}_t(j)} \left(r_{i,j} - \widehat{\mathbf{U}}_i\mathbf{V}^T \right)^2 + \lambda \cdot \#\mathcal{J}_t(j) \|\mathbf{V}\|$$

EL algoritmo incluyendo **mBALS- WR** se presenta a continuación:

Algorithm 1. SeALS: recommend in a sequential context

Input: T_u, p, λ, α **Input/Output:** $\mathbf{R}$, $\mathcal{S}$

$(\widehat{\mathbf{U}}, \widehat{\mathbf{V}}) \leftarrow \text{ALS-WR}(\mathbf{R}, \mathcal{S}, \lambda)$

for $t = 1,\ 2,\ \ldots$ **do**

get user i_t and set $\mathcal{A}_t$ of allowed items

$$j_t \leftarrow \begin{cases} \text{argmax}_{j \in \mathcal{J}_t}\ \widehat{\mathbf{U}}_{i_t} \widehat{\mathbf{V}}_j^T & \text{, with probability } 1 - \min(\alpha/t, 1) \\ \text{random}(j \in \mathcal{A}_t) & \text{, with probability } \min(\alpha/t, 1) \end{cases}$$

recommend item j_t and receive rating $r_t = r_{i_t, j_t}$

update $\mathbf{R}$ and $\mathcal{S}$

if $t \equiv 0 \mod T_u$ **then** $(\widehat{\mathbf{U}}, \widehat{\mathbf{V}}) \leftarrow \text{mBALS-WR}(\widehat{\mathbf{U}}, \widehat{\mathbf{V}}, \mathbf{R}, \mathcal{S}, \lambda, p)$ **end if**

end for

EXPERIMENTOS

DATASETS

Se hacen dos experimentos para validar el algoritmo, para esto, se utilizaron 4 datasets:

1. Movielens 1M (películas)
2. Movielens 20M (películas)
3. Douban (películas)
4. Yahoo (música)

	Movielens1M	Movielens20M	Douban	Yahoo!
Number of users	6,040	138,493	129,490	1,065,258
Number of items	3,706	26,744	58,541	98,209
Number of ratings	1,000,209	20,000,263	16,830,839	109,485,914

COMPARACIÓN DE ALGORTIMOS

El algoritmo SeALS-WR se compara con varias baselines:

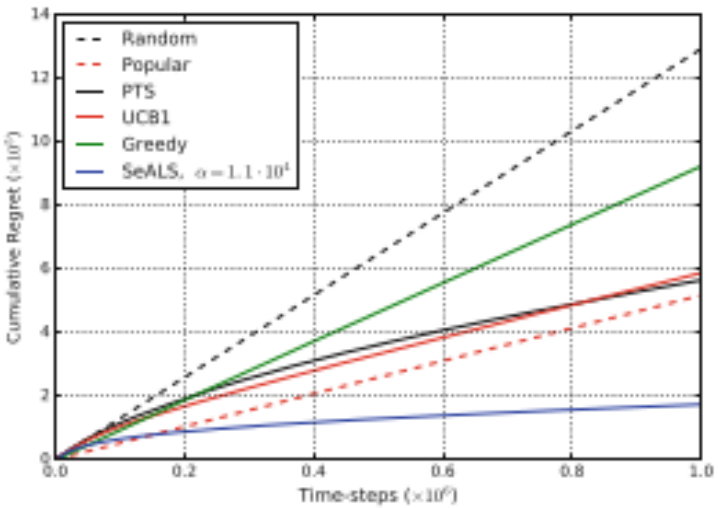
- **Random:** Un item random se recomienda
- **Popular:** Se recomienda un ite popular
- **UCB1:** Se considera cada recompensa r como una realización independiente de la distribución v. Se recomienda un item sin tomar en cuenta la identidad del usuario en cuestión.
- **PTS:** Se crea un modelo estadístico de la matriz R. La recomendación se hace después de un sampleo que se hace con un filtro de particulas.

EXP1: IMPACTO DE LA EXPLORACIÓN

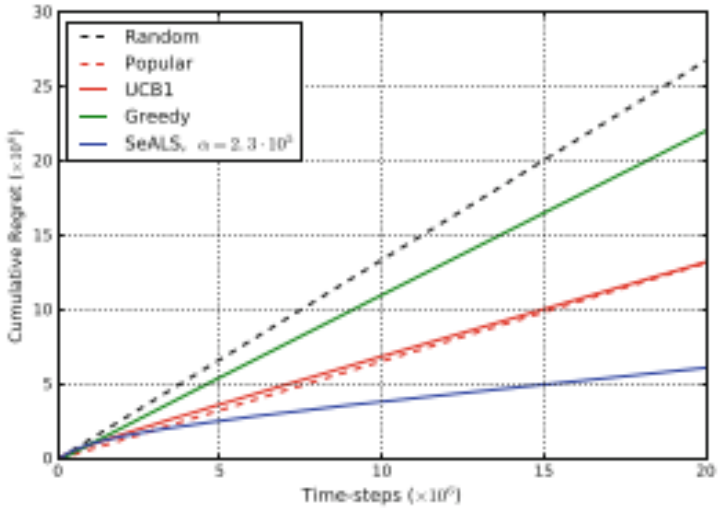
- El primer experimento pretende demostrar el impacto que tiene la exploración en los resultados.

- Se prueba el algoritmo primeramente con un $\alpha > 0$ y luego con $\alpha = 0$ (greedy). En la siguiente figura se presentan los resultados obtenidos en los cuatro datasets en el "Pseudo Regret"(R_T) después de 50 iteraciones.

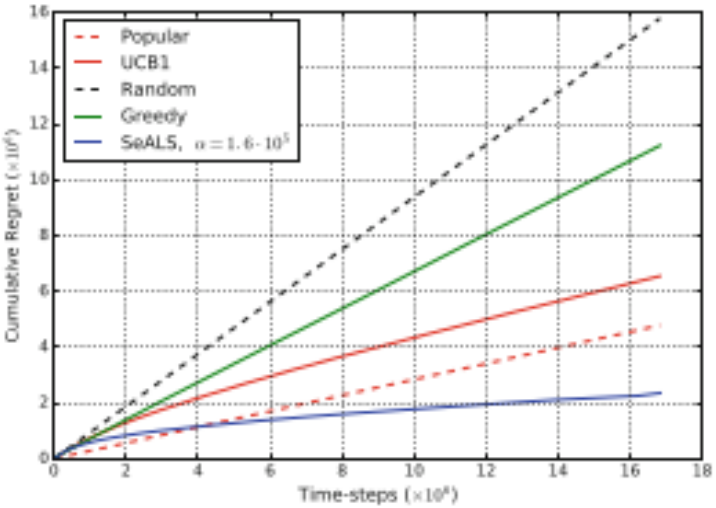
EXP1: IMPACTO DE LA EXPLORACIÓN



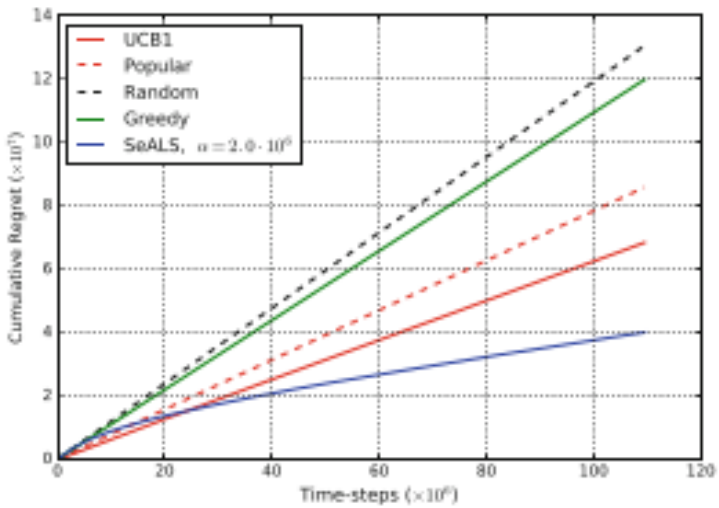
(a) Movielens1M



(b) Movielens20M



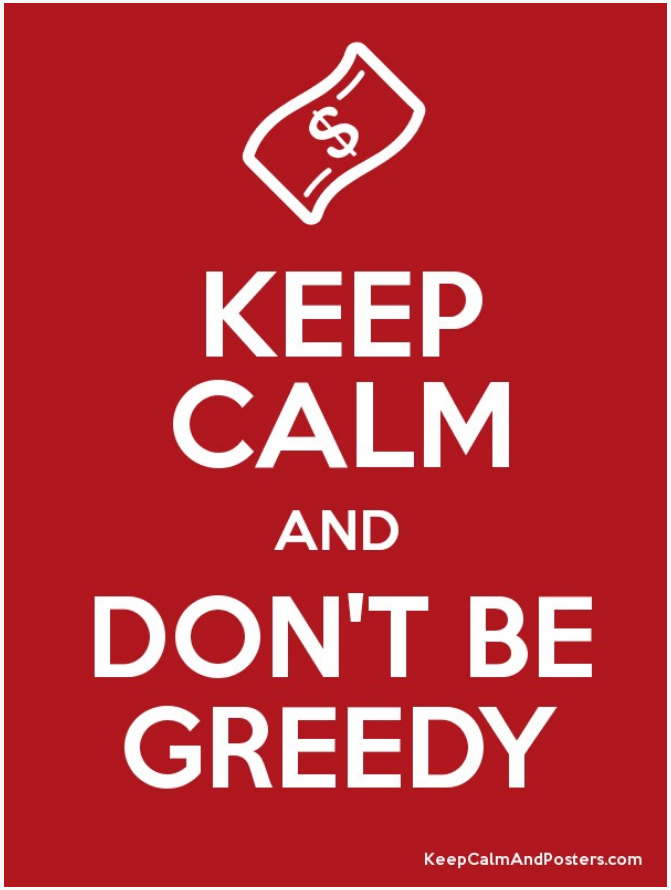
(c) Douban



(d) Yahoo

EXP1: IMPACTO DE LA EXPLORACIÓN

Como se puede ver, en todos los casos anteriores, la exploración produce mejores resultados que actuar greedy (sin exploración).

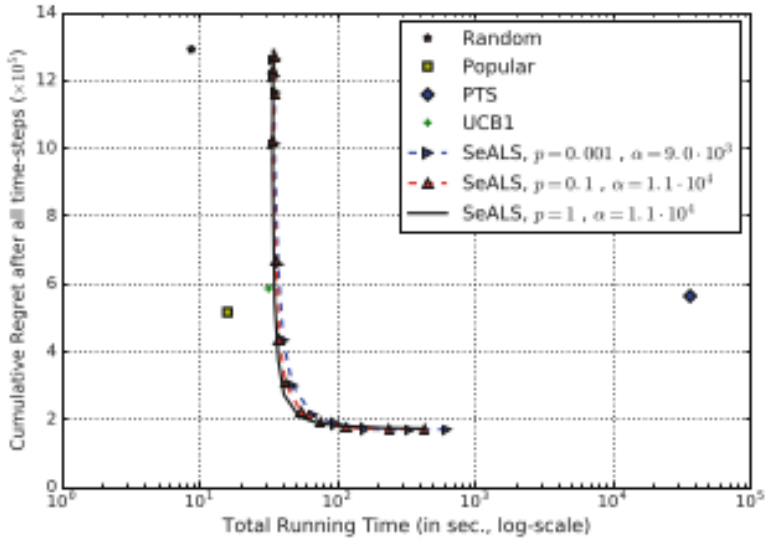


EXP2: IMPACTO DE LA ESTRATEGIA DE ACTUALIZACIÓN

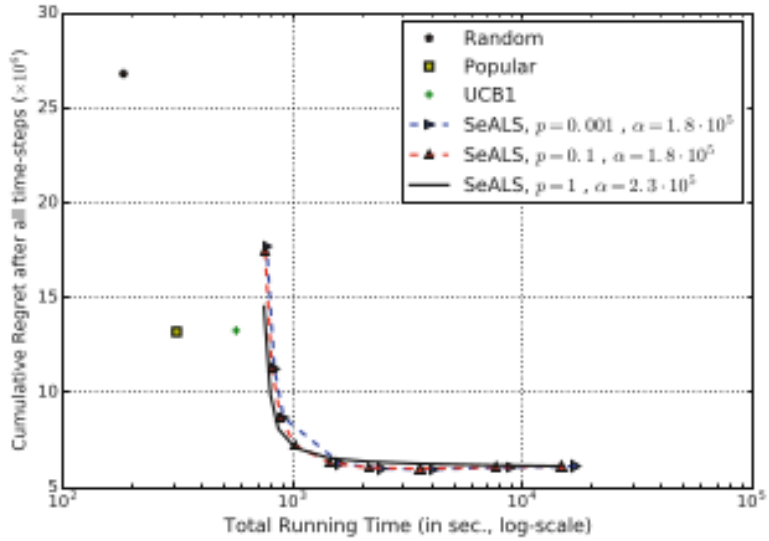
- Se probaron varias configuraciones, variando tato T_u (periodo de actualización) y variando el %p (% de elementos que se actualizan en el mBALS-WR) cada T_u pasos.

- Esto para encontrar el mejor trade-off entre el Pseudo-regret y el tiempo de cómputo. O equivalentemente, para encontrar el mejor trade-off entre %p y T_u que minimice el Pseudo-regret.

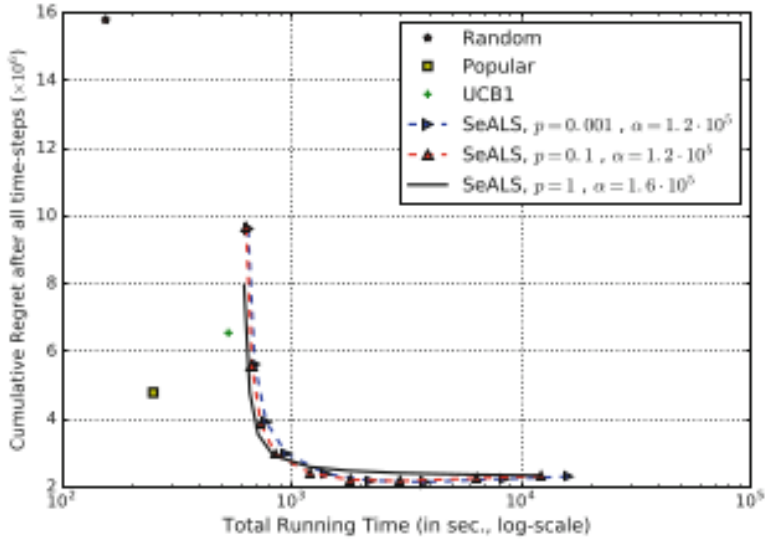
EXP2: IMPACTO DE LA ESTRATEGIA DE ACTUALIZACIÓN

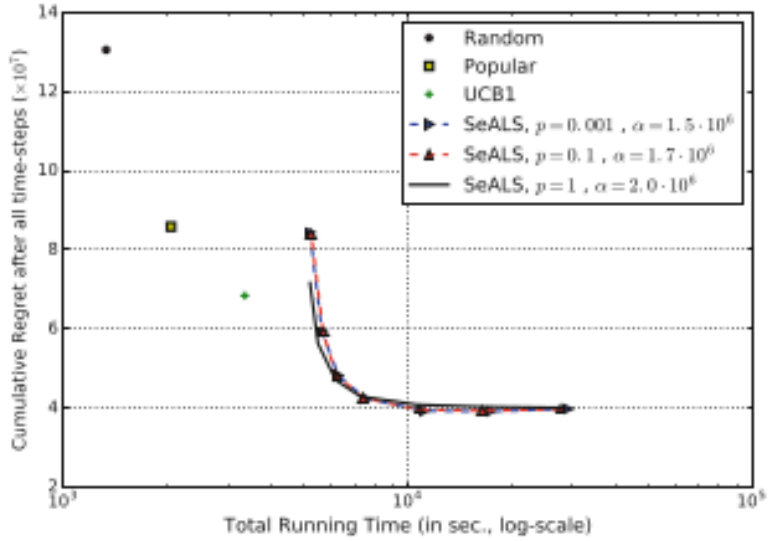


(a) Movielens1M



(b) Movielens20M





EXP2: IMPACTO DE LA ESTRATEGIA DE ACTUALIZACIÓN

- A partir de estos resultados se puede concluir que el **%p no tiene tanta importancia en los resultados mientras se encuentre un T_u adecuado**.

- También se puede ver que los mejores resultados con un %p pequeño se obtienen también con valores bajos de α (probabilidad de explorar). Esto puede deberse a que no actualizar todos los elementos cada T_u pasos significa no actuar de manera óptima según lo que se conoce, esto significa que explora implícitamente.

CONCLUSIONES

CONCLUSIONES PAPER

Las Conclusiones de los autores del paper son las siguientes:

1. La exploración es necesaria para mejorar las recomendaciones a largo plazo.
2. El algoritmo SeALS-WR es adecuado para hacer recomendaciones en tiempo real, pues logra realizarlas en **menos de un milisegundo**.
3. Algortimo completamente escalable con la versión mBALS-WR.
4. Posibles extensiones al algoritmo agregando contexto a la recomendación o usando otra técnica para tratar el dilema de exploración vs explotación pueden ser utilizadas.

CONCLUSIONES PERSONALES

1. Paper presenta una idea muy interesante para recomendaciones en tiempo real.
2. Relaciona dos áreas aparentemente muy distintas y logra extraer un buen algoritmo con esto.
3. Faltó ahondar en lo que es Reinforcement Learning.
4. Faltaron más experimentos (Añadir contexto, otros modelos, otras técnicas de Reinforcement Learning, etc)

REFERENCIAS

REFERENCIAS

Frédéric Guillou, Romaric Gaudel y Philippe Preux. “Large-Scale Bandit Recommender System”. En: Machine Learning, Optimization, and Big Data: Second International Workshop, MOD 2016, Volterra, Italy, August 26-29, 2016, Revised Selected Papers. Ed. por Panos M. Pardalos y col. Cham: Springer International Publishing, 2016, págs. 204-215. ISBN: 978-3-319-51469-7. DOI: 10.1007/978-3-319-51469-7_17. URL: https://doi.org/10.1007/978-3-319-51469-7_17.

FIN