Universidad de Barcelona

Un Recomendador para SnowFlake

[Escriba el subtítulo del documento]

Bruno Vescovi Arquillo

Contenido

Introducción

Objetivo

En este trabajo valoraremos la factibilidad y utilidad de un recomendador en una herramienta de colaboración específica.

Contexto

Para este trabajo hemos utilizado como contexto el código fuente SnowFlake que salió bajo el nombre comercial *Kezmo* en modo beta en agosto de 2016. La cual está diseñada para un público empresarial o educativo en el que sea necesaria la conversación entre los usuarios a la vez de que haya contenido que ayude a modelar la realidad deseada.

El objetivo de esta herramienta es el de facilitar la colaboración interequipos y interorganizaciones. La idea es que los elementos de la misma herramienta puedan ser adaptables a las necesidades de cualquier negocio. Con esta idea en mente, surgieron las dos aplicaciones a partir del mismo codigo fuente. *Closenet* y *Kezmo*.

Las herramientas de colaboración podríamos clasificarlas en dos tipos, herramientas estructuradas y desestructuradas. Las explicaremos a continuación.

Las herramientas estructuradas serían el correo electrónico, sharepoint y otras herramientas que permiten estructurar el contenido. Funcionan con formularios que permiten al usuario describir la realidad de su trabajo. Gestionan muy bien el contenido y es muy fácil encontrar lo que buscas, gracias a su visión estructurada.

El problema de estas herramientas surge en que los usuarios acaban teniendo que rellenar grandes formularios que al final nadie consulta y que enlentecen el proceso de colaboración cuando hubiese sido mucho más rápido una simple consulta por chat.

Es por eso que las herramientas desestructuradas han estado proliferando más últimamente. El avance de la tecnología y la inmediatez de la información ha empujado a que se utilicen herramientas más volátiles, centradas en un chat que permite un intercambio rápido, fácil y desestructurado.

El error de este tipo de colaboración es que es más difícil saber dónde se tomaron las decisiones, o que pasó mientras estabas fuera o la semana anterior. Ya que el chat es un elemento vivo y las conversaciones antiguas quedan enterradas bajo las nuevas conversaciones.

SnowFlake surge del interés de unir los dos mundos. La rapidez y la facilidad de un chat, con la comodidad de encontrar información y la posibilidad de un formulario que te guie durante el proceso tal como era habitual en las herramientas estructuradas.

Kezmo es una aplicación que surgiendo de este código base implementa una aplicación web junto con dos clientes móvil (Android y IOS). Esta aplicación aparece en fase alpha en junio de 2016 utilizada tan solo por el equipo de desarrollo y en fase beta privada en agosto del mismo año.

Recomendadores

Los recomendadores son un tipo de software que, por la cantidad de datos que actualmente se generan en las herramientas informáticas, ha venido en auge en los últimos años.

A diferencia de la búsqueda tradicional en la que buscamos algo que queremos y sabemos que existe a partir de términos o condiciones que conocemos, en la recomendación nos surge algo que no conocemos, que no sabíamos que queríamos o incluso desconocíamos su existencia.

El objetivo de los recomendadores es el de predecir cuanto a un usuario le va a gustar un ítem. Para ello se basa en distintos datos. (El comportamiento anterior del usuario, la relación con otros usuarios, el parecido entre ítems...)

No tiene sentido recomendar items que son obvios (eg. A un user que le gustó Harry Potter 1 recomendarle Harry Potter 2). Lo interesante de las recomendaciones es precisamente la de dar a conocer ítems con relaciones que no podemos conocer a simple vista.

Para entender mejor este tipo de software analizaremos un recomendador de películas. Pues es un sencillo ejemplo que nos permitirá explicar el objetivo de los mismos de forma menos abstracta.

¿Cómo podemos elegir qué película ver?

Tiene que ser una película que aún no haya visto y que me vaya a gustar. Este no es un problema fácil. Es imposible que el usuario te diga que película le va a gustar, porque precisamente aún no la ha visto.

¿Cómo podemos sugerirle al usuario una película que le vaya a interesar?

Lo que podemos hacer es pedirle al usuario que valore películas que **si** ha visto, y de esta forma generar una especie de *perfil de usuario* definiendo que películas le gustaron y que películas no. Con esta información, y la de muchos otros usuarios que hayan hecho lo mismo, podemos buscar otros usuarios que hayan valorado de forma parecida las películas que el usuario valoro y sugerirle alguna película que estos usuarios hayan valorado de forma positiva y que el usuario aún no haya visto. De esta forma realmente nunca podremos estar seguros de que la película recomendada será interesante para el usuario, pero tenemos muchas más posibilidades que si le recomendamos una película al azar.

El problema de esta solución es:

> ¿Qué película le recomendamos a un usuario nuevo?

¿Cómo incentivamos a los usuarios a valorar las películas para generar los datos suficientes para poder empezar a recomendar películas? (Cold Start)

Podemos clasificar los recomendadores en dos grandes grupos. A continuación detallaremos los mismos.

Content based: Este tipo de recomendadores examina las propiedades de los ítems a recomendar y basa sus recomendaciones respecto a esas propiedades. Por ejemplo, si un usuario ha visto muchas películas de romance, le recomendará una película de este género. Hay muchas formas de conocer que genero le gusta a un user y de que genero es una película. Podemos clasificar los items en N distintas categorías y darle un valor para cada película a cada genéro, consiguiendo así un vector por item definiendolo en las categorías que tengamos. Para los users podemos hacer lo mismo, cada vez que un user valora una película podemos actualizar su perfil dependiendo de los valores que esa película tenga para cada categoría.

Finalmente podemos buscar que película congenia más con el user elegido a partir de intentar maximizar el resultado de multiplicar el vector descriptor del user por el vector descriptor de la película.

Collaborative filtering: Este tipo de recomendadores se basa en la similitud entre ítems y/o entre usuarios. Los ítems recomendados a un usuario son aquellos preferidos por usuarios con preferencias similares o ítems parecidos a los preferidos por él. Por ejemplo si tenemos este set de datos:

	Juan	Norman	Claudia	Mónica	Ramón
Piratas del caribe		5	4	1	1
Harry Potter		4	4	4	
Ice Age		1		5	4
Espartaco			5	4	2
El Padrino					
Madagascar		1	1		5

Basados en estos datos podríamos intentar predecir que puntaje le darían cada uno a cada película que no ha visto.

	Juan	Norman	Claudia	Mónica	Ramón
Piratas del caribe		5	4	1	1
Harry Potter		4	4	4	4
Ice Age		1	3	5	4
Espartaco		5	5	4	2
El Padrino					

Madagascar		1	1	4	5

Podemos ver que Claudia y Norman valoran de manera parecida Piratas del Caribe, Madagascar y Harry Potter, por lo tanto podríamos recomendarle a Norman la película Espartaco, la cual no ha visto y Claudia ha valorado bien. Por otro lado Mónica y Ramón han valorado de manera similar Piratas del Caribe e Ice Age, así que podríamos recomendarle a Mónica Madagascar, la cual Ramón evaluó bien y a Ramón Harry Potter la cual Mónica evaluó bien.

Claudia la única película que no ha visto es Ice Age, la cual Norman no valoró muy bien pero Mónica sí. Por el parecido que Claudia tiene a Norman y a Mónica intentaremos predecir el valor que Claudia le daría. Y en función de eso le recomendaremos o no.

En este ejemplo es bastante sencillo ver qué recomendaciones hacer, tenemos pocos datos y casi todos los usuarios han valorado casi todas las películas.

Como podemos ver si un usuario no ha valorado películas no podemos saber que películas recomendarles y si una película no ha sido valorada tampoco podemos recomendarla.

Hay muchos tipos de recomendadores, no todos le piden al usuario que valore los elementos, datos explícitos, sino también procesan datos dados por el usuario de manera implícita (como el tiempo que pasa en un perfil de FaceBook o las búsquedas y las páginas que visita cuando busca en Google) estos datos, sean tanto implícitos como explícitos, multiplicados por el número de usuarios que potencialmente pueden utilizar nuestro sistema son muchos, muchísimos, datos. Procesar estos datos y generar de ellos conclusiones útiles es un problema muy difícil que expertos de todo el mundo están intentando resolver.

Es el mundo del Big Data.

SnowFlake

Snowflake es un código base desarrollado por la empresa Uruguaya *OrangeLoops* que salió en febrero de 2016 en la *AppleStore* bajo el nombre *Closenet,* como una aplicación para colaboración vía *Bluetooth* y *Wi-Fi* (usando la base de *Multipeer* de Apple). Esta aplicación apuesta por la colaboración por cercanías y es gratuita.

Snowflake es una aplicación de colaboración que intenta unir el mundo de colaboración de la vieja guardia, las cuales son herramientas muy estructuradas que permiten al usuario encontrar contenido de forma fácil pero que deja de lado la colaboración ágil, con las nuevas tendencias de herramientas de colaboración que

mediante un chat como eje central de la aplicación permiten una colaboración más ágil y adecuada para el uso actual de la tecnología.

Utilizando el mismo código y ampliándolo con una pata server este mismo código aparece, en agosto de 2016, bajo el nombre *Kezmo* como una aplicación de colaboración para equipos.

El código base SnowFlake basa su diseño en los siguientes conceptos que serán utilizados durante el texto de este trabajo.

User: Usuario del sistema.

Container Item: Un container item es un elemento estructurado de la aplicación. Serían las hojas si viéramos el sistema como un árbol. Puede haber de distintos tipos (eg. imagen, archivo, link, tarea, issue, ubicación...).

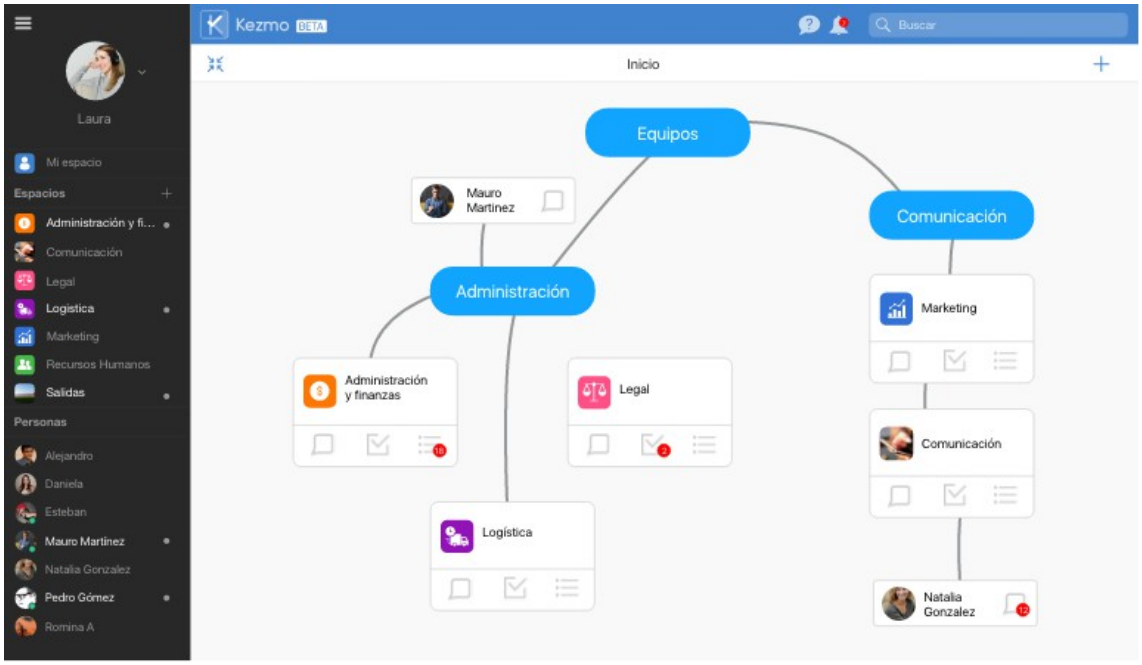
Container: El container es un contenedor de container items. Dentro del mismo se guardan los elementos estructurados del sistema, tanto sean creados desde los formularios o desde el chat.

Flow Item: El flow item es un mensaje de chat. Puede tener distintos tipos dependiendo del contenido que exprese. Puede tener relacionado un container item, guardándolo así en el container correspondiente.

Flow: El flow es una sala de chat. En él se envían los flow items y es el medio de colaboración rápida y desestructurada de la herramienta.

Space: Un space es un contexto de colaboración. A esta entidad están ligados un equipo de users, un conjunto de flows y un conjunto de containers. Para cada equipo con el que colaboro tendría por lo menos un space, podría tener más de uno para subdividir responsabilidades (eg Equipo de diseño, Equipo de desarrollo, General, Trabajo de fin de grado...).

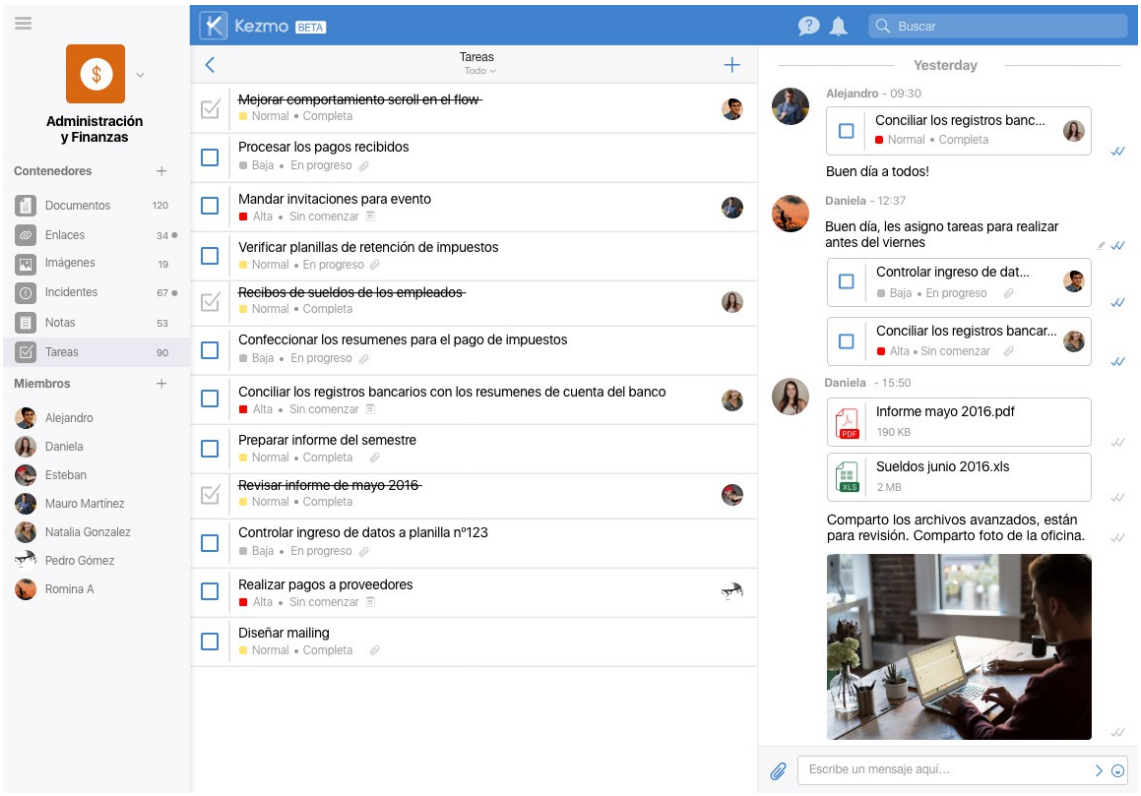
En *Kezmo* tenemos como pantalla inicial el Mindmap, que sirve para que cada usuario añada en el los spaces que más utiliza y que pueda organizarlos de forma jerárquica y realizar búsquedas en scopes específicos (las flechas direccionan la búsqueda a los spaces hijos del space seleccionado).



Entonces tenemos la vista de space que sirve como contexto de colaboración, cada space tiene un administrador que puede añadir usuarios y cambiar los roles de los mismos.

Para la colaboración entre users internamente se crea un space de tipo p2p con solo dos users. Eso significa que a parte de todos los spaces que creemos para colaborar alrededor de distintos temas tendremos, dentro de los datos, spaces que tan solo tengan dos usuarios. Pues son los que utilizamos para colaborar entre usuarios.

Para cada space tenemos algunos containers por defecto (imagenes, documentos, tareas, links, eventos...). Y un sólo flow, que es el chat que permite la colaboración ágil.



En estos screenshots podemos ver como hay dos spaces, en uno están abiertos los documentos y en otro las tareas. En los dos casos el chat aparece a la derecha para mantener la colaboración ágil y a la izquierda aparece el menú del space. Mientras en el panel central tienes un espacio de trabajo para administrar los containers y container items.

Cuando creas un elemento en el flow se crea un flow item en la base de datos. Y con una nomenclatura específica podemos crear container items, que se guardan en los containers, desde el flow. Cuando esto sucede se crea un flow item del tipo del container item que tendrá asociado el elemento creado.

Un recomendador en una herramienta de colaboración

Está bien, hasta aquí hemos hablado de lo que es un recomendador y de lo que es una herramienta de colaboración, así como la herramienta de colaboración específica sobre la que se basa este trabajo. Ahora bien, hemos dicho que este trabajo quiere valorar la viabilidad y la utilidad de un recomendador en este tipo de herramientas.

Pensemos en como utilizamos las herramientas de colaboración. En ellas compartimos archivos, tomamos decisiones, planificamos reuniones, escribimos documentos...

Todas estas tareas se hacen de forma grupal o al menos son revisadas por alguien, sino no tendría mucho sentido que fueran realizadas en el contexto de una herramienta de colaboración.

¿Qué nos interesaría que un recomendador nos recomendase?

Como respuesta a esta pregunta tenemos muchas posibilidades. Aquí hemos analizado dentro de la herramienta Kezmo que preguntas nos interesaría tener respondidas. También tiene que ver con qué información es relevante para nosotros y nos es difícil de obtener de manera sencilla:

- Qué flow items de los no leídos de todos mis spaces me interesa leer.

- Qué container items de todos mis spaces son interesantes para mí en este momento.

- Qué posibles relaciones pueden interesarme en mi mindmap.

- Qué personas que yo no tenga en mi primer radio de alcance (comparta un space con ellas) son interesantes para mí.

- Qué spaces para los cuales no tengo acceso son interesantes para mí.

De entre estas posibilidades nos hemos centrado en esta última. Valorar y recomendar contextos de colaboración para un user específico.

En varias de estas posibilidades puede ser que la información recomendada no sea directamente para el user en cuestión.

Por ejemplo, en la posibilidad elegida, no le vamos a recomendar a un user un space al que no tiene permiso para acceder. Esta recomendación se la podemos hacer al administrador del sistema, o del space, de forma que él pueda decidir a qué user añade a qué space.

Si estamos trabajando en una organización de 10 personas capaz este tipo de recomendación no es muy útil, ya que la información que hay para absorber no es muy grande. Podrás dividir a los trabajadores en equipos según su función y podemos crear contextos de colaboración por equipos. Seguramente tendremos entre 3 y 6 contextos de colaboración dependiendo del uso que le demos. Así no es muy difícil saber a quién añadir a cada contexto.

Pero en el caso de una gran organización, con centenares o millares de personas el número de contextos será mucho mayor y es por eso que es más difícil saber que personas tienen que trabajar en qué equipo. Además los contextos pueden ir cambiando. Puede ser que alguien empiece trabajando con un equipo pero tenga más afinidad con otros y acabe trabajando con ellos, o que existan contextos de colaboración que al no tener acceso no pueda dar su opinión pero pueda ser interesante.

Entonces en este caso llegamos a la conclusión de que hay demasiada información para que alguien pueda abarcarla toda, y menos aún saber quién es más indicado que trabaje con quién bajo qué objetivo.

Para un administrador de sistema sería muy interesante que, de alguna manera, la herramienta le sugiriera qué personas deberían trabajar bajo qué contexto de colaboración.

¿Cómo podría la herramienta saber quién debería colaborar con quién?

Hemos dicho que una herramienta de colaboración se utiliza para compartir información. Esto genera muchas trazas electrónicas, como cuantos mensajes alguien envió o recibió en un contexto de colaboración, cuantos elementos compartió, cuanto tiempo pasó en ese contexto... Si de alguna manera pudiéramos procesar estos datos para que se transformaran en sugerencias de colaboración un administrador podría tener más información a la hora de decidir quién debería integrar un contexto de colaboración.

En este trabajo vamos a diseñar y construir un recomendador de contextos de colaboración. Generaremos un rating de manera implícita a partir de la colaboración que un user tenga en un space y comparando el puntaje que cada user tenga para los spaces comunes, tal como hacíamos con las películas, decidiremos que spaces podemos sugerirle a cada user.

Objetivo del TFG

Recapitulando, el objetivo de este trabajo era el de valorar la factibilidad y utilidad de un recomendador en una herramienta de colaboración específica.

Ahora que hemos explicado el contexto en el que estamos, qué es un recomendador, qué es una herramienta de colaboración y qué herramienta de colaboración específica hemos decidido utilizar así como qué tipo de recomendación vamos a generar en este sistema. Llego la hora de definir el proceso para ello.

Primero vamos a estudiar el estado de arte de los recomendadores y evaluar qué tipo de recomendador se ajusta más a nuestras necesidades. Así como de qué manera vamos a predecir el rating que un user daría a un space que no haya valorado.

Después vamos a evaluar cómo vamos a hacer que un user valore un space que utiliza. Estos datos vamos a tener que calcularlos de manera implícita pues dentro de nuestro contexto hemos valorado que pedirle de forma explícita una valoración al usuario podría tener sentido para corroborar y ajustar nuestros datos pero no para utilizarlos como base de nuestro sistema de recomendación.

Después implementaremos el cálculo del rating implícito de forma que pueda ser ajustable y escalable conforme cambien los datos o los coeficientes de impacto que cada dato tendrá en nuestro rating calculado. Y buscaremos una librería que implemente el método de predicción decidido.

Finalmente generaremos primero unos datos de prueba para probar la funcionalidad de nuestro sistema. Probaremos también con los datos de movielens el método de predicción decidido. Y por último probaremos nuestro sistema con los datos reales recopilados desde la liberación beta de *Kezmo*.

Recomendadores

Como hemos visto en la introducción los recomendadores son un tipo de software el cual intentará predecir el interés de un user por un item. A continuación explicaremos diversos métodos de recomendación y después de evaluarlos elegiremos el más adecuado para nuestro contexto.

Dentro de los dos grandes grupos que hemos explicado anteriormente, nuestro contexto se inclina más por el de collaborative filtering. Ya que para el content based deberíamos buscar propiedades que se parecieran entre los elementos a recomendar. En este caso spaces.

Aunque tenemos título y descripción en un space, estos elementos son realmente pobres a la hora de decidir qué space sería parecido a otro. Es por eso que para decidir que un space es similar a otro tendremos que hacerlo a partir de la información que relaciona un user con un space. Y a partir de esa misma información deberemos decidir cuándo un user es similar a otro.

Dentro de collaborative filtering podemos dividirlo entre user similarity, item similiraity e híbrido. Explicaremos a continuación las distinciones entre los mismos.

En user similarity se buscan perfiles con ítems valorados de forma parecida y se recomiendan ítems que no tengan en común y que sean bien valorados.

En item similarity se consideran parecidos los ítems que cada user ha valorado de forma parecida y se recomienda a los users ítems que no conozca y que sean parecidos a los bien valorados por él.

Los sistemas híbridos son los que a la hora de hacer las predicciones tienen en cuenta las dos aproximaciones. Brindando así una mayor flexibilidad de aprendizaje.

El mayor problema de este tipo de recomendadores viene dado por el fenómeno que se da en la mayoría de estos sistemas en la matriz de utilidad (matriz que relaciona cada pareja user-item). Esto es conocido como Sparsity Matrix Problem. Significa que al haber muchos ítems en nuestro sistema hay muchos de los cuales los users no han valorado por lo tanto tenemos 0 en esos valores. Y como resultado las recomendaciones sufren de precisión.

La entrada de un algoritmo recomendador depende del algoritmo recomendador pero, en general, se basa en una matriz de utilidad, en la que un eje es ocupado por los users y el otro por los ítems. Tenemos entonces para cada pareja de user e ítem un puntaje, que es considerado el valor de utilidad que tiene ese ítem para ese user.

La salida de un algoritmo recomendador será una matriz de predicción, en la que los ejes son los mismos pero solo tendrán valores los elementos que no tengan valor, o tengan valor nulo, en la matriz anterior. Esto es porque solo vamos a predecir valores para las parejas user-item que no tengamos un puntaje antes de la utilización del recomendador. Ya hayamos calculado el puntaje de manera implícita o explícita.

Para la predicción de los puntajes hay distintos métodos que hemos evaluado.

K Nearest Neighborhood

KNN es uno de los algoritmos que mejor funciona en la práctica y que es más fácil de explicar y de entender. Es utilizado en diversos escenarios con muy buenos resultados. Es incluso uno del top 10 algorithms in data mining (IEEE in 2006).

Es un algoritmo *non parametric lazy learning.* Lo cual significa que no le atribuye a cada dato un parámetro específico, es decir no hace ninguna suposición sobre la distribución de los datos subyacente. Y que no tiene una fase de training, o es muy mínima. Es decir que no utiliza los datos de training para hacer ninguna generalización.

Esto significa que este algoritmo necesitará de todos los datos de training para la parte de test. Esto supone una fase de test más costosa, pues comparará con todos los datos de training o un subconjunto significativo de los mismos.

Se basa en tener un vector de features por elemento. Y encontrar, dados estos vectores, los K elementos más próximos al elemento objetivo. Suponiendo que estos elementos serán los deseados por alguien que valore positivamoente el elemento objetivo.

Para ello hace la suposición que el vector de features define una posición en un espacio multidimensional y calcula la distancia a cada uno de los otros vectores. En general el algoritmo más utilizado para medir la distancia es el de Euclidean.

Este algoritmo se basa en un conjunto de vectores de training y, en su forma clasificatoria, para cada uno de ellos se le define una clase de entre un conjunto más pequeño que el de los vectores. Lo que va a hacer el algoritmo en su forma clasificatoria es encontrar los elementos más cercanos y a partir de ello decidir de qué clase va a ser el elemento objetivo.

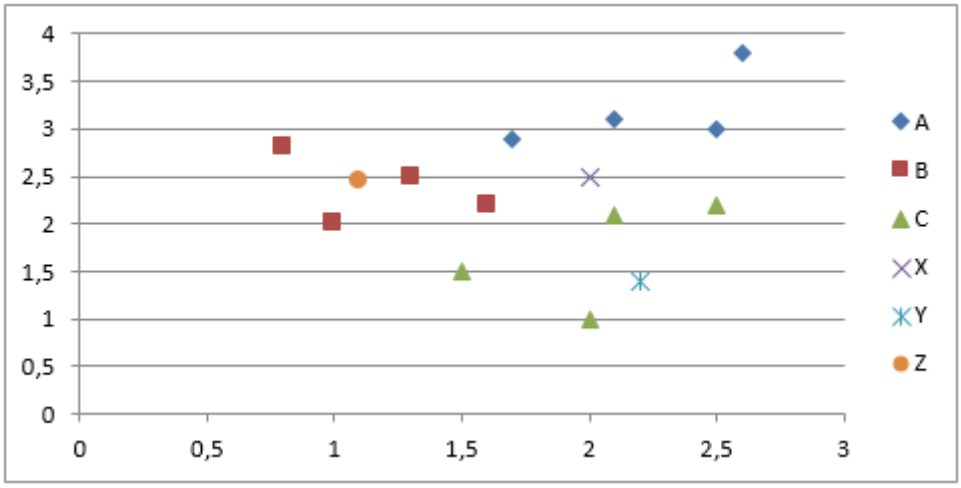
En la forma de Density Estimation, lo que hará es buscar los k elementos más cercanos y para cada uno le dará un coeficiente de acción calculado por la fórmula:

$$p(x) = \frac{k/n}{V}$$

Donde n es el número total de elementos y V el volumen del hipercubo que rodea el elemento sobre el cual calculamos el coeficiente con centro en el elemento del cual estamos intentando predecir los valores.

La otra entrada que requiere el algoritmo es el valor K, el cual decidirá cuantos elementos más cercanos comparará a la hora de hacer una predicción. Este número es muy importante a la hora de hacer nuestras predicciones pues dependiendo del mismo los resultados obtenidos pueden ser distintos.

Para explicar esto vamos a imaginarnos que nuestro vector de features es un vector bidimensional y de esta forma cada vector significa una posición en el eje x/y.



En este caso tenemos cuatro elementos de las clases A, B y C y queremos definir una clase para X.

Si K es 1, la clase del elemento X será C, pues es el elemento más cercano. En cambio sí K es 3 entonces tendremos dos elementos de la clase A y uno de la clase C por lo tanto X será de la clase A.

Justo este ejemplo es un poco difícil porque X está en medio. Pero si intentamos valorar la clase de Y o Z el problema es mucho más sencillo. Es por eso que este algoritmo suele funcionar bien para conjuntos lejanos de elementos.

Es más sencillo entender este algoritmo en un plano bidimensional pues estamos acostumbrados a medir las distancias en este plano y podemos ver que elementos son los más cercanos a X.

KNN aplicado a los recomendadores puede aplicarse tanto a los users como a los ítems, como a los dos. Dependiendo de lo que queramos hacer. Si lo aplicamos a los ítems buscaremos los ítems más cercanos y podremos así clasificar los ítems en clases. Podría ser que cada clase fuese un género de cine y así, dependiendo de los usuarios definir el género de un nuevo ítem.

Si estamos hablando de users vamos a encontrar los users más cercanos y dados estos users, y restando del conjunto que estos users han valorado el conjunto del user objetivo veremos que ítems nos quedarían y de alguna manera (podemos darle un coeficiente descendente a cada user más lejano y multiplicar este coeficiente por el valor que cada user le dio a cada ítem) predecir el valor que el user le daría a cada uno de esos ítems.

Matrix Factorization

Matrix Factorization es un cálculo que se hace con el producto escalar de dos vectores.

Para que tenga sentido dentro de nuestro contexto, collaborative filtering en system recommendation. Estos vectores deben representar de la misma forma en el mismo orden a los users y a los ítems.

Es decir, si tenemos un vector que describe a una película como [Romance, Comedia, Acción, Drama, Aventura] deberemos tener para cada user un vector que lo describa con esos parámetros también. Que tanto le gustan las películas de romance, comedia, acción, drama y aventura.

Por ejemplo si tenemos una película que puntúa [0.8, 0.2, 0, 0.3, 0.3] y un user que puntúa [0.3, 0.5, 0.8, 0.2, 0.6] al multiplicar estos dos vectores de forma escalar tenemos 0.8*0.3 + 0.2*0.5 + 0*0.8 + 0.3*0.2 + 0.3*0.6 = 0.24 + 0.1 + 0 + 0.06 + 0.18 = 0.58. Y si tenemos otra película que puntúa [0.2, 0.6, 0.8, 0, 0.4] para el mismo user tendremos 0.2*0.3 + 0.6*0.5 + 0.8*0.8 + 0*0.2 + 0.4*0.6 = 0.06 + 0.3 + 0.64 + 0 + 0.24 = 1.24.

Por lo tanto tenemos que la segunda película es más acorde al user que estamos valorando. El problema está en que normalmente no tenemos los datos expresados como los necesitamos así que tendremos que buscar un vector que defina al user como define al ítem. Para esto utilizamos algoritmos de Machine Learning.

Para acercarse cuanto más posible a estos vectores que definan al user y al ítem lo que vamos a hacer es intentar que el cálculo de Matrix Factorization de estos vectores sean lo más cercanos posible a los ratings que conocemos, maximizando así la precisión.

Esto quiere decir que intentaremos que la siguiente fórmula tienda a 0.

$$\sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2$$

Siendo r_{ui} el rating conocido para una película, q_i el vector descriptor del ítem y p_u el vector descriptor del user. Para todos los ratings de las parejas user-item conocidos. Lo que significa esta fórmula es que vamos a buscar los vectores q y p para que, para los elementos conocidos, de la matriz se acerquen lo más posible el producto vectorial de los mismos al valor conocido. Suponiendo que esto nos acercará más en los elementos desconocidos.

Entonces modificamos progresivamente los elementos de los vectores p y q hasta que consigamos minimizar el error. Para ello utilizamos dos formulas que comentamos en el apartado de resultados. Finalmente nos fijamos en el error que tenemos y cuando este error es aceptable nos quedamos con los vectores q y p que serán lo que mejor expresa a nuestros usuarios y items respectivamente.

Los algoritmos de Machine Learning evaluados són los siguientes, los cuales tienen una fase de aprendizaje intentando generar coeficientes para los datos de entrada del recomendador que maximicen la precisión con la fórmula dada. Muchas investigaciones han intentado mejorar la calidad de los sistemas de recomendación utilizando Machine Learning y transformando el problema de la predicción hacia un problema de clasificación.

Smoothing Support Vector Machine

Support Vector Machine es un algoritmo que ha sido muy útil en muchas aplicaciones, como clasificación de texto, de imágenes y reconocimiento facial. Este algoritmo intenta dividir los datos en dos clases, eso lo hace un algoritmo non-probabilistic binary linear classifier. Este algoritmo sitúa los puntos en el espacio de manera y los divide entre las dos clases de la manera más amplia posible. Los nuevos puntos son clasificados en una clase o otra.

Pero el problema de la escasez (sparsity) en la matriz de utilidad hace que no sea muy eficiente a la hora de utilizarlo en sistemas de recomendación. Para resolver este problema muchas veces se han utilizado valores neutros o negativos para estos valores nulos. Pero esto puede hacer perder eficacia a los clasificadores.

En esta sección vamos a evaluar un método que propusieron específicamente para este problema en “Support Vector Machines for Collaborative Filtering”.

Lo que ellos proponen, y logran con buenos resultados, es para todos estos valores nulos utilizar el mismo sistema svm para calcular los valores predichos, inicializándolos con valores por defecto como proponíamos anteriormente y ajustarlos con el propio algoritmo en la fase de training.

Esto supone que los datos que tenemos son más cercanos pues tenemos muchos menos datos faltantes y aunque utilicemos valores predichos por el mismo algoritmo como datos de entrada los resultados del experimento son más precisos que otras técnicas utilizadas para la recomendación en Collaborative Filtering como el de similaridad en users o en ítems.

Factorization Machine

Este método, a diferencia del SVM, permite la estimación de parámetros bajo matrices con mucha escasez de datos. No es que SVM no la permita, pero FM tiene mucho mejores resultados en matrices de ese tipo.

También la complejidad de FM es lineal lo que significa que escala en grandes cantidad de datos como Netflix con 100 millones de valores de training.

Como predictor general, FM, puede trabajar con cualquier feature vector con valores reales. Cuando muchos factorization models trabajan solo con datos de entrada muy restringidos.

FM se basa en que cada uno de los ratings para las parejas user-item es igual al producto escalar de dos vectores, uno por el user y otro por el ítem. Entonces el objetivo de este recomendador es el de generar un vector por cada user y por cada ítem que minimice la diferencia al calcular los ratings con el producto escalar de los vectores comparándolo con el rating dado por el user.

Se dice que FM es un algoritmo Properly Factorized Polynomial Regression. Lo cual significa que es equivalente a los modelos de regresión polinómicos donde los parámetros del modelo son sustituidos por representaciones factorizadas.

La racionalidad detrás de la regresión factorizada de polinomios es la de evitar la necesidad de evaluar las interacciones para estimar los efectos de las interacciones. Mientras la regresión de polinomios necesita esta evaluación y su falta lleva a poca precisión en el algoritmo. La regresión factorizada separa la estimación de los efectos de las interacciones de las ocurrencias de la interacción, lo que lo convierte en un algoritmo mucho más eficiente cuando tenemos una matriz escasa de datos.

Este algoritmo es parecido a svm con un kernel polinómico. Lo interesante de este algoritmo es que ha demostrado empíricamente mejores resultados de los esperados teóricamente. La fortaleza de este algoritmo es que ha trabajado en la debilidad de SVM, el cual es poco preciso con una matriz con pocos datos. FM ha demostrado tener gran efectividad en una matriz con pocos datos, incluso con una matriz con muy pocos datos. Como contrapartida no funciona tan bien como otros algoritmos con una matriz user-item muy poblada. Pero como este no es nuestro caso parece interesante utilizarlo.

El funcionamiento de este algoritmo es parecido al de matrixFactorization. Pero en vez de tener un vector que intentará describir el user y el item tenemos unos vectores de entrada que, de alguna manera, ya describen a los mismos. Entonces tenemos que encontrar una matriz (en realidad serán dos multiplicadas entre ellas) que, como en matrix factorization, minimicen el error a la hora de multiplicarlas con los vectores de features. Entonces buscaremos estas matrices para que a la hora de multiplicarlas por los vectores de features correspondientes de cada uno de nuestros pares de elementos (user-item) se acerquen, en todos los casos, lo más posible a los valores que conocemos.

Eso significa que la fórmula que tenemos que minimizar ahora para que tienda a 0 es:

$$\min_{u,v \in R}(R_{u,v} - f_u^T \cdot P^T \cdot Q \cdot g_v)$$

Donde R es el valor que tenemos para el user u y el item v, f y g son los vectores de features de los items y de los users respectivamente. Y P y Q son las dos matrices que calcularemos a partir de algoritmos de Machine Learning para que la multiplicación de estas dos matrices junto con los dos vectores de features sea lo más cercana posible, el mayor número de veces posible a los datos que tenemos. Es por eso que esta es una evolución del cálculo de Matrix Factorization.

Este algoritmo se compone de una fase de training y una de test. Puede usarse con regresión o con clasificación. Puede ser entrenado con distintos algoritmos, los más usados son: Stochastic Gradient Descent, Alternative Least Square y Markov Chain Monte Carlo.

Este algoritmo mapea las interacciones a un espacio de pocas dimensiones.

En el cálculo del parámetro que multiplicará los dos vectores de features descriptores del user y el ítem será calculado a partir de dos vectores, uno por user y otro por ítem, aprendiendo directamente del vector de features pero con una dimensión definida al principio del proceso de training.

También es interesante FM porque dependiendo de cómo sea la entrada al sistema puede emular a otros state-of-the-art algoritmos. Pues es una generalización de varios modelos de factorización como Matrix Factorization, SVD++...

Este algoritmo permite que los datos de entrada sean diferentes a un único rating por user-item, pero sin esa información explicita contrastable no puede generar los vectores. Esto hace que sea más flexible a la hora de aprender de los datos. Y que tenga más opciones donde estimar repercusiones que con tan solo un dato.

De todas formas parece ser el algoritmo que mejor funciona y que más se adapta a nuestras necesidades. El problema está en ese rating que necesita el algoritmo y que no podemos pedir explícitamente. Por ello vamos a intentar generarlo con los datos que tenemos, para al menos simular ese rating. Después podremos pedir datos explícitos a los usuarios para contrastarlos.

Recomendador

Una vez decidido el tipo de recomendador que vamos a utilizar vamos a decidir con qué tecnologías vamos a trabajar y cómo vamos a generar un ambiente de desarrollo que nos permita trabajar con una gran cantidad de datos y procesarlos de forma adecuada para conseguir recomendaciones.

Para ello vamos a valorar como queremos la salida del recomendador y que entrada podemos tener.

Para la entrada, persistencia y salida de los datos usaremos una base de datos postgresql, ya que el servidor de snowflake trabaja con ellas y esta va a ser la entrada del sistema, e idealmente también la salida. Y, por su facilidad de lectura y al ser este un programa que no debería ejecutarse muchas veces pero sí debería ser fácilmente mejorable, hemos decidido utilizar Python como lenguaje para el recomendador. Así que vamos a utilizar Python para escribir el algoritmo y SQL para guardar los datos.

Para acceder a la base de datos desde Python hemos instalado una librería llamada *psycopg2* que permite trabajar con bases de datos desde este lenguaje. Y para el cálculo de Matrix Factorization hemos utilizado una librería open source en Python desarrollada por *coreylynch* que nos ofrece el cálculo sobre la librería *numpy* y *sklearn*.

La entrada del sistema la vamos a hacer a partir de la tabla de eventos del sistema, después de pre procesarla vamos a generar una tabla SQL llamada user_metrics que generaremos en SQL con el siguiente script.

```
CREATE TABLE user_metrics (sf_user_id varchar(40) NOT NULL,
sf_date date NOT NULL, space_id varchar(40) NOT NULL,
created_flow_items int, read_flow_items int,
read_container_items int, created_container_items int,
updated_container_items int, total_session_time int, PRIMARY
KEY(sf_user_id,sf_date,space_id));
```

Esta tabla expresa, para un día concreto, el número de elementos de cada clase que un user generó en un space. Tiene los siguientes campos

sf_user_id: El id del user. (Primary Key)

space_id: El id del space. (Primary Key)

sf_date: El día sobre el cual hablan los datos. (Primary Key)

created_flow_items: El número de flow items creados por ese user en ese space ese día.

read_flow_items: El número de flow items leídos por ese user en ese space ese día.

read_container_items: El número de container items leídos por ese user en ese space ese día.

created_container_items: El número de container items creados por ese user en ese space ese día.

updated_container_items: El número de container items actualizados por ese user en ese space ese día.

total_session_time: El tiempo de sesión del usuario ese día. Del event_log actual no se puede discriminar el tiempo de sesión por space así que actualmente este valor será igual para todos los spaces para el mismo user.

Con esta entrada del sistema calcularemos el rating implícito de user por space. Para la salida del sistema tendremos la tabla suggested_ratings que se genera con el siguiente script:

```
CREATE TABLE suggested_ratings (sf_user_id varchar(40) NOT NULL,
space_id varchar(40) NOT NULL, rating real, PRIMARY
KEY(sf_user_id, space_id));
```

Esta tabla servirá para consultar por space id y por user id para ver qué valor genera nuestro predicción de puntaje después de pasar por el cálculo de Matrix Factorization. Los campos que tiene son:

sf_user_id: El id del user. (Primary Key)

space_id: El id del space. (Primary Key)

rating: El puntaje inferido a partir del cálculo de Factorization Machine.

Como tabla intermedia para el sistema crearemos la tabla de implicit_ratings, que será la encargada de almacenar que puntaje generó el algoritmo de generación de ratings a partir de la tabla de entrada de sistema. Se genera partiendo del siguiente script:

```
CREATE TABLE implicit_ratings (sf_user_id varchar(40) NOT NULL,
space_id varchar(40) NOT NULL, rating real, PRIMARY
KEY(sf_user_id, space_id));
```

Tiene los mismos campos que la tabla de salida del sistema, solo cambia el modelo conceptual de rating, el cual en vez de tener el puntaje inferido a partir del FM tiene el puntaje inferido a partir del algoritmo de cálculo de rating que implementamos.

sf_user_id: El id del user. (Primary Key)

space_id: El id del space. (Primary Key)

rating: El puntaje inferido a partir del cálculo por el algoritmo SFRater.

Ahora que hemos explicado las tres tablas SQL que utilizará el sistema, vamos a explicar qué proceso tendrá el diseño generado.

Una vez por día se correrá un demonio que generará los siguientes procesos:

1- De la tabla event_log discretizar a partir de los logs del sistema cuantos elementos se leyeron, crearon y actualizaron en el día anterior, por space y user. Y almacenarlos en la tabla user_metrics.

2- A partir de la tabla user_metrics generar un puntaje implícito a partir del uso de los users en los últimos días y en los últimos meses. (SFRater permite que dada una marca de tiempo darle una relevancia distinta a los datos posteriores a esa marca de tiempo de los datos totales). Y guardar ese puntaje en la tabla implicit_ratings.

3- A partir de la tabla implicit_ratings, usando la librería fastFM o el cálculo matrix factorization, generar un puntaje estimado para los spaces para los cuales los users no tengan uso (rating implícito == 0). Y guardar los datos en la tabla de salida del sistema.

A partir de que el proceso acabe tendremos actualizados los datos de recomendación del sistema respecto a los datos de los usuarios del día anterior. De esta manera siempre tendremos datos disponibles, y estos se irán actualizando conforme avance el tiempo y tengamos datos renovados.

SFRater

En este apartado vamos a explicar el funcionamiento del algoritmo de cálculo del rating implícito. Para el cálculo del rating tenemos que considerar los datos que tenemos. Los datos que tenemos son los de creación, lectura y modificación de elementos, así como el tiempo de sesión.

Lo que haremos es para cada user, para cada space y para cada user-space calcular el número de elementos totales y máximos para cada valor. Este cálculo lo haremos para tener distintos tipos de cocientes, uno por user entre el máximo del user, uno por user entre el máximo del space y para cada uno de ellos lo haremos para el total de los valores y para los valores dentro de un rango de tiempo más corto (ajustable).

Esta distinción la hacemos porque consideramos que el interés de un user por un space es algo que cambia a menudo en el tiempo y que hay que tener en cuenta tanto la totalidad del uso de la herramienta como el uso que el user ha tenido últimamente. (Este rango de tiempo y su coeficiente de acción es ajustable también).

Después multiplicaremos el cociente por un coeficiente típico para cada valor, produciendo así un resultado entre 0 y el coeficiente determinado. Y finalmente multiplicaremos el resultado de cada rating por un coeficiente de impacto que cada uno tendrá en el rating final. La suma de todos los resultados producirá un valor, que normalizado, se convertirá en el rating que utilizaremos como rating implícito.

Esta explicación es tan genérica como el código escrito, de forma que si en el futuro tenemos otros datos distintos, o queremos cambiar el coeficiente de alguno de los valores no afecte a la escala del puntaje total.

Ahora comentaremos el código y explicaremos de forma más detallada el cálculo realizado.

El primer paso es calcular el total y el máximo de todos los valores, esto lo hacemos porque para un user que ha enviado 10 flow items un space no significa lo mismo que para un user que haya enviado 1000 flow items en todos sus spaces que para uno que haya enviado 100. Obviamente el cálculo de interés que un user tiene por un space debe hacerse en función de los flow items que haya enviado en ese space, pero también en función de los que haya enviado en total.

Este proceso es el que ejecuta el método clean big data.

La salida de este método serán dos diccionarios:

- spacesData: Es un diccionario que tendrá como llaves los id's de los spaces.
    - Space: Es un diccionario que tendrá como llaves los días que se usó ese space el total que se utilizó el space, el máximo qué se utilizó y el total máximo que se utilizó.
        - Day: En cada uno de estos elementos hay una array, de todos los datatypes que tenemos como entrada del sistema, con la suma de todos los valores de todos los users que utilizaron el space ese día.
        - Max: En este elemento hay una array, de todos los datatypes que tenemos como entrada del sistema, con el máximo que un user usó ese space en un día.
        - Total: En este elemento hay una array, de todos los datatypes que tenemos como entrada del sistema, con el total de uso que se le dio al space en todos los días todos los users.
        - Total_max: En este elemento hay una array, de todos los datatypes que tenemos como entrada del sistema, con el total máximo que un user uso este space.
- usersData: Es un diccionario que tendrá como llaves los id's de los users.
    - User: Es un diccionario que tendrá como llaves los id's de los spaces y el total que el user utilizó el sistema, el máximo qué utilizó un space en un día y el total máximo que utilizó un space en total.
        - Space: Es un diccionario que tendrá como llaves los días que el usuario utilizó el space y el total que lo utilizó.
            - Day: En cada uno de estos elementos hay una array, de todos los datatypes que tenemos como entrada del sistema, con los valores que el user en cuestión utilizó el space en ese día.
            - Total: En este elemento hay una array, de todos los datatypes que tenemos como entrada del sistema, la cual refleja el total que el user utilizó el space en todos los días.

- Total: En este elemento hay una array, de todos los datatypes que tenemos como entrada del sistema, con los valores totales que el user en cuestión utilizó en todos los spaces en todos los días.
- Max: En este elemento hay una array, de todos los datatypes que tenemos como entrada del sistema, con los valores máximos que este user utilizó un space.
- Total_max: En este elemento hay una array, de todos los datatypes que tenemos como entrada del sistema, con los valores máximos que este user utilizó un space en total en todos los días.

Para conseguir estos datos utilizamos el siguiente método, que explicaremos a continuación:

```
def cleanBigData(data):
    spacesData = {};
    usersData = {};
    for oneData in data:
```

Cada oneData es un vector con los datos de la tabla user_metrics.

```
        user = oneData[INDEX_TYPE_USER];
        day = oneData[INDEX_TYPE_DAY];
        space = oneData[INDEX_TYPE_SPACE];
```

Los índices del vector podrían cambiar, así que lo hemos hecho es guardar los índices con una constante.

```
        dayData =
numpy.array(oneData[0:len(DATA_TYPES)],'float');
```

En los primeros valores del vector están los datos (los valores que hemos recopilado del event_log en la tabla user_metrics) así que guardamos en dayData todos los valores.

```
        #Data of Users
        if (usersData.has_key(user)):
            usersData[user][MAX] = getMaxData(usersData[user]
[MAX], dayData);
            usersData[user][TOTAL] = usersData[user][TOTAL] +
dayData;
        else:
            usersData[user] =
{TOTAL:dayData.copy(),MAX:dayData.copy(),TOTAL_MAX:dayData.copy(
)};
```

En esta porción de código guardamos en una posición del vector de user el máximo de cada dato. Guardaremos el máximo de cada valor que cada user ha tenido en uno de sus spaces.

También guardamos el total de los valores que tiene el user para todos los spaces.

```
        userSpaceData = {};
        if (usersData[user].has_key(space)):
            userSpaceData = usersData[user][space];
```

```
            userSpaceData[TOTAL] = userSpaceData[TOTAL] +
dayData;
        else:
            userSpaceData[TOTAL] = dayData.copy();
        usersData[user][TOTAL_MAX] = getMaxData(usersData[user]
[TOTAL_MAX],userSpaceData[TOTAL]);
        userSpaceData[day] = dayData;
        usersData[user][space] = userSpaceData;
```

También guardamos el valor máximo y el total para cada user en cada space.

```
        #Data of Space
        spaceData = {};
        if (spacesData.has_key(space)):
            spaceData = spacesData[space];
            spaceData[MAX] = getMaxData(spaceData[MAX],dayData);
            spaceData[TOTAL] = dayData + spaceData[TOTAL];
        else:
            spaceData[MAX] = dayData.copy();
            spaceData[TOTAL] = dayData.copy();
            spaceData[TOTAL_MAX] = dayData.copy();
        spaceData[day] = dayData;
        spaceData[TOTAL_MAX] =
getMaxData(spaceData[TOTAL_MAX],userSpaceData[TOTAL]);
        spacesData[space] = spaceData;
```

También para todos los spaces guardamos el máximo y el total de cada valor para cada space.

```
    return spacesData, usersData;
```

Todos estos datos los calculamos para que los valores que tengamos a la hora de calcular los ratings sean relativos al total y el máximo de valores por cada space y user.

De esta forma para cada valor (created flow items, read flow items, created container items, updated container items, read container items y total session time) podremos calcular un valor de 0 a 1 que será cuantas de esas operaciones ha realizado el user en cuestión en el space decidido en función de cuantos de estas operaciones haya realizado cómo máximo, en función del user y en función del space.

Teniendo estos valores de 0 a 1 por cada valor de entrada del sistema, los sumaremos multiplicándolos antes por un coeficiente, que deberemos ajustar con los datos reales, para con ellos calcular cuánto interesa a un user un space.

Para que este valor también sea entre 0 y 1 sumaremos todos los coeficientes y dividiremos la suma de los valores que tenemos por el resultado de la suma de los coeficientes para conseguir normalizarlo.

Este cálculo se hará tanto para el total de los días que haya utilizado la herramienta el user como para el rango de tiempo que consideremos tiempo corto (el número de días que tengamos en cuenta para el uso "reciente" de la herramienta).

Generando de esta manera cuatro ratings:

- UserSpaceTotal/UserTotalMax

- UserSpaceTotal/SpaceTotalMax
- UserSpaceRecent/SpaceRecent
- UserSpaceRecent/UserRecent

Después estos cuatro ratings entre 0 y 1 los multiplicaremos por un coeficiente específico para cada tipo de rating, los sumaremos y los dividiremos entre la suma de los coeficientes generando así un solo rating entre 0 y 1.

Finalmente normalizaremos este valor para tener un puntaje entre 0 y 5.

Este proceso se hará en el método getImplicitRatingsParametric. La entrada de este método es:

Los datos generados de limpiar user_metrics en cleanBigData (spacesData y usersData)

recentLastDays: El número de días que tendremos en cuenta para el cálculo reciente.

today: El número de día que es hoy.

alphas: Un array con los coeficientes para cada tipo de rating.

alphasDataTypes: Un array con los coeficientes para cada tipo de dato.

La salida del sistema será:

- ratings: Una array con users en el eje x y spaces en el eje y.
- users: Una array que en cada índice tiene el id del user en cuestión.
- spaces: Una array que en cada índice tiene el id del space en cuestión.
- spaceRatings: Una array que para cada space tiene un array de todos los users y para cada user tiene una array con los valores que salieron de los cuatro tipos de rating que tenemos (debug).

```
def
getImplicitRatingsParametric(spacesData,usersData,recentLastDays
,today,alphas,alphasDataTypes):
    ratings = numpy.zeros([len(usersData),len(spacesData)]);
    spaces = spacesData.keys();
    users = usersData.keys();

    alphaSum = sum(alphas);
    alphaDataTypeSum = sum(alphasDataTypes);
    spaceRatings = [];
```

Inicializamos la salida de los datos y calculamos una sola vez la suma de todos los coeficientes.

```
    for spaceIdx in xrange(len(spaces)):
        space = spaces[spaceIdx];
        recentSpaceData =
getRecentData(spacesData[space],recentLastDays,today);
        userRatings = [];
```

Para cada space recuperamos los datos recientes.

```
        for userIdx in xrange(len(users)):
            user = users[userIdx];
            if (usersData[user].has_key(space)):
                recentUserData = getRecentData(usersData[user]
[space],recentLastDays,today);
```

Si el user ha utilizado el space recuperamos los datos recientes para ese user en ese space.

```
                rating = numpy.zeros(len(RATING_TYPES));
                ratingTotal = 0;
                for ratingType in RATING_TYPES:
```

Para cada tipo de rating hacemos el cálculo específico y lo guardamos en la array rating.

```
                    if (ratingType == RATING_TYPE_USER):
                        userRatingTotal = sum((usersData[user]
[space][TOTAL]/usersData[user][TOTAL_MAX])*alphasDataTypes);
                        rating[ratingType] =
userRatingTotal/alphaDataTypeSum;

                    elif (ratingType == RATING_TYPE_SPACE):
                        spaceRatingTotal = sum((usersData[user]
[space][TOTAL]/spacesData[space][TOTAL_MAX])*alphasDataTypes);
                        rating[ratingType] =
spaceRatingTotal/alphaDataTypeSum;

                    elif (ratingType ==
RATING_TYPE_RECENT_USER):
                        userRecentRatingTotal =
sum(recentUserData/recentSpaceData*alphasDataTypes);
                        rating[ratingType] =
userRecentRatingTotal/alphaDataTypeSum;

                    elif (ratingType ==
RATING_TYPE_RECENT_SPACE):
                        spaceRecentRatingTotal =
sum(recentSpaceData/spacesData[space][TOTAL]*alphasDataTypes);
                        rating[ratingType] =
spaceRecentRatingTotal/alphaDataTypeSum;

                    ratingTotal = ratingTotal +
rating[ratingType] * alphas[ratingType];

                userRatings = userRatings + [rating];
                ratingTotal = ratingTotal / alphaSum;
```

Una vez hemos calculado todos los tipos de rating los sumamos multiplicados por el coeficiente específico y los dividimos entre la suma de los coeficientes.

```
                ratingTotal = ratingTotal * MAX_RATING;
                ratings[userIdx,spaceIdx] = ratingTotal;
```

Después normalizamos el rating a la escala deseada y lo guardamos en la variable de salida rating en el índice apropiado.

```
    else:
```

```
            userRatings = userRatings +
[numpy.zeros(len(RATING_TYPES))];
```
Si el user no ha interactuado en el space guardamos una array de 0 en la variable de debug

```
        spaceRatings = spaceRatings + [userRatings];
```
Guardamos en la variable de debug los ratings de todos los users en el space.

```
    return ratings,users,spaces,spaceRatings;
```

Una vez hemos calculado el rating implícito para todos los users para todos los spaces lo guardaremos en la tabla intermedia del sistema, implicit_ratings.

Matrix Factorization

Para la predicción de los ratings también hemos utilizado el cálculo de mátrix factorization en la matriz de ratings. Esto genera unos valores de predicción para las parejas user-space para las cuales no tenemos datos.

A continuación el algoritmo de matrix factorization extraído de un tutorial para la implementación de este algoritmo en python.

```
def matrixFactorization(R, P, Q, K, steps=500, alpha=0.0002,
beta=0.02):
    for step in xrange(steps):
```
Haremos este calculo el numero de steps que indique el que llame a la función.

```
        for i in xrange(len(R)):
            for j in xrange(len(R[i])):
```
Para todos los elementos de R si es más grande que 0 significa que tenemos datos, entonces nos fijamos en el error que tenemos al multiplicar P y Q.

```
                if R[i][j] > 0:
                    eij = R[i][j] - numpy.dot(P[i,:],Q[:,j]);
```
Para todos las features descriptoras del vector (K) modificamos los elementos descriptores del user y del space con las siguientes formulas.

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$
$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij} p_{ik}$$

```
                    for k in xrange(K):
                        P[i][k] = P[i][k] + alpha * (2 * eij *
Q[k][j] - beta * P[i][k]);
                        Q[k][j] = Q[k][j] + alpha * (2 * eij *
P[i][k] - beta * Q[k][j]);
        eR = numpy.dot(P,Q);
        e = 0;
```

Después de actualizar las matrices P y Q nos fijamos en que error tiene. Si el error es menor de 0.001 dejamos de actualizar las matrices.

Para el calculo del error utilizamos esta fórmula:

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

```
        for i in xrange(len(R)):
            for j in xrange(len(R[i])):
                if R[i][j] > 0:
                    e = e + pow(R[i][j] -
numpy.dot(P[i,:],Q[:,j]), 2);
                    for k in xrange(K):
                        e = e + (beta/2) * ( pow(P[i][k],2) +
pow(Q[k][j],2) );
        if e < 0.001:
            break
    return P, Q
```

Para utilizarlo tenemos la siguiente función, que dados los ratings implícitos genera datos para los que tienen rating 0.

```
def suggestRatings(ratings):
    N = len(ratings);
    M = len(ratings[0]);
    K = MATRIX_FACTORIZATION_K;

    P = numpy.random.rand(N,K);
    Q = numpy.random.rand(K,M);
    nP, nQ = matrixFactorization(ratings, P, Q, K);
    return numpy.dot(nP,nQ);
```

N y M son el numero de users y space por user, K es el tamaño de los vectores descriptores de los users y los items. Y P y Q son dos matrices, que generan un vector por user y por space, descriptores de los mismos.

La idea es que cuando multipliques P y Q encontremos R siendo R los ratings con todos los valores. Esto realmente lo que hace es modificar P y Q para que su producto se acerque lo máximo posible a los valores que tenemos en la matriz. Y de esa forma ajustar las matrices a partir de el calculo explicado en el proceso anterior.

Una vez tenemos los valores predecidos de los ratings los guardamos en la tabla suggested_ratings. Para que tan solo una consulta a esta tabla, dado un spaceid o un userid podamos tener los ratings sugeridos de los mismos.

FastFM

Esta librería ofrece unas primitivas en Python para hacer el cálculo de Factorization Machine. En el apartado de resultados veremos una comparación de distintas librerías que implementan este algoritmo a partir de la librería libFM. Dada esa evaluación hemos decidido utilizar esta implementacion en python.

Para la utilización de esta librería tendremos dos parámetros de entrada. Uno será una array de vectores de features que describen la relación user-space. Y el otro un vector con el valor de rating que tenemos generado a partir de SFRater.

Esta forma de utilizar la librería no es la más indicada, pero como de momento no tenemos un valor explícito (dado por el usuario) que describa la relación user-space nos conformaremos con ese valor generado como objetivo del recomendador.

Como vector descriptor de la relación user-space tenemos distintas posibilidades para generar los resultados.

- Podemos tan solo usar los id's de los elementos.
- Podemos usar los datos de entrada que recopilamos en user_metrics.
- Podemos usar un vector con los cuatro tipos de ratings que generamos.
- Podemos usar también un vector que conjunte algunos datos recopilados de user_metrics, pero sin ser del todo brutos, haciéndole un pre-procesamiento previo.

La librería es opensource, y tiene documentación online que ayuda a su utilización. Tiene una parte escrita en python y la parte crítica en C, lo que la hace más estable que pylibFM.

Finalmente para la predicción, tendremos que darle al algoritmo un array con vectores descriptores de relaciones user-space para que el algoritmo haga una regresión y prediga los valores que generarían los vectores descriptores.

Necesitamos como entrada las siguientes variables:

x_train: csc_sparsity matrix. Representa el vector de features. En las columnas del mismo tiene que estar representado de que user y space estamos hablando.

y_train: Vector. Representa el valor que tienen cada uno de los vectores de features en la matriz x_train. En nuestro caso es el rating implícito, pero podría también ser un valor categórico.

x_test: csc_sparsity matrix. Representa el vector de features con las relaciones de las cuales queremos saber el correspondiente al vector y_train.

Y Devuelve:

y_test: Vector. Representa el valor predecido para cada uno de los vectores de features en la matriz x_test. En nuestro caso es el rating sugerido.

Para la utilización de esta librería tenemos que darle como entrada una matriz encodeada como csc_sparsity matrix. La cual se genera con 3 vectores, uno de datos y otros dos de filas y columnas. Por lo tanto tenemos que procesar nuestros datos para que la entrada sea la deseada.

Para ello escribimos una función que lo hace. Tiene dos funciones utilitarias que son las siguientes para crear vectores típicos necesarios.

```
def vectorWithLengthAndData(length,data):
     result = [];
     for i in range(length):
      result = result + [data];
     return result;
def vectorWithLengthFromNumber(length,number):
     result = [];
     for i in range(length):
      result = result + [number + i];
     return result;
def isVector0(vector):
     for x in vector:
      if x != 0:
           return false;
     return true;
```

La función que nos devuelve las entradas necesarias para el cálculo con FM es la siguiente. Nos devuelve como vector de features los distintos tipos de ratings generados, es decir los debug_ratings.

```
def getFastFMEntryFromUsersSpacesRatingsAndDebugRatings(users,
spaces,ratings,debugRatings):
     rows = [];
     cols = [];
     data = [];
     rowsTest = [];
     colsTest = [];
     dataTest = [];

     m = len(spaces);
     n = len(users);
     l = len(debugRatings[0][0]);
```

La nueva matriz x_train será una matriz de (numero de datos) * (numero de features) = (numero de parejas user-space con datos (rating implícito)) * (m+n+l (una columna por user, space y feature)).

```
     featuresIdxs = vectorWithLengthFromNumber(l,m+n);
     rowIdx = 0;
     rowTestIdx = 0;
     y_train = [];
     for spaceIdx in xrange(m):
      space = spaces[spaceIdx];
      for userIdx in xrange(n):
           user = users[userIdx];
           features = debugRatings[spaceIdx][userIdx];
```

Para todos los usuarios y todos los spaces nos fijamos si tiene rating implícito.

```
          if (ratings[userIdx,spaceIdx] == 0):
               rowsTest = rowsTest + [rowTestIdx, rowTestIdx];
               colsTest = colsTest + [spaceIdx, m + userIdx];
               dataTest = dataTest + [1,1];
               rowTestIdx += 1;
```

Si no tiene rating lo guardamos en los vectores de test.

```
          else:
               rows = rows + [rowIdx, rowIdx] +
vectorWithLengthAndData(l,rowIdx);
               cols = cols + [spaceIdx, m + userIdx] +
featuresIdxs;
               data = data + [1,1] + features.tolist();
               y_train = y_train + [ratings[userIdx,spaceIdx]];
```

Si tiene rating lo guardamos en los vectores de training. Para saber como guardarlo, ponemos dos unos en la columna respectiva al space y al user. Así como los valores de las features en los indices de las features.

También guardamos el rating de la pareja space-user en la variable y_train.

```
               rowIdx += 1;
     rows = numpy.array(rows);
     cols = numpy.array(cols);
     data = numpy.array(data);
     result = csc_matrix((data, (rows, cols)), shape=(rowIdx,
m+n+l))
     rows = numpy.array(rowsTest);
     cols = numpy.array(colsTest);
     data = numpy.array(dataTest);
     resultTest = csc_matrix((data, (rows, cols)),
shape=(rowTestIdx, m+n+l));
```

Finalmente pasamos nuestros vectores de filas, columnas y datos a las variables matrices deseadas y las devolvemos, así como y_train.

```
     return result, resultTest, numpy.array(y_train);
```

Después de esto tan solo tenemos que inicializar la Factorization Machine y pedirle que prediga con los datos previamente preparados. Para ello escribimos la función testFM.

```
def testFM():
     data = getDataFromServer();
     spacesData, usersData = cleanBigData(data);
     ratings,users,spaces,debugRatings =
getImplicitRatings(spacesData,usersData);
     x_train, x_test, y_train =
getFastFMEntryFromUsersSpacesRatingsAndDebugRatings(users,spaces
,ratings,debugRatings);
```

En estas primeras líneas recuperamos los datos del server, los limpiamos, calculamos los ratings implícitos y ejecutamos la función descrita anteriormente para tener las entradas necesarias para la FM. Utilizaremos el módulo *Markov Chain Monte Carlo* (MCMC) .

```
        fm = mcmc.FMRegression(n_iter=1000,init_stdev=0.1, rank=MAX_RATING,
random_state=123);
        y_test = fm.fit_predict(x_train, y_train, x_test, n_more_iter=0);

        m = len(spaces);
        n = len(users);
        suggestedRatings = numpy.zeros([n,m]);
        rowIdx = 0;
        for spaceIdx in xrange(m):
         for userIdx in xrange(n):
                if (ratings[userIdx,spaceIdx] == 0):
                        suggestedRatings[userIdx,spaceIdx] = y_test[rowIdx]*MAX_RATING;
                        rowIdx += 1;
```

Después de predecir los resultados los pasamos a una matriz de numpy y los guardamos en el server (después de borrar los datos anteriores).

```
      dataBase.dropSuggestedRatingsTable();
      dataBase.createSuggestedRatingsTable();

      saveSuggestedRatingsToServer(ratings, suggestedRatings,
users, spaces);
      return suggestedRatings;
```

Entonces finalmente tendremos los ratings sugeridos predecidos por Factorization Machine guardados en las tablas deicididas. Tenemos que mencionar que este método es bastante más eficiente que el cálculo de matrix factorization.

Database

Para la interacción con la base de datos hemos generado un módulo en python con distintas funciones. Permitiéndonos pasar de nuestros datos a la base de datos.
Tiene funciones para generar y borrar las tablas necesarias así como para recuperar y guardar los datos guardados en ellas. Para ello utilizamos una librería en python llamada *psycopg2*. La cual nos ofrece una api para trabajar con bases de datos postgresql.
Hemos decidido utilizar una base de datos postgresql porque así será más fácil de integrar el módulo con el resto del servidor. De esa forma cualquier proceso puede consultar los datos que necesite.
También tenemos 2 métodos para importar y exportar datos desde un fichero csv a la tabla deseada. Esto nos permite guardar datos en disco.

Resultados

En esta sección vamos a mostrar y comentar los resultados obtenidos

Matrix Factorization

Los primeros resultados que vamos a comentar son los de el algoritmo de Matrix Factorization con datos de movielens. Para esta prueba dividimos los datos en una parte de training y una de test.

Vamos a evaluar que error tenemos a la hora de predecir los ratings con Matrix Factorization y con Factorization Machine, para este último vamos a basarnos en los datos recopilados en internet y a comentar como son los distintos resultados dependiendo de la entrada que tenga el algoritmo.

El ejercicio que hicimos para probar el algoritmo de matrix factorization con los datos de los ratings produjo estos resultados: Probamos con 9.912 ratings. Estos datos dicen por cuanto erraron nuestras predicciones. Sobre 5 puntos cuantos puntos se equivocaron.

El 75% de los ratings predecidos diviergen del real por menos de 1/5. Después tenemos un 3% que diviergen por más de 2/5 y un 23% que diviergen entre 1 y 2 puntos sobre 5. Lo cual es una predicción bastante buena.

% error	% de peliculas
> 40%	3.64%
20% > 40%	22.96%
10% > 20%	31.15%
4% > 10%	24.92%
< 4%	17.31%

Factorization Machine

Para la prueba con Factorization Machine voy a evaluar sobre la librería libfm con los datos de movielens, en distintas implementaciones y los resultados que tuvo respecto a la cantidad de features introducidas.

En un dataset de 100k con tan solo los ids las distintas implementaciones dieron los siguientes resultados.

	Time	RMSE
LibFM	8.970900	0.913520
FastFM	4.84002	0.915184
PylibFM	13.157164	0.944870

En un dataset de 100k con informacion adicional las distintas implementaciones dieron los siguientes resultados.

	Time	RMSE
LibFM	49.632649	0.896349
FastFM	53.611804	0.896543
PylibFM	55.756278	NaN

En un dataset de 1m con tan solo los ids las distintas implementaciones dieron los siguientes resultados.

	Time	RMSE
LibFM	275.672684	0.861539
FastFM	307.400295	0.858305
PylibFM	132.618739	0.870263

En un dataset de 1m con informacion adicional las distintas implementaciones dieron los siguientes resultados.

	Time	RMSE
LibFM	779.900802	0.850382
FastFM	1170.46813	0.852738
PylibFM	564.922632	NaN

Podemos ver que pylibfm falla con grandes cantidades de información y que libFM y fastFM tienen resultados bastante parecidos, siendo libFM la librería que tiene mejores resultados.

SFRater con datos aleatorios

Ahora comentaremos los resultados que tuvimos al utilizar datos aleatorios como entrada del recomendador para simular los datos reales.

usermetrics

sf_user_id	sf_date	space_id	created_flow_items	read_flow_items	read_container_items
user 0	2015-01-01	space 0	178	380	8
user 0	2015-01-01	space 2	10	118	1
user 0	2015-01-01	space 3	140	130	28
user 0	2015-01-01	space 5	85	485	43
user 0	2015-01-01	space 6	58	93	7
user 0	2015-01-01	space 8	127	316	24
user 0	2015-01-01	space 9	40	38	44
user 0	2015-01-01	space 14	46	54	42
user 0	2015-01-01	space 16	48	216	38
user 0	2015-01-01	space 17	42	305	2
user 0	2015-01-01	space 18	137	81	3
user 0	2016-01-01	space 19	186	54	8
user 1	2015-01-01	space 0	115	480	35
user 1	2016-01-01	space 1	120	175	31
user 1	2016-01-01	space 3	26	186	28
user 1	2016-01-01	space 4	62	348	11
user 1	2016-01-01	space 5	170	229	29
user 1	2016-01-01	space 8	171	0	17
user 1	2016-01-01	space 7	36	186	24
user 1	2015-01-01	space 8	125	294	28
user 1	2016-01-01	space 12	186	323	17
user 1	2015-01-01	space 14	151	378	41
user 1	2015-01-01	space 18	18	499	38
user 1	2015-01-01	space 19	42	283	23
user 2	2015-01-01	space 3	118	115	18
user 2	2015-01-01	space 4	18	318	22
user 2	2015-01-01	space 8	102	273	1
user 2	2015-01-01	space 7	50	452	40
user 2	2015-01-01	space 8	187	486	48
user 2	2015-01-01	space 9	2	208	14
user 2	2015-01-01	space 10	48	487	10
user 2	2015-01-01	space 14	38	211	32
user 2	2015-01-01	space 15	136	282	41
user 2	2016-01-01	space 18	53	450	47
user 2	2015-01-01	space 17	55	52	6
user 2	2016-01-01	space 19	147	279	31
user 3	2016-01-01	space 1	184	294	8
user 3	2016-01-01	space 7	105	384	49
user 3	2016-01-01	space 13	98	15	27

usermetrics

created_container_items	updated_container_items	total_session_time
11	21	21528
14	23	21528
9	22	21528
9	2	21528
1	13	21528
8	18	21888
11	11	21528
5	4	21888
8	8	21888
8	20	21888
4	8	21888
11	9	21528
0	22	181.57
9	8	181.57
18	21	181.57
4	5	181.57
1	13	181.57
17	3	181.57
0	21	181.57
2	20	181.57
3	13	181.57
15	1	181.57
19	15	181.57
17	11	181.57
8	8	10210
4	10	10210
8	28	10210
18	7	10210
0	2	10210
18	0	10210
1	15	10210
10	23	10210
2	18	10210
4	4	10210
11	28	10210
18	3	10210
18	8	19078
15	8	19078
15	5	19078

Esta es la tabla de entrada, tan solo para los primeros users el primer día (es una tabla extensa y no podía ponerla completa).

En esta tabla podemos ver la entrada de nuestro sistema en la tabla user_metrics.

Matrix Factorization

En las siguientes tablas veremos los ratings implícitos para algunos users y spaces, así como los ratings sugeridos por el calculo de matrix factorization.

implicitratings

user 1	space 5	3.94491
user 1	space 4	3.09536
user 1	space 7	2.89776
user 1	space 6	3.61903
user 1	space 1	3.31309
user 1	space 0	3.77499
user 1	space 3	1.74404
user 1	space 2	3.51693
user 1	space 9	0
user 1	space 8	3.3085
user 1	space 11	1.91398
user 1	space 10	0
user 1	space 13	3.47151
user 1	space 12	2.94931
user 1	space 15	0
user 1	space 14	3.51726
user 1	space 17	0
user 1	space 16	3.43318
user 1	space 19	3.0577
user 1	space 18	0

user 0	space 5	3.20425
user 0	space 4	0
user 0	space 7	0
user 0	space 6	2.65241
user 0	space 1	3.50985
user 0	space 0	2.02631
user 0	space 3	2.39262
user 0	space 2	1.81251
user 0	space 9	3.0837
user 0	space 8	2.82612
user 0	space 11	2.70958
user 0	space 10	0
user 0	space 13	1.78919
user 0	space 12	0
user 0	space 15	0
user 0	space 14	1.30568
user 0	space 17	2.16496
user 0	space 16	3.10768
user 0	space 19	2.91002
user 0	space 18	2.47342

Como podemos ver aquí en la tabla suggested ratings tan solo tenemos valores para los elementos que tienen valor 0 en la tabla de implícit ratings.

suggestedratings

user 1	space 9	3.31321
user 1	space 10	2.75934
user 1	space 15	3.13662
user 1	space 17	2.67198
user 1	space 18	3.13561
user 0	space 4	2.65100
user 0	space 7	2.43325
user 0	space 10	2.46681
user 0	space 12	2.36322
user 0	space 15	2.47428

Ahora vamos a evaluar y comentar el proceso de generación del rating implícito.

Vamos a ver los datos de el user 0 en el space 1 y en el space 14.

			usermetrics					usermetrics	
user 0	2015-01-02	space 1	180	167	24	8	16	4979	
user 0	2015-01-03	space 1	157	73	24	1	1	10219	
user 0	2015-01-04	space 1	187	413	2	5	17	15888	
user 0	2015-01-05	space 1	14	423	48	15	22	968	
user 0	2015-01-07	space 1	22	352	27	8	21	14580	
user 0	2015-01-08	space 1	112	48	45	2	19	27207	
user 0	2015-01-09	space 1	130	87	28	5	18	26832	
user 0	2015-01-11	space 1	17	291	12	15	29	1888	
user 0	2015-01-12	space 1	27	228	41	17	13	18264	
user 0	2015-01-01	space 14	46	54	42	5	4	21888	
user 0	2015-01-02	space 14	1	183	34	2	12	4979	
user 0	2015-01-03	space 14	103	70	40	18	6	10219	
user 0	2015-01-04	space 14	28	8	3	18	23	15888	
user 0	2015-01-07	space 14	2	438	3	4	17	14580	

Para el user 0 y space 1 tenemos el rating 3.50985.

Para el user 0 y space 14 tenemos el rating 1.30568.

En este caso nuestros datos de prueba son durante 12 días. Y el intervalo próximo es de 3 días. Por lo tanto el user 0, para nuestros datos, no utilizó últimamente el space 14. De ahí viene la diferencia en el rating generado. También podemos ver que el hecho de que hayan menos días en el space 14 influirá en el puntaje.

Para el user 0 y el space 1 los debug ratings nos muestran:

[0.88570295, 0.76819834, 0.67880633, 0.05305433]

Para el user 0 y el space 14 los debug ratings nos muestran:

[0.40210955, 0.34919627, 0. , 0.05379842]

Los valores de los ratings son sobre 1, y tienen el siguiente significado.

[userSpaceTotal/userTotalMax,

userSpaceTotal/spaceTotalMax,

recentUserData/recentSpaceData,

recentSpaceData/spaceTotal]

Donde **userSpaceTotal** son todos los valores de entrada que el user ha tenido en el space en total. Es decir lo que el user ha interactuado en el space desde el principio del mismo.

UserTotalMax es el máximo que este user ha interactuado en un space, es decir el userSpaceTotal máximo de este user.

SpaceTotalMax es el máximo que un user ha interactuado en un space, es decir el userSpaceTotal máximo de este space.

RecentUserData son todos los valores de entrada que el user ha tenido en ese space recientemente. Es decir lo que el user ha interactuado en el space últimamente.

RecentSpaceData es el total de lo que todos los users han usado el space últimamente.

SpaceTotal es el total que todos los users han usado el space desde el principio del mismo.

Entonces estos cuatro valores significan:

[

Lo que el usuario ha utilizado el space respecto a su uso general de spaces,

Lo que el usuario ha utilizado el space respecto a lo que se ha utilizado el space,

Lo que el usuario ha usado últimamente el space respecto a lo que se ha utilizado recientemente,

Lo que el space se ha utilizado últimamente respecto a lo que se ha utilizado en total el space

]

Para el space 1 entonces tenemos valores bastante grandes (mayores de 0.5 en los tres primeros valores. El ultimo valor es un descriptor del space en realidad. Por eso los dos tienen el mismo valor.

Viendo el primer y segundo valor, que son los que podemos comparar porque el tercer valor es 0 en el segundo caso porque no se utilizó ultimamente.

Vemos que el user 0 ha utilizado proporcionalmente a si mismo y respecto a los otros users del space bastante más el space 1 que el 14, lo que termina produciendo esa gran diferencia a la hora de la puntuación.

Vemos que el hecho de tener un valor para el uso del space respecto al uso total lo que va a hacer es darle un valor añadido a los spaces recientes, para que estos tengan mayor valor para los users.

Entonces aplicandole el cálculo de Matrix Factorization podemos predecir los ratings para las parejas space-user que no tengan valores. De esta manera, podemos usar los ratings generados a partir del uso de la herramienta para definir cuanto un space interesa a un user.

Después de analizar estos datos viendo que la fecha influye de forma importante en la generación del rating. Parece interesante, para hacer un buen benchmarking tanto del calculo de ratings y la predicción de ratings, deberíamos añadirle a las tablas de ratings la fecha.

Factorization Machine

De esa forma podemos generar un rating por día y guardar los datos para poder hacer una gráfica y compararlos a posteriori.

A la hora de utilizar el algoritmo de Factorization Machine es importante tener en cuenta la entrada que debemos darle al mismo.

Por un lado debemos darle una array de vectores de features, que describan la relación user-space. Después para cada una de ellas un valor de rating. Y finalmente un array de vectores de features de la relación user-space sobre las cuales queramos predecir su rating.

El problema que tenemos es que para las parejas user-space para los cuales queremos predecir su rating no tenemos datos de entrada, pues el user no ha utilizado el space, sino no tendria sentido recomendarselo. Por lo tanto el vector de features de la entrada de test estará vacío.

Usando el algoritmo Factorization Machine con los debugRatings como vector de features y el rating generado de forma implícita como valor de training. En modo regresivo hemos obtenido estos resultados:

sf_user_id	space_id	rating
user 9	space 3	1.93448
user 9	space 2	2.09013
user 9	space 12	2.26600
user 9	space 15	1.79119
user 9	space 14	1.93042
user 9	space 16	2.12317
user 8	space 4	1.92928
user 8	space 1	1.77692
user 8	space 3	1.84172
user 8	space 10	1.79748
user 8	space 17	1.98394
user 8	space 19	2.35634
user 7	space 6	1.70918
user 7	space 3	1.66086
user 7	space 2	1.79286
user 7	space 11	1.68745
user 7	space 17	1.73882
user 7	space 19	2.14725
user 6	space 0	1.63899
user 6	space 10	1.76466
user 6	space 14	1.7908
user 6	space 19	2.32242
user 5	space 0	1.73337
user 5	space 8	1.8528
user 5	space 10	1.71927
user 5	space 13	2.01052
user 5	space 15	1.6877
user 5	space 19	2.31085
user 4	space 5	2.0873
user 4	space 4	2.23012
user 4	space 1	2.00518
user 4	space 0	1.95027
user 4	space 2	2.17831
user 4	space 9	2.41155
user 4	space 8	2.11774
user 4	space 19	2.54889
user 3	space 5	2.27607
user 3	space 4	2.38396
user 3	space 6	2.31873
user 3	space 3	2.22065
user 3	space 2	2.38278
user 3	space 9	2.57767
user 3	space 11	2.2681
user 3	space 12	2.56577
user 3	space 17	2.35006
user 2	space 5	1.7924
user 2	space 1	1.68586
user 2	space 0	1.63395
user 2	space 11	1.81294
user 2	space 12	2.03749
user 2	space 18	1.9946
user 1	space 9	2.00623
user 1	space 10	1.6567
user 1	space 15	1.55384
user 1	space 17	1.81268
user 1	space 18	1.92554
user 0	space 4	1.39215
user 0	space 7	1.31229
user 0	space 10	1.21422
user 0	space 12	1.49761
user 0	space 15	1.11983

Comparandolos con los resultados obtenidos en el calculo de Matrix Factorization podemos ver que distan bastante, tampoco parece que sean proporcionales.

Podemos decir que el algoritmo de Factorization Machine és más eficiente que el de Matrix Factorization.

También podemos decir que FM tiene mejores resultados en otros contextos, por lo que podríamos esperar tener mejores resultados en nuestro contexto.

Para ver que tal funciona la predicción con estos datos hemos hecho el cálculo de RMSE con los valores aleatorios para 30 días 60 users y 100 spaces.

Los resultados que hemos obtenido con los distintos módulos son:

Módulo	Time	RMSE
Rank (bpr)	7.4522	1.3578
SGD	3.2375	0.6200
MCMC	5.3520	0.0071
ALS	5.0807	0.0068
MF	54.3339	0.7336

Viendo estos resultados tan solo nos faltaría probar con datos reales generados en la plataforma y ver si podemos de alguna manera hacer el benchmarking de forma explícita con los usuarios.

SFRater con datos reales

Hemos conseguido probar nuestro algoritmo con datos reales, para ello hemos evaluado el período alpha y beta desde el 21 de junio de 2016 hasta el 21 de agosto de 2016.
En este período hay un total de 60 users y otros 600 spaces. Por lo tanto nuestra matriz de parejas space-user es de 36.000 espacios. De los cuales solo tenemos datos para rellenar 1.000.
Nuestros datos son proveídos por OrangeLoops, para rellenar la tabla user_metrics. Debido a la temprana fase de desarrollo y uso de la misma, al hacer el primer run del script que extrae los datos para rellenar la tabla user_metrics, el total_session_time no ha sido calculado, porque actualmente, a parte de ser un valor difícil de calcular, se están sobreponiendo sesiones así que rara vez se acaba una sesión, tan solo con el logout explícito.
Por ello hemos tenido que ajustar el código para que las variables de entrada fueran menos, por suerte ya escribimos el código de forma que tan solo comentando un par de líneas fue suficiente para acogernos a los nuevos datos.
De todas maneras el valor lo puse a 0 en la tabla user_metrics para no cambiar las tablas postgresql. Aunque no lo utilizemos a la hora de generar los ratings.

user_id	date	space_id	created_flow_items	flow_items_read	created_container_items	modified_container_items
23N2XV9ISYB63sU9pW-JlQ	2018-08-21	15sa8B8SRX-MzUlFczKA2A	4	18	0	0
gTccXtgSR9CBFKb5kE9IHA	2018-08-21	15sa8B8SRX-MzUlFczKA2A	8	17	0	0
JkluCULOSySZDCtVeuIyWA	2018-08-21	15sa8B8SRX-MzUlFczKA2A	10	19	0	0
23N2XV9ISYB63sU9pW-JlQ	2018-08-21	1X7HClXET3e8tRR5CVk3hA	4	2	2	0
72VCdf2XTQqGr4oZHLLn2A	2018-08-21	29954d30-37a9-11e8-a531-7fdd4b17202b	0	0	2	0
gTccXtgSR9CBFKb5kE9IHA	2016-06-21	3e6644e0-3771-11e6-bc65-d302566e4666	0	0	0	0
JkluCULOSySZDCtVeuIyWA	2018-08-21	493bb9c0-378c-11e8-bc85-d302888e4888	0	0	1	1
3lAw8ZdGRuWZ6kqK-Lm-lQ	2016-06-21	6d6e34b0-37ee-11e6-8f24-519d3d62535e	0	0	6	2
3xOx6xCcR7DfvR3Bocc8fQ	2016-06-21	6d6e34b0-37ee-11e6-8f24-519d3d62535e	0	0	0	0
72VCdf2XTQqGr4oZHLLn2A	2016-06-21	6d6e34b0-37ee-11e6-8f24-519d3d62535e	0	0	0	0
JkluCULOSySZDCtVeuIyWA	2016-06-21	6d6e34b0-37ee-11e6-8f24-519d3d62535e	0	0	0	0
p4Ygo-I4SLWNjB0XMXqDJQ	2018-08-21	8d8e34b0-37ee-11e8-8f24-519d3d82535e	0	0	0	0
Smpz6NaFT321LoGJIC5RXQ	2016-06-21	6d6e34b0-37ee-11e6-8f24-519d3d62535e	0	0	0	0
23N2XV9ISYB63sU9pW-JlQ	2018-08-21	712d9f90-37c1-11e8-8dcd-d79c0180e0ea	0	0	0	0
3xOx6xCcR7DfvR3Bocc9fQ	2018-08-21	712d9f90-37c1-11e8-8dcd-d79c0180e0ea	0	0	0	0
72VCdf2XTQqGr4oZHLLn2A	2018-08-21	712d9f90-37c1-11e8-8dcd-d79c0180e0ea	0	0	0	0
gTccXtgSR9CBFKb5kE9IHA	2018-08-21	712d9f90-37c1-11e8-8dcd-d79c0180e0ea	0	0	0	0
JkluCULOSySZDCtVeuIyWA	2018-08-21	712d9f90-37c1-11e8-8dcd-d79c0180e0ea	0	0	3	0
Smpz8NaFT321LoGJIC5RXQ	2018-08-21	712d9f90-37c1-11e8-8dcd-d79c0180e0ea	0	0	0	0
72VCdf2XTQqGr4oZHLLn2A	2018-08-21	8olqRXQKSU2F4hkwI7bAA	2	2	0	0
23N2XV9ISYB63sU9pW-JlQ	2018-08-21	941f4160-3770-11e8-bc85-d302888e4888	0	0	0	0
3xOx6xCcR7OfvR3Bocc9fQ	2018-08-21	941f4160-3770-11e8-bc85-d302888e4888	0	0	1	1
72VCdf2XTQqGr4oZHLLn2A	2018-08-21	941f4160-3770-11e8-bc85-d302888e4888	0	0	0	0
gTccXtgSR9CBFKb5kE9IHA	2018-08-21	941f4160-3770-11e8-bc85-d302888e4888	0	0	0	0
JkluCULOSySZDCtVeuIyWA	2018-08-21	AACg8MhQz-OV8zuODJjbWg	0	0	1	0
a14mmk8ATLKJ_uoMtymW	2018-08-21	b53c0320-3771-11e8-bc85-d302888e4888	0	0	0	0
3lAw8ZdGRuWZ8kqK-Lm-lQ	2018-08-21	b8d34a20-37b2-11e8-8f24-519d3d82535e	0	0	0	0
23N2XV9ISYB63sU9pW-JlQ	2018-08-21	CXA87O0hRJ82UNKSyqmYKQ	0	10	0	0
3lAw8ZdGRuWZ6kqK-Lm-lQ	2016-06-21	CXA67O0hRj62UNKSyqmYKQ	6	0	0	0
3xOx6xCcR7OfvR3Bocc9fQ	2018-08-21	CXA87O0hRJ82UNKSyqmYKQ	0	10	0	0
72VCdf2XTQqGr4oZHLLn2A	2016-06-21	CXA67O0hRj62UNKSyqmYKQ	0	10	0	0

Con estos datos hemos generado ratings implícitos. Nuestra tabla de ratings implícitos tiene ahora 36000 datos, 35000 de los cuales son 0. (lo que significa una gran escasez de datos). Ningún rating llegó a más de 3 y la mayoría 80% están por debajo del 1. Esto significa que deberemos ajustar los coeficientes para que los ratings sean más dispersos.

3lAw9ZdGRuWZ6kqK-Lm-lQ	DK2w_JSj1JMLThB0l0ePiK	1.39573
3lAw9ZdGRuWZ6kqK-Lm-lQ	JHri9yk7Szy-H7hSLtQpMQ	1.39291
JkkuCUL0SySZDCtVeuiyWA	JkkuCUL0SySZDCtVeuiyWAoO-vJeahTwmrEB2VegVnNw	1.39165
MZ1Y_gjASfOBdJjR9kl5Yw	JkkuCUL0SySZDCtVeuiyWAMZ1Y_gjASfOBdJjR9kl5Yw	1.37778
p4Ygo-l4SLWNjBtXMXqOJQ	DfoxYX9fpBHlw-6-VtAVUs	1.37749
gTccXtgSR9CBFKb5kE9iHA	w2sAppoYQ3KCnJEdyWMw7A	1.36906
3xDxBxCcR7OfvR3Boce9fQ	am9TJ001SMq8Bu7lD8MKcg	1.36287
gTccXtgSR9CBFKb5kE9iHA	dHoE4nfUQnyYbGCqx7r9xg	1.35902
72VCdt2XTQqGr4oZHLLn2A	CKDPCgtk1H3rS-mCJ4U676	1.35356
3lAw9ZdGRuWZ6kqK-Lm-lQ	BkNwdpXKpB3pBdFiF3lwaA	1.35244
IfNLh-K0STmOyH7hD5nOwQ	6yyyuCAxLpq1f8ldzhKLt	1.35036
JkkuCUL0SySZDCtVeuiyWA	e40d71d0-3770-11e6-bc65-d302866e4688	1.34686
3xDxBxCcR7OfvR3Boce9fQ	3xDx8xCcR7OfvR3Boce9fQYlhpfBjQMOtylTzNAwwag	1.33745
23N2Xv9lSY683su9pW-JlQ	DfIBWTJfRM-Hl5c6MuPYgg	1.33524
oO-vJeahTwmrEB2VegVnNw	JkkuCUL0SySZDCtVeuiyWAoO-vJeahTwmrEB2VegVnNw	1.33237
JkkuCUL0SySZDCtVeuiyWA	JkkuCUL0SySZDCtVeuiyWAXodUc7iuQyKWOrneHbMNZA	1.33221
p4Ygo-l4SLWNjBtXMXqOJQ	ZTKAjGqzSwmxQ7OWMZOlLg	1.32836
gTccXtgSR9CBFKb5kE9iHA	bvPlEZXCQbe9jJZX1z2nqA	1.31859
3lAw9ZdGRuWZ6kqK-Lm-lQ	ClJxjljiTBuWQ4JaNeqztg	1.31456
LigHRqWzRceiIT7CisqHeg	BDRrvnjFpFS6Oe1Q792Rxl	1.31111
23N2Xv9lSY683su9pW-JlQ	fdrjvJKER7K1VWLp07sOBg	1.30774
RaXk5lbyScCVpw21s5lGlw	JkkuCUL0SySZDCtVeuiyWARaXk5lbyScCVpw21s5lGlw	1.30554
y-24chYuSeSXqASAPfkweg	jgUjrRx1PRp-VlJue4cOf	1.30483
y-24chYuSeSXqASAPfkweg	TOarlD91TMqqGBimSQ5Hwg	1.30483
3lAw9ZdGRuWZ6kqK-Lm-lQ	3lAw9ZdGRuWZ6kqK-Lm-lQSnpz6NaFT321LoGJlC5RXQ	1.30452
4ErsVbxPTH2FcHE7EwmQlg	4ErsVbxPTH2FcHE7EwmQlgJkkuCUL0SySZDCtVeuiyWA	1.3037
JkkuCUL0SySZDCtVeuiyWA	BXLlGtn3hHKK9TEBHELOEv	1.3016
gTccXtgSR9CBFKb5kE9iHA	ElYCfyU5TdKWb8PEs72O7w	1.29519
JkkuCUL0SySZDCtVeuiyWA	JkkuCUL0SySZDCtVeuiyWA	1.28683
T5f9qj0TRt6HafWOyo_6iw	JkkuCUL0SySZDCtVeuiyWAT5f9qj0TRt6HafWOyo_6iw	1.28444
y-24chYuSeSXqASAPfkweg	XCFVkmAoRe-DHmBNhmQ2Qg	1.28067
y-24chYuSeSXqASAPfkweg	DJPkt4TiJlQlqtJS9n9kMf	1.27484
BMomZGUbQROKn35F-JdtKQ	DzsGrlT-dL8odOmqFCQ4JQ	1.26569
JkkuCUL0SySZDCtVeuiyWA	BDRrvnjFpFS6Oe1Q792Rxl	1.26501
mJ9JptfDQ5mxxFAAjmbNqw	SUP-mJ9JptfDQ5mxxFAAjmbNqw	1.26382
3lAw9ZdGRuWZ6kqK-Lm-lQ	BGfH5ltdtEtqJ89niVOv4A	1.26003
a14mmkBATLKj_uoVit-ymw	JkkuCUL0SySZDCtVeuiyWAa14mmkBATLKj_uoVit-ymw	1.25968
JkkuCUL0SySZDCtVeuiyWA	BGjvHuPFxE3K9H1otla7s2	1.25635
JkkuCUL0SySZDCtVeuiyWA	n3Qs7n9RF7Z2ts-BXuuXC	1.25222
23N2Xv9lSY683su9pW-JlQ	Z1lIDZuGRC2lWDQskRBHvQ	1.24846
3lAw9ZdGRuWZ6kqK-Lm-lQ	w3ZJSWWUSCCbhUD_H8r8qw	1.24738
gTccXtgSR9CBFKb5kE9iHA	T5K5pTKhQ7WFfy6YJraQhQ	1.24671
gTccXtgSR9CBFKb5kE9iHA	5E1-tMecTeihhHt_sEk7lA	1.24664
3xDxBxCcR7OfvR3Boce9fQ	BXkG9B10BGjYrOvem6pg-x	1.24533
IfNLh-K0STmOyH7hD5nOwQ	BP3Y3EMOVLdZSVuVMM76-p	1.24331
a14mmkBATLKj_uoVit-ymw	BK76WCVC1EnooPrC1_B2OP	1.23153
3lAw9ZdGRuWZ6kqK-Lm-lQ	c647d220-3d5c-11e6-9a79-ddeeaa2a9da5	1.23086
IfNLh-K0STmOyH7hD5nOwQ	BPYhNgtYdP4KOMxTmUhVde	1.22075
Snpz6NaFT321LoGJlC5RXQ	aKdwg9B5LwafU3mQAsU6A	1.21562
3lAw9ZdGRuWZ6kqK-Lm-lQ	6bxWHlWaTAWWWyZO9o9cSQ	1.21342
gTccXtgSR9CBFKb5kE9iHA	DD-7-tNDpAQICU63bN13lh	1.2016
IfNLh-K0STmOyH7hD5nOwQ	CXucVr81NAb55S7XLsg9a3	1.19932
JkkuCUL0SySZDCtVeuiyWA	493bb9c0-376e-11e6-bc65-d302866e4688	1.19882

También tenemos que tener en cuenta que estos son datos generados a partir del dogfooding de la herramienta en el equipo de desarrollo y un poco tiempo de datos en estado beta, lo cual significa que durante este período los users han estado todo el tiempo familiarizando-se con la herramienta y seguramente utilizando paralelamente otra aplicación que utilizaran anteriormente para la colaboración.

Una vez tenemos los ratings implícitos vamos a utilizar los dos algoritmos que hemos evaluado para obtener ratings sugeridos.

Matrix Factorization

Con el algoritmo de Matrix Factorization tenemos buenos resultados, muchos muy pequeños, pero llegan hasta predecir ratings con valor 3.

Lo cual parece tener sentido, pues hemos comentado que en ratings implícitos llegábamos casi hasta el valor 3.

ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQ3jom-lYoTr2LTjcmQldRtw	1.37956
ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQ3xDxBxCcR7OfvR3Boce9fQ	1.21554
ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQ72VCdt2XTQqGr4oZHLLn2A	1.25405
ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQgTccXtgSR9CBFKb5kE9iHA	1.32706
ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQlfNLh-K0STmOyH7hD5nOwQ	0.8501
ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQlp9G2XqZS8eQgmGksTerBg	1.77342
ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQJkkuCUL0SySZDCtVeuiyWA	1.61527
ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQMbh_5-BiR1aa_sM-hMh2Cw	1.28087
ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQNllUNNmdRPO6KN9Ovs-u3g	1.56942
ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQoO-vJeahTwmrEB2VegVnNw	1.49078
ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQPMoKasBPSAWWeWKCjav4A	2.08386
ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQSnpz6NaFT321LoGJlC5RXQ	1.64072
ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQuW3l3lXzQlKtlyFf-XTlzQ	0.627454
ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQvHuuZybCSpq9c1jM7_03Jw	0.932072
ZeJsMBVtSKyqqwWk4HVL6Q 23N2Xv9lSY683su9pW-JlQy-24chYuSeSXqASAPfkweg	0.945857
ZeJsMBVtSKyqqwWk4HVL6Q 29954d30-37a9-11e6-a531-7fdd4b17202b	1.74372
ZeJsMBVtSKyqqwWk4HVL6Q 2ETxrlDYSOWuD7n7mN43cA	1.90941
ZeJsMBVtSKyqqwWk4HVL6Q 2ihrk56oSr6qkrGPuglplQ	1.05954
ZeJsMBVtSKyqqwWk4HVL6Q 2xlsuWrnQWabU6yar60ZTA	1.37013
ZeJsMBVtSKyqqwWk4HVL6Q 3a8844e0-3771-11e6-bc65-d302866e468B	1.58814
ZeJsMBVtSKyqqwWk4HVL6Q 3e4fb7b0-3772-11e6-bc65-d302866e468B	1.2575
ZeJsMBVtSKyqqwWk4HVL6Q 3hsH_j1pTMmsLcSpXOt-EA	0.967848
ZeJsMBVtSKyqqwWk4HVL6Q 3lAw9ZdGRuWZ6kqK-Lm-lQ	0.827669
ZeJsMBVtSKyqqwWk4HVL6Q 3lAw9ZdGRuWZ6kqK-Lm-lQ3JJ5ZDjDRqCo5naEmcpNZg	1.61679
ZeJsMBVtSKyqqwWk4HVL6Q 3lAw9ZdGRuWZ6kqK-Lm-lQ3xDxBxCcR7OfvR3Boce9fQ	1.39807
ZeJsMBVtSKyqqwWk4HVL6Q 3lAw9ZdGRuWZ6kqK-Lm-lQ72VCdt2XTQqGr4oZHLLn2A	1.81007
ZeJsMBVtSKyqqwWk4HVL6Q 3lAw9ZdGRuWZ6kqK-Lm-lQgTccXtgSR9CBFKb5kE9iHA	0.918614
ZeJsMBVtSKyqqwWk4HVL6Q 3lAw9ZdGRuWZ6kqK-Lm-lQGV8DGMFHQBCRkML5oCPç	1.78246
ZeJsMBVtSKyqqwWk4HVL6Q 3lAw9ZdGRuWZ6kqK-Lm-lQlfNLh-K0STmOyH7hD5nOwQ	0.665373
ZeJsMBVtSKyqqwWk4HVL6Q 3lAw9ZdGRuWZ6kqK-Lm-lQiSGXwbQqQAqlwuXR5qvGg	0.929913
ZeJsMBVtSKyqqwWk4HVL6Q 3lAw9ZdGRuWZ6kqK-Lm-lQJkkuCUL0SySZDCtVeuiyWA	1.2955
ZeJsMBVtSKyqqwWk4HVL6Q 3lAw9ZdGRuWZ6kqK-Lm-lQmJ9JptfDQ5mxxFAAjmbNqw	1.441B

Factorization Machine

Una vez hemos utilizado el algoritmo de Factorization Machine para predecir los ratings para los cuales tenemos valor 0.

Podemos ver que la mayoría de los valores predichos son muy cercanos a 0.

p4Ygo-I4SLWNjBtXMXqOJQ	juAT_W1zSMa2Z-2ABsYDCg	0.0377965
p4Ygo-I4SLWNjBtXMXqOJQ	3lAw9ZdGRuWZ6kqK-Lm-lQ	0.307131
p4Ygo-I4SLWNjBtXMXqOJQ	JkkuCUL0SySZDCtVeuiyWAMbh_5-BiR1aa_sM-hMh2Cw	0.172022
p4Ygo-I4SLWNjBtXMXqOJQ	BwgxdXHBpFfJur5DiFc7DN	0.0575673
p4Ygo-I4SLWNjBtXMXqOJQ	jgUjrRx1PRp-VlJue4cOf	0.0770356
p4Ygo-I4SLWNjBtXMXqOJQ	23N2Xv9lSY683su9pW-JlQvHuuZybCSpq9c1jM7_03Jw	0.090577
p4Ygo-I4SLWNjBtXMXqOJQ	LKK3eZ6sS8yAVJC0VNWVGg	0.0939231
p4Ygo-I4SLWNjBtXMXqOJQ	BcCrWFkktKZK3_iXMxFGqM	0.0643743
p4Ygo-I4SLWNjBtXMXqOJQ	X2xQMbQ0TTaKKMQ8ARJ2jg	0.0766612
p4Ygo-I4SLWNjBtXMXqOJQ	CqMzejpvdFVKYYYNhVkDl9	0.082687
p4Ygo-I4SLWNjBtXMXqOJQ	GVBDGMFHQBCRkML5oCPgLgJkkuCUL0SySZDCtVeuiy	0.0743011
p4Ygo-I4SLWNjBtXMXqOJQ	Cn4AW47n1KVK5dRZ8zWhH7	0.0506512
p4Ygo-I4SLWNjBtXMXqOJQ	JkkuCUL0SySZDCtVeuiyWAmJ9JptfDQ5mxxFAAjmbNqw	0.153802
p4Ygo-I4SLWNjBtXMXqOJQ	oO-vJeahTwmrEB2VegVnNwoO-vJeahTwmrEB2VegVnNw	0.0367464
p4Ygo-I4SLWNjBtXMXqOJQ	B6aLonfKQDu9Sq1AVXiumQ	0.192614
p4Ygo-I4SLWNjBtXMXqOJQ	23N2Xv9lSY683su9pW-JlQy-24chYuSeSXqASAPfkweg	0.211964
p4Ygo-I4SLWNjBtXMXqOJQ	GPq1__VNS5OptzrbEOYkig	0.0993184
p4Ygo-I4SLWNjBtXMXqOJQ	712d9f90-37c1-11e6-8dcd-d79c0160a0aa	0.0971761
p4Ygo-I4SLWNjBtXMXqOJQ	1AFHvmROQg6qkQsYFDcmig	0.0254683
p4Ygo-I4SLWNjBtXMXqOJQ	JkkuCUL0SySZDCtVeuiyWA	0.134255
p4Ygo-I4SLWNjBtXMXqOJQ	WOawdhn4Tju72n6LPe17NQ	0.107462
p4Ygo-I4SLWNjBtXMXqOJQ	rke6f9e9So2RJD3FwCeOxg	0.0883715
p4Ygo-I4SLWNjBtXMXqOJQ	XCFVkmAoRe-DHmBNhmQ2Qg	0.206146
p4Ygo-I4SLWNjBtXMXqOJQ	ElYChyU5TdKWb8PEs72O7w	0.132135
p4Ygo-I4SLWNjBtXMXqOJQ	vfk2-ZxII3YmataBBe2B7	0.201687
p4Ygo-I4SLWNjBtXMXqOJQ	BuYtYtGktlkq5L36xC_cgn	0.192908
p4Ygo-I4SLWNjBtXMXqOJQ	gTccXtgSR9CBFKb5kE9iHAoO-vJeahTwmrEB2VegVnNw	0.0874584
p4Ygo-I4SLWNjBtXMXqOJQ	CS-WwzvINCcKvfW1Km07Fx	0.0739665
p4Ygo-I4SLWNjBtXMXqOJQ	6d6TZ9xdPypHTeNKXdg5F	0.0831353
p4Ygo-I4SLWNjBtXMXqOJQ	72VCdt2XTQqGr4oZHLLn2A7DOvBT1YTdOkRRvYsFkEM	0.0901884
p4Ygo-I4SLWNjBtXMXqOJQ	GoBxiLTdQo-58-f_d0FWdQJkkuCUL0SySZDCtVeuiyWA	0.116979
p4Ygo-I4SLWNjBtXMXqOJQ	Dw9tBlCmxAVa6_Yf53yP5V	0.0977249
p4Ygo-I4SLWNjBtXMXqOJQ	AwXJHiYXR6eKiQF64N54zQJkkuCUL0SySZDCtVeuiyWA	0.118946
p4Ygo-I4SLWNjBtXMXqOJQ	lzJXXd4XS-ufClzo8K533w	0.0379963
p4Ygo-I4SLWNjBtXMXqOJQ	b8d34a20-37b2-11e6-8f24-5f9d3d62535e	0.0320536
p4Ygo-I4SLWNjBtXMXqOJQ	JkkuCUL0SySZDCtVeuiyWALigHRqWzRceiIT7CisqHeg	0.107367
p4Ygo-I4SLWNjBtXMXqOJQ	SUP-CQeTumV6RQScfG2Yvzl1Qg	0.0101481
p4Ygo-I4SLWNjBtXMXqOJQ	c647d220-3d5c-11e6-9a79-ddeeaa2a9da5	0.119906
p4Ygo-I4SLWNjBtXMXqOJQ	cd747dc0-376f-11e6-bc65-d302866e4688	0.166809
p4Ygo-I4SLWNjBtXMXqOJQ	h0N0k7zXQ9usRZvXTFgt4A	0.110436
p4Ygo-I4SLWNjBtXMXqOJQ	T5K5pTKhQ7WFfy6YJraQhQ	0.151631
p4Ygo-I4SLWNjBtXMXqOJQ	sHvwSPWITRmbCcfG3qFraQ	0.0455121
p4Ygo-I4SLWNjBtXMXqOJQ	JkkuCUL0SySZDCtVeuiyWAiSGXwbQqQAqliwuXR5qvGg	0.149626
p4Ygo-I4SLWNjBtXMXqOJQ	0b32b210-3772-11e6-bc65-d302866e4688	0.166798
p4Ygo-I4SLWNjBtXMXqOJQ	Q5oaBGaiSom5B0BSi5G9MA	0.0652711
p4Ygo-I4SLWNjBtXMXqOJQ	CqwzejrDR5StVjaCTlyChQ	0.104656
p4Ygo-I4SLWNjBtXMXqOJQ	DBTXlbGhdNbLNymsxpSWAv	0.0845044
p4Ygo-I4SLWNjBtXMXqOJQ	Mbh_5-BiR1aa_sM-hMh2Cw	0.08042

El valor máximo de rating predicho es 0.5.

Ya en el anterior apartado habíamos notado que los valores predichos eran notablemente inferiores a los dados como entrada al algoritmo.

Pero hemos visto que aunque los valores sean pequeños con estos datos este algoritmo es más eficiente que el de MatrixFactorization. Ahora podemos probar distintos módulos, hasta ahora tan solo utilizamos el módulo MCMC. Pero la librería que utilizamos tiene posibilidades de utilizar otros módulos.

A continuación probaremos con otros módulos con los mismos datos.

Con el módulo ALS los valores que tenemos también son menores a 0.5.

ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAmJ9JptfDQ5mxxFAAjmbNqw	0.217004
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAMZ1Y_gjASfOBdJjR9kl5Yw	0.20799
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWANIlUNNmdRPO6KN9Ovs-u3g	0.196581
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWANU4CtnXXRsiB1EUcp8n7QQ	0.179789
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAoO-vJeahTwmrEB2VegVnNw	0.243568
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAoTBuY4hiTxmSb19eMOcSGw	0.188703
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAp4Ygo-I4SLWNjBtXMXqOJQ	0.170641
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAPMoKas8PSAWWeWKCjav4A	0.182558
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWApNHTqrAwS_ODM-wHu5oNpQ	0.187705
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAq5p0SFyfSxiZglQs-snQ1Q	0.120781
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAQv8abNGVQvmkH7Kdv8WiH	0.1604
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWARaXk5IbyScCVpw21s5lGlw	0.204564
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWArELZ-LmJQv-VyxFvpMOkAg	0.17964
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWASnpz6NaFT321LoGJlC5RXQ	0.173616
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAT5f9qj0TRt6HafWOyo_6iw	0.149512
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWATNzsGOmjQz-HGCZlXrNw7w	0.155228
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAtRCqaW1dRwSWQ9zue_jffg	0.174652
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAU_QgA-QlRDW3ANgSd5socw	0.187257
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAuW3l3lXzQlKtlyFf-XTlzQ	0.186746
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAvHuuZybCSpq9c1jM7_03Jw	0.174652
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAXodUc7iuQyKWOrneHbMNZA	0.185907
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAy_NUhYi9Q4yF1aU4ED-9yQ	0.188962
ZeJsMBVtSKyqqwWk4HVL6Q JkkuCULOSySZDCtVeuiyWAy-24chYuSeSXqASAPfkweg	0.120481
ZeJsMBVtSKyqqwWk4HVL6Q jQj37ERGSRmMimRYA2vliw	0.147141
ZeJsMBVtSKyqqwWk4HVL6Q juAT_W1zSMa2Z-2ABsYDCg	0.132521
ZeJsMBVtSKyqqwWk4HVL6Q jvlTgiB2QHaDSTpB2YN-sg	0.165097
ZeJsMBVtSKyqqwWk4HVL6Q k20tbduKQKayT-wvHs99kw	0.177148
ZeJsMBVtSKyqqwWk4HVL6Q k2LvwcMQRYaiE1ohSsj5BQ	0.167207
ZeJsMBVtSKyqqwWk4HVL6Q kBpTemBxTOqD_4PDvhroDg	0.0892157
ZeJsMBVtSKyqqwWk4HVL6Q kEh9TgHBR9-AyKlzHb1rMw	0.1198
ZeJsMBVtSKyqqwWk4HVL6Q kkBwRBjvRQGNSgtADv2-qw	0.166563
ZeJsMBVtSKyqqwWk4HVL6Q KpFTx7z-QkO5FSArtM1Giw	0.16647

En cambio, con el módulo SGD los valores predichos tienen mayor dispersidad, y llegan a puntuar parejas user-spaces con valores de 4…

YZGMAqk7S7ygpOHZH432UvNPlzkwwJPrlkB1aORjqkd	2.19769
YZGMAqk7S7ygpOHZH432UvnU5EFDQwT4uuLjwW-k3ZBA	1.41765
YZGMAqk7S7ygpOHZH432UvnZA6Ki3dBpKZ1nqXV-0v1	2.20768
YZGMAqk7S7ygpOHZH432UvO5rzA5fhLBoDUMHyFYMly	1.57949
YZGMAqk7S7ygpOHZH432Uvobj7aKj8SlqklSWgjf-p2A	2.10221
YZGMAqk7S7ygpOHZH432UvOCbto45jRluVQGFSMxBPoQ	1.66568
YZGMAqk7S7ygpOHZH432UvolydgBSSQH-VHWo_s0FSFw	1.14826
YZGMAqk7S7ygpOHZH432UvoO-vJeahTwmrEB2VegVnNwoO-vJeahTwmrEB2VegVnNw	0.76398
YZGMAqk7S7ygpOHZH432UvOQcmlyCdCJapB_oP41ilx	2.02079
YZGMAqk7S7ygpOHZH432UvoxQNplyVS8SulVU40lmTLA	1.89614
YZGMAqk7S7ygpOHZH432Uvp4Ygo-l4SLWNjBtXMXqOJQ	2.19938
YZGMAqk7S7ygpOHZH432Uvp4Ygo-l4SLWNjBtXMXqOJQuW3l3lXzQlKtlyFf-XTlzQ	0.999479
YZGMAqk7S7ygpOHZH432UvpEL0i4VpCU4gbkljuCy97	2.23775
YZGMAqk7S7ygpOHZH432UvPMoKas8PSAWWeWKCjav4Aw	1.38214
YZGMAqk7S7ygpOHZH432UvpNHTqrAwS_ODM-wHu5oNpg	1.16254
YZGMAqk7S7ygpOHZH432UvPXwSTcnxAyLWFrXiiJhlM	1.10716
YZGMAqk7S7ygpOHZH432Uvq3w9_qazQm-EBZxAHM9grQ	2.34077
YZGMAqk7S7ygpOHZH432UvQ5oaBGaiSom5B0BSi5G9MA	0.860996
YZGMAqk7S7ygpOHZH432UvQ7ev9Ow4SMuD_yZ_B5Z6Eg	1.94254
YZGMAqk7S7ygpOHZH432UvQBqCgCF5SOGw5JghBeKBYQ	1.69041
YZGMAqk7S7ygpOHZH432UvqLsGbKuaSwWmAEUjBOAl4g	1.35926
YZGMAqk7S7ygpOHZH432UvqsjTKmCCRLuaEbp0LNbs3A	1.49002
YZGMAqk7S7ygpOHZH432UvQv8abNGVQvmkH7Kdv8WiHw	1.69138
YZGMAqk7S7ygpOHZH432UvQv8abNGVQvmkH7Kdv8WiHwtRCqaW1dRwSWQ9zue_jf	1.40836
YZGMAqk7S7ygpOHZH432UvQWvchN_2QMWJFkHXzUp2HA	0.840765

Entonces con SGD tenemos valores que parecen más cercanos a valores que nos esperaríamos de la salida del recomendador.

También hay un módulo de Ranking en la librería fastFM. Al utilizar este módulo los resultados también son coherentes aunque muchos son menores a 1 y hay muchos resultados negativos, hasta el orden de -4.

No tengo claro que significado tiene las recomendaciones negativas, pero viendo las positivas parecen tener sentido.

Este módulo es un módulo especial para obtener ratings. Según la página de github de fastFM, podemos ver que recomiendan módulos según sea tu interés.

Task	Solver	Loss
Regression	Als, mcmc, sgd	Square Loss
Classification	Als, mcmc, sgd	Probit(Map), Probit, Sigmond
Ranking	Sgd	bpr

BPR es también como se conoce el módulo de Ranking.

p4Ygo-l4SLWNjBtXMXqOJQ	1X7HCiXET3e6tRR5CVk3hA	0.758026
p4Ygo-l4SLWNjBtXMXqOJQ	843heFEZTNKmLB6lRbBK-g	1.90115
p4Ygo-l4SLWNjBtXMXqOJQ	23N2Xv9lSY683su9pW-JlQuW3l3lXzQlKtlyFf-XTlzQ	2.4545
p4Ygo-l4SLWNjBtXMXqOJQ	am9TJ001SMq8Bu7lD8MKcg	2.89098
p4Ygo-l4SLWNjBtXMXqOJQ	94LzWeFplp5pGmOTvuSOC	2.50525
p4Ygo-l4SLWNjBtXMXqOJQ	ewPSr6zLTuGcxjMutcWpmQ	1.81451
p4Ygo-l4SLWNjBtXMXqOJQ	jQj37ERGSRmMimRYA2vliw	1.03809
p4Ygo-l4SLWNjBtXMXqOJQ	BFbEg8Mm5Bc5bsWUuouhKD	3.04463
p4Ygo-l4SLWNjBtXMXqOJQ	9Yf2uzW8TeKTzgWPs5SfNg	2.40123
p4Ygo-l4SLWNjBtXMXqOJQ	gT5YDCMpPt7UPhsvqf6Zq	1.94222
p4Ygo-l4SLWNjBtXMXqOJQ	SUP-Y33V2ko3RKqDzvGgFBZ4ng	2.35495
p4Ygo-l4SLWNjBtXMXqOJQ	OQcmlyCdCJapB_oP41ilx	2.72742
p4Ygo-l4SLWNjBtXMXqOJQ	3lAw9ZdGRuWZ6kqK-Lm-lQ3JJ5ZDjDRqCo5naEmcpNZg	2.33419
p4Ygo-l4SLWNjBtXMXqOJQ	72VCdt2XTQqGr4oZHLLn2AgTccXtgSR9CBFKb5kE9iHA	3.16622
p4Ygo-l4SLWNjBtXMXqOJQ	YZGMAqk7S7ygpOHZH432Uw	2.91163
p4Ygo-l4SLWNjBtXMXqOJQ	493bb9c0-376e-11e6-bc65-d302B66e4688	2.69044
p4Ygo-l4SLWNjBtXMXqOJQ	23N2Xv9lSY683su9pW-JlQlp9G2XqZS8eQgmGksTerBg	2.08032
p4Ygo-l4SLWNjBtXMXqOJQ	SUP-TNzsGOmjQz-HGCZlXrNw7w	1.48875
p4Ygo-l4SLWNjBtXMXqOJQ	3xDx8xCcR7OfvR3Boce9fQSnpz6NaFT321LoGJlC5RXQ	2.74081
p4Ygo-l4SLWNjBtXMXqOJQ	Ct7x-apFxHHbDET_Tz3pFP	2.65134
p4Ygo-l4SLWNjBtXMXqOJQ	3lAw9ZdGRuWZ6kqK-Lm-lQuW3l3lXzQlKtlyFf-XTlzQ	1.4145
p4Ygo-l4SLWNjBtXMXqOJQ	fGapWS_aQ9W_T0Jxh2ZZpw	1.91874
p4Ygo-l4SLWNjBtXMXqOJQ	23N2Xv9lSY683su9pW-JlQSnpz6NaFT321LoGJlC5RXQ	2.25348
p4Ygo-l4SLWNjBtXMXqOJQ	SUP-72VCdt2XTQqGr4oZHLLn2A	3.13884
p4Ygo-l4SLWNjBtXMXqOJQ	Q7ev9Ow4SMuD_yZ_B5Z6Eg	1.47645
p4Ygo-l4SLWNjBtXMXqOJQ	ClJxjljiTBuWQ4JaNeqztg	1.26543
p4Ygo-l4SLWNjBtXMXqOJQ	HWCojmZcT2SAVrCBulVFqQ	3.09743
p4Ygo-l4SLWNjBtXMXqOJQ	TBsOZRBnQdK-nSmFEGDZew	2.26682
p4Ygo-l4SLWNjBtXMXqOJQ	BGjvHuPFxE3K9H1otla7s2	2.90663

Tenemos que tener en cuenta que la regresión (el cálculo que hemos hecho con todos los módulos) nos devuelve valores entre 0 y 1. Por eso multiplicamos ese valor por 5 (en todos los casos) para tenerlo en una escala más fácil de entender.

El problema es que no podemos estar seguros de la precisión del algoritmo, pues no tenemos forma de hacer el benchmarking del mismo. Pero podemos ver que los resultados sean coherentes, y parece que el único módulo que produce resultados coherentes es el de SGD y Rank. Así que trabajaremos con ese módulo cuando queramos probar la eficacia del recomendador contrastando con los users reales.

El benchmarking no podemos hacerlo porque no sabemos realmente el valor que un user le daría a un space. Los valores que tenemos son valores que hemos calculado nosotros, y que por tanto, no podemos tener la seguridad de que sean correctos.

Pero para ver que tal funcionan los métodos de recomendación podemos guardarnos parte de los datos que tenemos para ver que tal se le da al algoritmo predecir esos datos. Esta es una práctica común que pusimos a prueba a la hora de evaluar los algoritmos con movielens.

Para ello hemos escrito un código que calculará el error rmse dados nuestros datos de prueba contrastandolos con un 20% de los datos que nos guardaremos. Para tener una mayor certeza de que el cálculo del rmse es correcto lo haremos 20 veces cambiando las muestras que tendremos de training y de prueba. Para ello usamos la librería de python *Sklearn.*

Los resultados que hemos obtenido con los distintos módulos son:

Módulo	Time	RMSE
Rank (bpr)	0.6402	0.7332
SGD	0.3819	0.6187
MCMC	1.4353	0.0267
ALS	0.9242	0.0137
MF	23.7059	0.5226

Por lo tanto podemos ver que el módulo de Ranking és el que mejor resultados tiene. Como nos dimos cuenta anteriormente, tan solo evaluando los resultados, los módulos que mejor trabajan son el de MF el de SGD y el de Rank. El resto de módulos dan resultados poco coherentes y viendo el rmse nos damos cuenta de que es realmente malo el error. Hay que tener en cuenta que dentro del tiempo esta contado desde que obtenemos los datos de la tabla user_metrics hasta que conseguimos las recomendaciones. También podemos ver como el método de MF es considerablemente más lento que FM.

Por ahora este proceso de investigación ha llevado a la implementación de un recomendador con distintos algoritmos posibles y distintos resultados. Ahora tendremos que volver a plantearnos que estamos buscando para revisar este proceso y construir algo que nos ayude en el futuro de la herramienta de colaboración.

Conclusiones

Después de las pruebas realizadas y los resultados obtenidos hemos llegado a la conclusion de que el algoritmo que mejor funciona es el de Factorization Machine y de que el módulo más propicio es el de Ranking.

Una vez probado con datos reales, deberíamos ajustar los coeficientes para ajustar el puntaje lo más posible a lo que el usuario valoraría. Pues no debemos olvidarnos que este cálculo es un cálculo implícito y no podemos estar seguros de que el usuario los valoraría de la misma manera.

Por ejemplo, muchos flow items leídos y pocos creados podría significar tanto que le interesa mucho como que le interesa poco.
Puede ser que no le interese nada y solo entre en el space para que se le vaya el indicador que dice que no ha leído los mensajes y por eso no crea flow items.
Pero también puede ser que le interese mucho, que lea siempre los flow items por interés pero que no cree ninguno porque no tiene permisos o porque siente demasiado respeto por la conversación que se está manteniendo.
Nunca podremos saber cuál es el tipo de interacción que está manteniendo el usuario con los datos que tenemos, pero lo que hacemos es un software que tenga un comportamiento correcto en el 80% de los casos, y deberemos ajustar los coeficientes para que los puntajes se acerquen el máximo posible a la valoración real del user. Para conocerla podemos preguntarle explícitamente para contrastar nuestros cálculos implícitos.

Para este marco parece un poco raro el hecho de valorar un space. Por eso, capaz un algoritmo de clasificación, con algunas reacciones que tan populares se han hecho últimamente sea más adecuado. Facebook tiene distintas reacciones posibles. Si permitimos que cada usuario reaccione a cada space y a cada elemento dentro del mismo tendríamos una entrada muy valorada para el recomendador. Una entrada explicita muy interesante, que permitiría (dependiendo del sistema de reacciones) recomendar elementos a los usuarios.
De esta manera podremos tener un recomendador que tenga directamente los datos en “bruto” para que pueda sacar conclusiones para llegar a lo que nosotros le proporcionamos.
Actualmente, dada la temprana fase en la que esta este software, no podemos implementar este sistema de reacciones o alguno parecido de valoraciones. Pero este trabajo ha ayudado a ver con que visión enfocar esta meta que tendría el recomendador.

Como hemos visto a la hora de estudiar el estado del arte de los recomendadores, lo que parece mas propicio para nuestro sistema es la idea de hacer un pre-procesamiento de los

datos para ayudar al recomendador, pero permitirle acceder a la mayor cantidad de “datos brutos” que pueda, para que el algoritmo tenga mayor capacidad de aprendizaje.
Para el benchmarking y el ajuste del algoritmo deberemos de alguna manera contrastar estos datos implícitos con datos explícitos dados por los usuarios.
Esto es porque hasta ahora hemos generado nosotros un rating o una valoración, pero esta es totalmente objetiva, basada en datos implícitos y aunque podamos ajustarla a nuestra conveniencia con los coeficientes de impacto no seguirá siendo mas que una generalización y un intento de expresar esa valoración de usuario que no tenemos.
Por eso hemos evaluado distintos caminos para la recolección de datos explícitos.
1 - Rating
El rating es algo mundialmente conocido, en general se basa en una escala de estrellas de 0-5. Siendo 0 el peor valor y 5 el máximo. Este sistema esta muy bien para una web de productos como películas, hoteles, restaurantes… Pero para nuestro contexto, el cual esta en una herramienta de colaboración. Es difícil para el usuario, saber que esta valorando con esa puntuación, y quien va a tener acceso a esa información.

En general es un poco extraño tener que valorar un contexto de colaboración. Quien puede puntuar eso de forma consciente y objetiva?
El administrador? Todos los usuarios? Todas las valoraciones tienen el mismo peso?

Que estarían valorando?
La utilidad del space? El rendimiento del mismo? Lo que nos gusta colaborar en ese space?
Es difícil que alguien valore de forma objetiva un space. No queda claro bajo que consigna debería valorarlo.

2 - Reacciones
Por otro lado con el crecimiento de las reacciones (FaceBook, paso de tan solo Like a tener otras acciones posibles por elemento).
Este tipo de “valoración” es mas fácil para el usuario pues estas dándole un sentimiento a un elemento.
A la hora de elegir las reacciones posibles hay distintos elementos a tener en cuenta.
El primero es la cantidad de reacciones posibles. Si damos la posibilidad de elegir muchas reacciones a los usuarios les sera difícil elegir y pocos coincidirán. Por ejemplo en slack podemos reaccionar con cualquiera de los emoticonos posibles del chat.
Esto permite al usuario la libertad de elección y participación. Es un tipo de reacciones mas orientadas a los demás usuarios y a su interpretación. Pero como entrada de el algoritmo sera mas pobre.

En cambio si elegimos un conjunto pequeño de reacciones podemos intentar clasificar los elementos respecto a estas reacciones, de esa forma poder predecir que clasificación le daría un user a un space. Y de ahí elegir nuestras recomendaciones.
Otra gran duda que nos asalta ahora es que reacciones elegir.
Facebook eligió estas:

- **Me gusta**
- **Me encanta**
- **Me divierte**
- **Me alegra**
- **Me sorprende**
- **Me entristece**
- **Me enfada**

Como nosotros no estamos en una red social, sino en un contexto de colaboración en el trabajo las diferencias entre las 4 primeras reacciones, podríamos juntar-las en dos que podrían ser, me gusta y me alegra. Porque me encanta y me divierte son mas dirigidas a una herramienta de ocio.
Después seguramente podríamos poner una reacción que significara utilidad. Que es uno de los objetivos que promueve la herramienta. Podría ser me sirve o me ayuda.
También me parece interesante la reacción contraria a me sirve o me ayuda para dar la posibilidad de demostrar cuando un contexto de colaboración esta empezando, como es usual, a dejar de funcionar como debería. Lo cual es subjetivo pero es una buena categoría en la cual clasificar elementos.
Las reacciones de me sorprende, me entristece y me enfada bien podrían ser categorías interesantes.
Después deberíamos tener en cuenta como mostramos estas reacciones.
Si es correcto mostrar la reacción que cada uno tiene a cada cosa. Para los elementos tiene sentido, pero capaz, para los spaces podrían ser anónimas y mostrar tan solo las categorías predominantes para dar una idea de la “valoración” de los users.

Viendo entonces estos sistemas pareciera que el mas interesante para incorporar seria el de reacciones, aunque deberíamos afinar cuantas y que reacciones son interesantes ofrecer.
Con esta visión, nuestro sistema podría basarse entonces un sistema de clasificación a partir de estas categorías, teniendo como entradas los datos limpios (tan solo una relativización de los datos brutos respecto a los datos totales y máximos, tal como hemos dividido nuestro calculo de rating en cuatro tipos de ratings).

Después de analizar el sistema desarrollado y los resultados obtenidos parece interesante añadir una columna de fecha a la tabla de ratings implicitos y a la tabla de ratings sugeridos. Esto permitiría tener un control sobre la fluctuación de los ratings implícitos y sugeridos para ver como se ha desarrollado el recomendador y como a ido evolucionando el uso de los spaces por los users. Esto permitiría también ayudar en el benchmarking del recomendador, pues después de haber recomendado un space y que la relación space-user empiece a funcionar. Podremos ver que tan acertada estaba nuestra predicción con respecto a los datos generados a partir del uso.

Después del lanzamiento de *Kezmo* beta, dado el contacto que hay con el equipo de desarrollo del mismo, parece ser que uno de los problemas sin solución que tienen ahora mismo es el de el ordenamiento o priorización de contenido para el usuario.

Dentro de un contexto con mucho contenido, hay que priorizar para el usuario los elementos que deben ser más interesantes para el. Y uno de las primeras fases de este problema es el de el ordenamiento de spaces para el usuario.

Hasta ahora habíamos visto como objetivo el poder recomendar spaces para los users, pero dada la problemática actual, y el software desarrollado, podemos reutilizarlo o re-dirigirlo para que el rating implícito calculado para cada pareja space-user con interacción nos sirva para decidir el orden en el que se muestran los spaces a los users.

Este orden tampoco podría ir cambiando mucho, porque produciría confusiones a los usuarios. Pero con nuestro algoritmo solo cambiarían diariamente, y seguramente no mucho. Lo que parece razonable.

Otro punto de este problema sería el de ordenar o priorizar contenido dentro del space, pero ese ya sería un problema más complicado y si necesitaríamos entrada granulada a menor nivel.

Un sistema de reacciones también podría ayudar en este sentido, por lo que parece algo interesante a tener en cuenta para el futuro de la aplicación.

Como este es un sistema que aún esta en fase beta, podemos innovar y probar conceptos que en otro tipo de sistemas sería más arriesgado probar.

Con el algoritmo desarrollado podemos implementar un demon que cargue la tabla user_metrics y después corra SFRater para generar los ratings implícitos. Y consultar el rating implícito a la hora de mostrarlos a los usuarios para así tener un orden personalizado que no sea por reciente o por orden alfabético.

Tan solo estaría bien modificar-lo brevemente para que añada el día como habíamos comentado anteriormente.

También estaría bien que el demon, una vez ejecutada la primera vez, tan solo recorriera los datos del último día. Pues no es necesario recalcular todos los datos cada vez.

Para ajustar el rating tendremos que ajustar los coeficientes de acción de cada uno de los parámetros al rating final.

Esto generaría una consulta más a la hora de devolver los spaces, quizás un join para que cada plataforma los muestre según le parezca relevante.

Con esta aplicación del software desarrollado podemos ver que la investigación tiene sentido, y que en una aplicación en que la información es un concepto tan importante, el data mining, el procesar los datos para extraer conclusiones es muy interesante.

En un contexto de colaboración dentro del trabajo en una empresa grande (+ de 1000 empleados) esta herramienta permite colaborar en contextos cerrados y compartir toda clase de contenido en ese contexto. Respecto a las sugerencias de space-user, durante el desarrollo de la investigación no han aparecido más casos en los que podamos utilizarlo que no sean los planteados en la introducción. Después de plantearlo más parece más interesante un recomendador de contenido que un recomendador de contextos de colaboración.

Porque aunque el hecho de encontrar afinidades ocultas entre spaces y users es una buena información que puede ser útil en una organización grande, que de alguna manera la herramienta me recomiende contenido al que yo si tengo acceso pero no he visto. Facebook de alguna manera hace eso con la bandeja de últimas noticias. Podría ser que los eventos del sistema generaran tarjetas que después pudiesen ser vistas en un scroll infinito.

Podríamos tener un container o un lugar al que pudieses ir dentro de un space y ver que pasó en las últimas horas en el mismo. Podríamos ver eventos cuando alguien modificó algun container item, envío un mensaje relevante (un mensaje con muchas reacciones) o algún otro contenido que parezca interesante mencionar. Análogamente podríamos tener un lugar general en el que esté la información de todos los spaces a los que tengo acceso.

En ese caso tendríamos muchísimos eventos y de alguna manera deberíamos priorizarlos. Para ello podríamos utilizar el recomendador de contenido.

De todas formas pienso que para llegar a un recomendador de contenido es importante empezar con recomendaciones de space, porque cabe suponer, que el contenido de un space te interesará también en función de lo que te interese de un space y viceversa. Si tuviésemos un sistema de reacciones podríamos seleccionar bien las mismas y así los users pueden dar feedback de forma sencilla a otros users sobre como está funcionando ese contexto de colaboración. Así como a mensajes y otro tipo de contenido permitirían tener una valiosa entrada para recomendar contenido.

Una vez tuviésemos una lista de eventos priorizados en función del user en cuestión, solo hay que intentar maximizar la precisión de la misma y permitir acceder a ella de forma sencilla.

En esta especie de últimas noticias hay un potencial interesante para el mundo de la colaboración. Podrías ver un feed de lo que tus colaboradores hayan generado en tu ausencia. Esto es muy interesante, pues actualmente con las herramientas de colaboración ágiles es difícil saber que se decidió en tu ausencia, pues pueden haber muchos mensajes después que sean irrelevantes. Así que encontrar de casualidad el que es relevante es una tarea difícil. En cambio si pudiésemos diferenciar la importancia de un mensaje y otro, podríamos decidir que mensajes son interesantes mostrar en el feed de eventos. Para ello debemos permitir acciones sobre los mensajes para que puedan tener un vector de features que los describa. El hecho de generar un sistema de reacciones, a parte de para poder clasificar los elementos respecto a esas categorías, permite a los usuarios dar más información sobre un elemento de la aplicación. Ya sea un space, un flow item o un container item.

Si las problemáticas que están surgiendo ahora son relacionadas con priorizar spaces, el hecho de priorizar contenidos sería un nivel mucho más avanzado y mucho más interesante. Si yo colaboro en varios spaces es muy difícil estar al día de todos, pero esta herramienta me permitiría hacer un seguimiento más cercano de la actividad en un space de manera más sencilla.

Algo a tener en cuenta a la hora de recomendar contenido de un chat es como funciona el mismo.

El chat es un elemento vivo, generado a partir de conversaciones, se van generando conversaciones sobre algun tema y estas generan varios mensajes de ida y vuelta.

Si quisiésemos generar eventos describiendo actividad en el chat, sería más interesante hablar de conversaciones que de mensajes. Para ello deberíamos segmentar el contenido del chat en conversaciones. Para ello una primera versión podría venir por la marca de tiempo. Si hay una ventana de tiempo mayor a x entre dos mensajes considero que son de dos conversaciones distintas, aunque este método es sencillo rápidamente podemos ver que no será muy exacto.

Pensando en este tipo de segmentación y en intentar mejorar la herramienta para que el contenido sea más sencillo de encontrar. Parece muy interesante generar un tipo de container que sea post, y que fácilmente podamos pasar de flow item a post (a posteriori de su creación).

Para que ese flow item quede perdurado en el container de post y se pueda generar una conversación a su alrededor. Si esta función se utilizara bien podríamos entonces tener la mayoría de conversaciones importantes en un lugar de fácil acceso. Por ese lado también parece interesante marcar flow items como importantes para que generen un container item y puedan ser accedidos desde el contenido estructurado.

Como el objetivo de esta herramienta es el de facilitar la colaboración ágil mezclando la con la colaboración estructurada, tenemos que pensar mecanismos para facilitar al user la tarea de convertir el contenido desestructurado en estructurado.

Referencias

A Detailed Introduction to K-Nearest Neighbor (KNN) Algorithm
17 Mayo 2010
https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/

Top 10 Algorithms in Data Mining.
4 Diciembre 2007
http://www.cs.umd.edu/~samir/498/10Algorithms-08.pdf

What is a layaman's explanation of Matrix Factorization in Collaborative Filtering
18 Enero 2015
https://www.quora.com/What-is-a-laymans-explanation-of-matrix-factorization-in-collaborative-filtering

Testing implementations on LibFM
15 Febrero 2016
https://arogozhnikov.github.io/2016/02/15/TestingLibFM.html

Support Vector Machines for Collaborative Filtering
22 Enero 2006
http://sci2s.ugr.es/keel/pdf/specific/congreso/xia_dong_06.pdf

Recommendation system based on Collaborative Filtering
12 Diciembre 2008
http://cs229.stanford.edu/proj2008/Wen-RecommendationSystemBasedOnCollaborativeFiltering.pdf

Recommendation Systems
22 Diciembre 2014
http://infolab.stanford.edu/~ullman/mmds/ch9.pdf

Recommender Systems
21 Julio 2014
http://www.slideshare.net/xamat/recommender-systems-machine-learning-summer-school-2014-cmu

Tutorial Recommender Systems
4 Agosto 2013
http://ijcai13.org/files/tutorial_slides/td3.pdf

Factorization Machines
11 Octubre 2010
http://www.csie.ntu.edu.tw/~b97053/paper/Rendle2010FM.pdf

Factorization Polnomial Regression Models
6 Noviembre 2011
https://www.ismll.uni-hildesheim.de/pub/pdfs/FreudenthalerRendle_FactorizedPolynomialRegression.pdf

What are Factorization Machines and how are they used in Machine Learning
24 Julio 2015
https://www.quora.com/What-are-factorization-machines-and-how-are-they-used-in-machine-learning

Factorization Machines a new way of looking Machine Learning
5 Noviembre 2015
https://securityintelligence.com/factorization-machines-a-new-way-of-looking-at-machine-learning/

Matrix Factorization: A simple tutorial and implementation in Python
16 Septiembre 2010
http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python/