Framework de recomendación P*yreclab*.

Gabriel Sepúlveda Villalobos
Pontificia Universidad Católica de Chile
grsepulveda@ing.puc.cl

Abstract: Dentro de las tareas de recomendación, existen dos componentes fundamentales que permiten llevarlas a cabo, tanto para la predicción de ratings, como para la recomendación de ítems. Por un lado, tenemos los datos adquiridos desde un sistema real, desde el cual se obtienen medidas que relacionan el nivel de afinidad de usuarios con determinados ítems. Por otra lado, se encuentran los algoritmos capaces de convertir esos datos en información útil, para predecir gustos de usuarios, y así, poder realizar recomendaciones de calidad. En este ámbito, existen múltiples herramientas de software de libre acceso, pero que no logran concentrar una batería completa de herramientas de recomendación que permitan tener un solo ambiente para la realización de experimentos y análisis. Por este motivo, se ha desarrollado la biblioteca Pyreclab, que intenta cubrir estas falencias proporcionando métodos de recomendación de fácil uso, y que se espera puedan ir ampliando su cobertura a través del tiempo.

Key Words: Sistemas de recomendación, *Python*, algoritmos, dataset, *MovieLens*, factorización matricial.

1. Introducción

A grandes rasgos, todo problema de recomendación tiene dos grandes aristas. La primera de ellas correspondiente a los datos, y cumple el rol de proporcionar un conjunto de valores desde los cuales se espera extraer algún tipo de información con respecto a las relaciones internas del sistema, con el fin de poder predecir relaciones futuras que aún no se han manifestado. En particular, dado un grupo de usuarios que puede consumir de un grupo ítems, se espera poder determinar la afinidad que cada sujeto pueda tener con ciertos productos no consumidos, para lograr predecir que relaciones tienes mayor potencial de ser establecidas entre ambos conjuntos.

Por otra parte, la segunda arista corresponde a las técnicas que pueden ser aplicadas sobre los datos, con el objetivo de convertirlos en información que nos lleve a aumentar nuestro conocimiento del sistema, y haga posible la tarea final, que consiste en la recomendación de ítems a usuarios.

Hoy en día existe una gran variedad de técnicas que se aplican de forma satisfactoria a una gran diversidad de problemas. Pero al llegar a este punto, a menudo nos encontramos con la dificultad de no contar con herramientas de cálculo que satisfagan por completo nuestras necesidades, ya sea por que las que se encuentran disponibles de forma gratuita son limitadas, o por simple desconocimiento.

En este trabajo se presenta el desarrollo de una herramienta de software para la generación de recomendaciones, el cual pretende alcanzar un alto nivel de eficiencia en cuanto a la utilización de recursos computacionales, y entregando al usuario una interfaz simple y amigable que permita la experimentación con datasets con información de ratings, y que provengan de diversas fuentes.

Este documento posee la siguiente estructura: en el punto (2), se entrega una breve reseña de algunas de las herramientas de software de uso libre, más populares en este ámbito. En el punto (3), se presenta la solución de software desarrollada, como alternativa a las estudiadas. En el punto (4), se dan detalles sobre el dataset utilizado para obtener medidas de rendimiento del desarrollo. En los puntos (5) y (6), se presenta la metodología de los experimentos realizados, y los resultados obtenidos para diferentes variantes de los algoritmos analizados. Para finalmente, en el punto (7) plantear las conclusiones del trabajo desarrollado.

2. Estado del Arte

Haciendo un rápido barrido por la red, es posible encontrar software de recomendación de las más diversas características, que varían desde su modo de utilización y lenguaje de programación, hasta la cantidad y tipo de herramientas que proveen.

A continuación se presentan algunas de las aplicaciones y/o bibliotecas analizadas antes de realizar el diseño de nuestra propuesta.

Cabe mencionar, que todas las herramientas presentadas poseen soporte en los tres sistemas operativos de mayor relevancia en la actualidad, es decir, *Linux*, *Windows* y *Mac OS X*.

2.1. My Media Lite

Esta biblioteca implementa una gran cantidad de algoritmos para la predicción de ratings y recomendación de ítems. Para esto no solo es capaz de aplicar técnicas de filtrado colaborativo utilizando información de ratings e *implicit feedback*, sino que también, tiene soporte para el procesamiento de atributos y contenido de datos.

Además, cuenta con la capacidad de calcular métricas como MAE, RMSE, prec @ N, MAP y NDCG.

Sin embargo, para utilizar la biblioteca como parte de sistemas más complejos, es necesario programar en lenguajes como C#, Clojure o F#, lo cual puede ser visto como un punto en contra, si lo que se desea es el análisis expedito de datos, invirtiendo el menor tiempo posible en tareas de desarrollo.

2.2. Lenskit

Otra alternativa que ofrece una batería muy completa de métodos, es la biblioteca *Lenskit*. Esta proporciona todos los algoritmos básico para predicción de ratings, como *User* e *Item KNN*, *Slope One* y factorización matricial utilizando el método *FunkSVD*.

Sin embargo, a diferencia de *My Media Lite*, no tiene soporte para analizar datos del tipo *implicit feedback*, y además, el lenguaje de programación utilizado es Java, lo cual podría disminuir el rendimiento de los algoritmos desde el punto de vista de recursos de computación.

2.3. LibRec

Al igual que las anteriores, esta biblioteca es muy poderosa en cuanto a los algoritmos que tiene disponibles,y la cantidad de métricas que es capaz de calcular.

Sus falencias se encuentran asociadas a la poca facilidad de uso, su pobre documentación y a que su implementación está realizada completamente en Java.

2.4. Lightfm

Lightfm proporciona mecanismos para recomendación basada en *implicit* y *explicit feedback*, utilizando para ello, algoritmos de factorización matricial.

Además, posee una interfaz en *Python*, lo cual la convierte inmediatamente en una herramienta amistosa y de fácil manejo.

Debido a que su especialización está dada por el lado de la factorización matricial, no cuenta con los métodos básicos de recomendación.

2.5. Mrec

Esta es una segunda buena alternativa si lo que se desea es programar un sistema sencillo basado en *Python*.

Pero una vez más, nos encontramos con la ausencia de métodos tradicionales como *User KNN* y *Slope One*.

2.6. Rrecsys

Esta es una biblioteca implementada en lenguaje *R*, de gran simplicidad en cuanto a su utilización, lo que la hace muy cómoda para la realización de pruebas en el ámbito académico.

Sin embargo, no es muy completa en cuanto a la cantidad de algoritmos implementados, y posee una fuerte limitación en cuanto a la magnitud de los datos que se pueden analizar, debido a que no soporta matrices ralas.

3. SOLUCIÓN

La motivación principal que nos llevó a tomar la decisión de implementar *Pyreclab*, fue la carencia de una biblioteca que reuniera tres características, que a nuestro juicio, son fundamentales en una herramienta de manejo de grandes cantidades de datos. Estas son: 1) Implementación de estructuras de datos eficientes en un lenguaje de bajo nivel, que no sobrecarguen los recursos de computación en tareas que son ajenas a los algoritmos de recomendación. 2) Interfaz amigable que acelere el proceso de desarrollo y pruebas, orientado al estudio de datos y análisis de sensibilidad de parámetros. 3) Base completa de algoritmos tradicionales, y con una arquitectura que permita aumentar la batería de métodos de forma rápida y sencilla.

Para responder a estos requerimientos, se plantea el desarrollo de un módulo basado en los siguientes puntos:

- Para lograr simplicidad de uso, se define una interfaz de usuario en lenguaje *Python*, ya que esto facilitará los aspectos de implementación y permitirá la realización de pruebas por línea de comando.
- Adicionalmente, se implementan los mecanismos necesarios para generar dos sabores de la biblioteca, correspondientes a las versiones 2 y 3 de *Python*.
- Implementación eficiente y escalable: Todo el módulo está implementado de forma íntegra en C++, lo cual otorga mayores garantías de rendimiento en comparación con otros lenguajes.
- Soporte para los algoritmos de recomendación tradicionales: se han implementado los algoritmos *User KNN*, *Item KNN*, *Slope One*, *Item Average*, *Most Popular* y *Funk SVD*, y se deja la posibilidad abierta, de seguir incorporando otros métodos.
- Facilidad para realización de análisis de sensibilidad, debido al acceso de los parámetros de cada algoritmo, a través de la modificación de lor argumentos de los métodos ejecutados.
- Capacidad para el cálculo de predicciones de ratings y recomendaciones de ítems por ranking, para un usuario en particular, y para un grupo de usuarios contenidos en un set de test.
- Soporte para flexible de formatos de entrada y salida: csv, json, texto plano.

3.1. Diseño general.

La figura 1, muestra el diseño general de la arquitectura de la biblioteca *Pyreclab*, y los principales módulos que la componen.

Tal como puede verse de la parte inferior, el bloque azul representa el interprete *Python*, el cual tendrá la misión de cargar los métodos y estructuras de datos, al importarse el módulo *pyreclab*. De este modo, el interprete tendrá a

disponible todo el conjunto de algoritmos desarrollados para la realización de recomendaciones.

En la parte superior, en color naranja, aparecen todos los sub-módulos que componen la biblioteca, y cuyas tareas, son descritas a continuación.

3.1.1 File IO

Este componente representa el punto de entrada y salida de datos, ya que permite la lectura de archivos que contienen el conjunto de entrenamiento y/o prueba, además de la escritura de predicciones de ratings y/o rankings, si así se desea.

Este otorga un buen nivel de flexibilidad en cuando a formatos de textos, debido a que sus campos pueden estar separados por cualquier tipo de carácter, el cual debe ser especificado por el usuario a través de la interfaz de alto nivel.

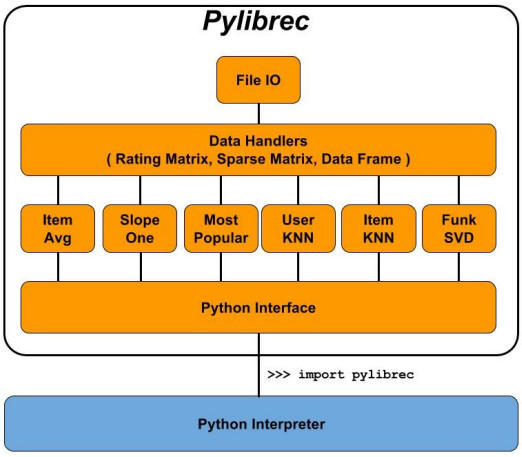

Figura 1. Arquitectura y componentes de biblioteca *Pyreclab*.

3.1.2 Data handlers

Este módulo contiene una serie de estructuras de datos, que permiten tener un acceso homogéneo a los valores de ratings, otorgando un mayor nivel de independencia del formato original en que fueron leídos, y con un mayor nivel de abstracción. Estas serán directamente utilizadas por los algoritmos de recomendación para el procesamiento, almacenamiento y generación de datos.

3.1.3 Algoritmos de recomendación

Bajo el bloque *Data Handlers*, se encuentra una serie de bloques contiguos que representan a cada uno de los algoritmos de recomendación implementados, y que están disponibles para la predicción de ratings y/o recomendación de ratings.

La lista de algoritmos disponibles para la predicción de ratings son: *Item Average*, *Slope One*, *User KNN*, *Item KNN* y *Funk SVD*. Por otra parte, para la generación de recomendaciones a través de ranking se cuenta por el momento, solo con el método *Most Popular*.

3.1.4 Python Interface

Este módulo representa la interfaz de comunicación entre los algoritmos de recomendación, y el intérprete *Python*. Al igual que el resto de la biblioteca, esta interfaz fue desarrollada completamente en C++, para lo cual fue necesario evaluar una serie de alternativas que permitieran realizar esta conexión entre lenguajes, pero manteniendo por sobre todo, un nivel aceptable de legibilidad del código.

Por esta razón, se decidió utilizar la *API Python/C*, la cual viene incluida en el paquete estándar de *Python*. Esta permite definir estructuras de bajo nivel en lenguaje C, las que tienen un mapeo directo con los objetos manejados por el intérprete de *Python*. Con ella se podrán definir estructuras nativas de *Python* como listas, diccionarios y tuplas, así como también, clases de mayor complejidad para permitir el manejo y creación de todo tipo de datos abstractos.

Gracias a esto, se ha definido un tipo de dato por cada uno de los algoritmos de recomendación, los cuales podrán ser instanciados directamente desde el intérprete de *Python*, y permitirán controlar el comportamiento de cada algoritmo a través de los parámetros a los que se ha dado acceso.

A continuación, se detalla la interfaz que tiene el usuario a su disposición.

3.2. Detalles de implementación.

Las clases contenidas por el módulo *pyreclab*, son listadas a continuación.

- `pyreclab.UserAvg`
- `pyreclab.ItemAvg`
- `pyreclab.SlopeOne`
- `pyreclab.MostPopular`
- `pyreclab.UserKnn`
- `pyreclab.ItemKnn`
- `pyreclab.SVD`

Al ser instanciadas, cada una de ellas debe recibir como parámetro obligatorio, el archivo que contiene los datos de entrenamiento. Además, se pueden establecer otros parámetros que tienen relación con el carácter delimitador de campos en el archivo, existencia de cabecera informativa y posición de columnas correspondientes a usuario, ítem y rating. A continuación se presenta un ejemplo de dicha situación.

```
obj = pyreclab.RecAlg( 'dataset/u1.base',
                       dlmchar = b'\t',
                       header = False,
                       usercol = 0,
                       itemcol = 1,
                       ratingcol = 2 )
```

Una vez instanciada la clase *obj* para la ejecución del algoritmo, se tendrá acceso a los parámetros particulares de cada uno de ellos según corresponda.

3.2.1 User Average

Este algoritmo no posee parámetros especiales para el entrenamiento del modelo, por lo que su método *train()* no recibe argumentos.

```
obj.train()
```

Una vez entrenado el modelo, el método train() retornará para poder realizar tareas de predicción, ya sea en un usuario e ítem particular, como en un archivo con datos de prueba.

Para ello se cuenta con los métodos *predict()* y *test()* respectivamente, los cuales son utilizados como se muestra a continuación.

```
obj.predict( userId, itemId )
```

Este método recibe como parámetros, los identificadores de usuario e ítem, y retornará como resultado, el rating predicho por el modelo.

Por otra parte, el método test() debe recibir como parámetro obligatorio, el archivo que contiene el set de datos de prueba. Además, posee parámetros opcionales que permiten dar información extra sobre la estructura del archivo leído, tal como se planteó para el caso del constructor de cada clase.

```
obj.test( 'dataset/u1.test',
          dlmchar = b'\t',
          header = False,
          usercol = 0,
          itemcol = 1,
          output_file = 'predictions.csv' )
```

Como resultado, este método retornará una lista de tuplas, cuyo contenido serán los siguientes: userId, itemId, predicción.

Adicionalmente, se da la alternativa de entregar un nombre de archivo en el parámetro output_file, para que las predicciones no solo sean retornadas, sino que también, sean escritas en el archivo indicado.

3.2.2 Item Average

Al igual que en el caso anterior, este algoritmo no posee parámetros especiales para el entrenamiento del modelo, por lo que su método *train()* no recibe argumentos.

```
obj.train()
```

Una vez entrenado el modelo, el método train() retornará para poder realizar tareas de predicción, ya sea en un usuario e ítem particular, como en un archivo con datos de prueba.

Para ello se cuenta con los métodos *predict()* y *test()* respectivamente, los cuales son utilizados como se muestra a continuación.

```
obj.predict( userId, itemId )
```

Este método recibe como parámetros, los identificadores de usuario e item, y retornará como resultado, el rating predicho por el modelo.

Por otra parte, el método test() debe recibir como parámetro obligatorio, el archivo que contiene el set de datos de prueba. Además, posee parámetros opcionales que permiten dar información extra sobre la estructura del archivo leído, tal como se planteó para el caso del constructor de cada clase.

```
obj.test( 'dataset/u1.test',
          dlmchar = b'\t',
          header = False,
          usercol = 0,
          itemcol = 1,
          output_file = 'predictions.csv' )
```

Como resultado, este método retornará una lista de tuplas, cuyo contenido serán los siguientes: userId, itemId, predicción.

Adicionalmente, se da la alternativa de entregar un nombre de archivo en el parámetro output_file, para que las predicciones no solo sean retornadas, sino que también, sean escritas en el archivo indicado.

3.2.3 Slope One

A continuación se presenta la firma de sus métodos.

```
obj.train()

obj.predict( userId, itemId )

obj.test( 'dataset/u1.test',
          dlmchar = b'\t',
          header = False,
          usercol = 0,
          itemcol = 1,
          output_file = 'predictions.csv' )
```

3.2.4 User KNN

A diferencia de los métodos *train()* vistos anteriormente, para el caso de *User KNN*, es posible entregar un parámetro opcional de entrada, el cual corresponde al número de vecinos cercanos con que será construido el modelo.

```
obj.train( knn )

obj.predict( userId, itemId )


obj.test( 'dataset/u1.test',
          dlmchar = b'\t',
          header = False,
          usercol = 0,
          itemcol = 1,
          output_file = 'predictions.csv' )
```

3.2.5 Item KNN

A continuación se presenta la firma de sus métodos.

```
obj.train( knn )

obj.predict( userId, itemId )

obj.test( 'dataset/u1.test',
          dlmchar = b'\t',
          header = False,
          usercol = 0,
          itemcol = 1,
          output_file = 'predictions.csv' )
```

3.2.6 Funk SVD

De todos los algoritmos, este es quizás el que mayor cantidad de parámetros de entrenamiento posee, tal como se ve a continuación.

```
obj.train( factors = 1000, maxiter = 100, lr = 0.01, lamb = 0.1 )
```

Estos parámetros corresponden a en orden a:

- Número de factores latentes.
- Número máximo de iteraciones antes de detener el algoritmo de descenso del gradiente estocástico.
- Taza de aprendizaje.
- Parámetro lambda del regularizador.

El resto de los métodos, tiene el mismo comportamiento visto hasta ahora en el resto de los algoritmos.

```
obj.predict( userId, itemId )

obj.test( 'dataset/u1.test',
          dlmchar = b'\t',
          header = False,
          usercol = 0,
          itemcol = 1,
          output_file = 'predictions.csv' )
```

3.2.7 Most Popular

Esta clase posee un grado mayor de diferencia con respecto al resto, debido a que el algoritmo que implementa está pensado para la generación de rankings de ítems según su popularidad, y su forma pura no tiene una interpretación a nivel de ratings.

Por esta razón, al instanciar este objeto, ya no contaremos con el método *predict()*, sino que en su reemplazo, se tendrá el método *recommned()*, el cual entregará recomendaciones de ítems según la construcción del ranking.

Según esto, el método *train()* recibirá como parámetro de entrada, el número de ítems que se desea compongan el ranking *topN*.

```
obj.train( topN )

obj.recommend( userId )
obj.test( 'dataset/u1.test',
          dlmchar = b'\t',
          header = False,
          usercol = 0,
          output_file = 'ranking.json' )
```

4. DATASET

Para evaluar el presente proyecto, se utilizará el conjunto de datos denominado *ml-100k*, el cual consiste en un conjunto de ratings de películas, los cuales fueron recolectados durante un período de siete mese desde el sitio web de *MovieLens*. A continuación se listan algunas de sus principales características:

- 100.000 ratings en escala de 1 a 5, de 943 usuarios sobre 1682 películas (ítems).
- Cada usuario ha evaluado al menos 20 películas.
- Contiene información demográfica de los usuarios: edad, género, ocupación y código postal.
- Contiene descripción completa de películas: título, fecha de lanzamiento, URL *IMDb* y género.

A continuación se presenta el análisis estadísticos realizado sobre el conjuntos de datos de entrenamiento, que serán utilizados para el establecimiento de modelos de recomendación. La siguiente tabla, resume los principales datos obtenidos del conjunto de datos de entrenamiento para predicciones de ratings.

Tabla 1.- Información estadística del conjunto de datos de entrenamiento.

Parámetro	Valor
Usuarios	943
Ítems	1.682
Ratings	100.000
Min. rating	1
Max. rating	5
Paso entre ratings	1

Densidad	0.06304732
Sparsity	0.9369527
Promedio de ratings / usr.	106.05 ± 100.93
Rating promedio / usr.	3.59 ± 0.45
Máx. ratings / usr.	737
Usuario con mas ratings	405
Máx. rating / item	583
Ítem con más ratings	50 (*Star Wars*)

Además de obtener estos datos estadísticos, gracias a la información adicional provista por el dataset, es posible complementar el análisis con información cualitativa. Como por ejemplo, para el caso del identificador de ítem con más ratings, fue posible saber que corresponde a la película *Star Wars*.

Para complementar estos datos, la siguiente figura muestra como se distribuyen los usuarios, en función del número de ítems evaluados por ellos.

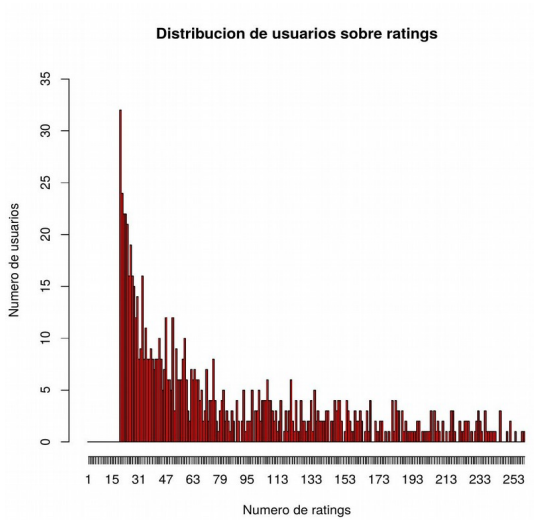

Figura 2. Distribución de usuarios sobre ratings del conjunto de datos de entrenamiento.

Al observar la figura 2, se aprecia un tipo de distribución característica para estas variables. Aquí se puede ver que la mayoría de los usuarios realiza pocas evaluaciones, concentrándose estos en las cantidades más bajas de número de ratings.

Además, tal como se señala en la descripción del dataset, se comprueba que todos los usuarios con menos de 20 ratings han sido eliminados del conjunto.

Por otra parte, la figura 3 presenta la distribución de ratings en función de los posibles valores que pueden ser asignados a cada ítem.

De ella se desprende que existe una alta concentración de votos con valores 3 y 4, lo cual es coincidente con el valor medio de promedio de ratings por usuarios, el cual tiene un valor de 3.58 como se presenta en la tabla 1.

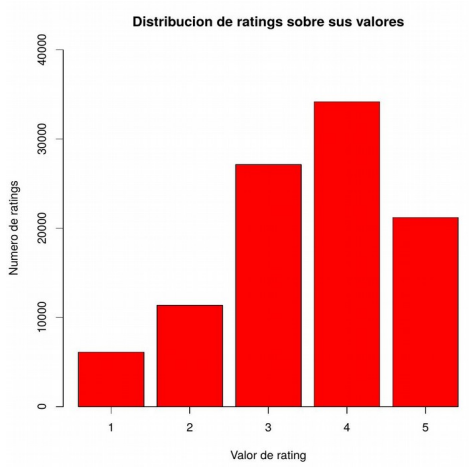

Figura 3. Distribución de ratings sobre sus valores posibles del conjunto de datos de entrenamiento.

5. METODOLOGÍA

Para evaluar la implementación de *pyreclab*, se realizan dos tipos de comparaciones con la biblioteca *LibRec*, mencionada en la sección 2.

La primera de ellas corresponde a una evaluación de las predicciones de ratings generadas por nuestra biblioteca. Para ello, se toma una de los 5 particiones proporcionadas por el dataset de *MovieLens*, y se ejecutan los mismos algoritmos en igualdad de condiciones, tanto de parámetros, como de datos de prueba. Luego de obtener los modelos entrenados, es posible calcular las métricas de evaluación MAE y MRSE sobre el conjunto de datos de prueba, para posteriormente, comparar dichos valores con los obtenidos por *LibRec*.

Adicionalmente, se realizaron pruebas comparativas en cuanto al tiempo que toman los procedimientos de entrenamiento y prueba en cada una de las herramienta, con el fin de tener una línea base que permita evaluar la calidad del desarrollo, y nos muestre si fue posible alcanzar uno de los objetivos principales a lograr, el cual consistía en obtener una biblioteca eficiente.

6. RESULTADOS

Para comenzar, se presentan los resultados obtenidos al calcular las métricas de evaluación implementadas, sobre los datos de test.

En la siguiente tabla es posible ver que los valores obtenidos por cada biblioteca son similares para cada método, y por lo tanto, es posible validar la capacidad predictiva de la presente implementación.

Algoritmo	MAE		RMSE	
	pyreclab	LibRec	pyreclab	LibRec
UserAvg	0.850191	0.850191	1.062995	1.062995
ItemAvg	0.827568	0.827568	1.033411	1.033411
SlopeOne	0.748552	0.748299	0.952795	0.952460
User KNN	0.754816	0.755361	0.962355	0.966395
Item KNN	0.749316	0.748354	0.953637	0.953433
Funk SVD	0.732820	0.731986	0.925390	0.923978

A continuación se presentan los resultados obtenidos al medir los tiempos que demora cada herramienta de software en su proceso de entrenamiento y pruebas.

Tal como podrá verse de las gráficas y tablas, los resultados temporales obtenidos resultan ser desfavorables en la gran mayoría de los casos, ya que el tiempo que demora *Pyreclab* en realizar sus operaciones es casi siempre mayor que el tiempo que le toma a *LibRec* realizar la misma tarea.

Sin embargo, hay que destacar que los tiempos obtenidos al utilizar el algoritmo de *FunkSVD*, son similares a los de *LibRec* en etapa de entrenamiento, y mucho mejores que *LibRec*, en etapa de prueba, lo cual puede apreciarse en las figuras 8 y 9.

A continuación, se presenta todo el detalle de los tiempos obtenidos para cada algoritmo evaluado, en donde también se realizaron variaciones de parámetros en los casos donde ellos eran de gran relevancia para el método en cuestión.

Para todos las gráfica, *Pyreclab* es representada con la línea azul, y *LibRec*, con la roja.

6.1 Item Average

Train		Test	
Pyreclab	LibRec	Pyreclab	LibRec
0,366876	0,049	0,111922	0,658

6.2 Slope One

Train		Test	
Pyreclab	LibRec	Pyreclab	LibRec
8,642671	0,525	1,376621	0,805

6.3 User KNN

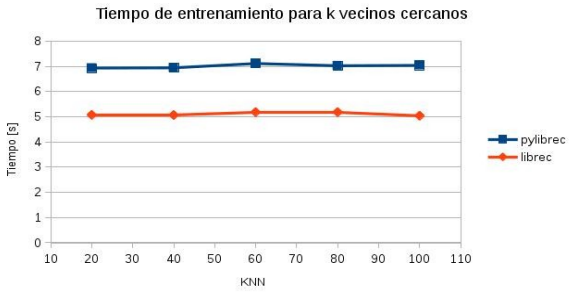

Figura 4. Comparación entre tiempos de entrenamiento de Pyreclab y LibRec usando User KNN.

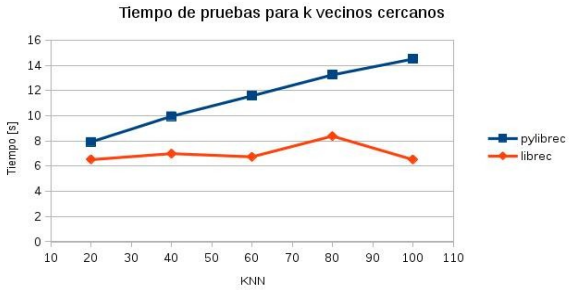

Figura 5. Comparación entre tiempos de pruebas de Pyreclab y LibRec usando User KNN.

6.4 Item KNN

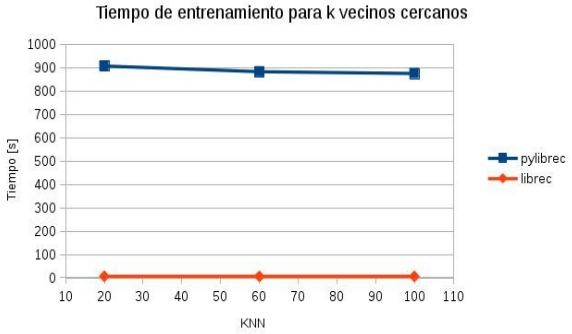

Figura 6. Comparación entre tiempos de entrenamiento de Pyreclab y LibRec usando Item KNN.

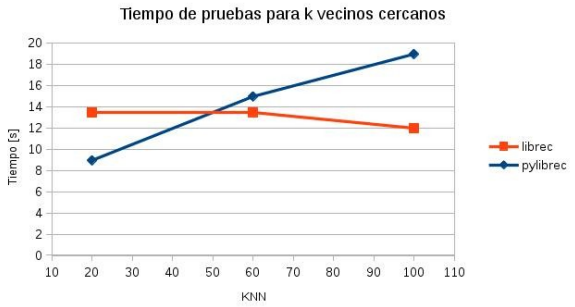

Figura 7. Comparación entre tiempos de pruebas de Pyreclab y LibRec usando Item KNN.

6.5 Funk SVD

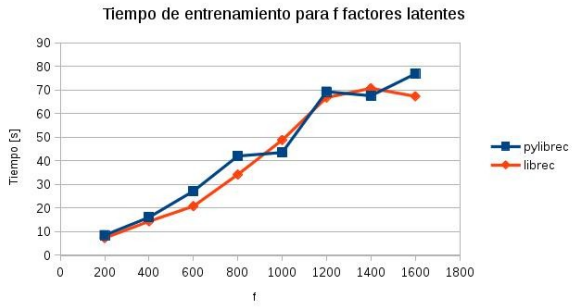

Figura 8. Comparación entre tiempos de pruebas de Pyreclab y LibRec usando Funk SVD.

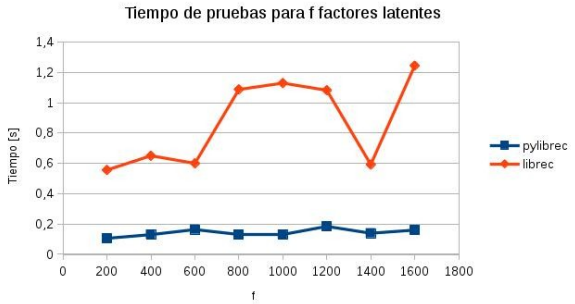

Figura 9. Comparación entre tiempos de pruebas de Pyreclab y LibRec usando Funk SVD.

7. CONCLUSIONES

El logro de sistemas de procesamiento eficiente requiere de un fuerte análisis de cada una de las estructuras de datos involucradas en el proceso, debido a que tanto los tiempos de acceso como escritura de datos, comienzan a incidir de forma importante cuando su número su manejo es en grandes cantidades.

En el caso particular de los sistemas de recomendación basados en ratings, se tiene un desafío adicional que tiene directa relación con el manejo de matrices ralas. Aún cuando existen estructuras capaces de manejar de forma eficiente estos datos, y sin provocar un sobre consumo de memoria, las distintas técnicas elaboradas no son siempre las mejores en todas las situaciones. Un claro ejemplo de ello son las dos metodologías para almacenar datos de una matriz en memoria, una posicionando datos de forma contigua recorriendo las filas, y el otro recorriendo las columnas. Aquí claramente habrá que sopesar cual de los dos métodos es más conveniente según que tipo de recorrido o acceso es más frecuente para nuestra aplicación.

Uno de los resultados más importantes obtenidos, se presentó durante la fase iniciales del proyecto, en la cual se estudiaba el beneficio de utilizar C++ por sobre otros lenguajes. En uno de los experimentos, se programó un algoritmo para calcular una integral aproximada por rectángulos. Este fue codificado en dos versiones, una en *Python* puro, y la otra, en C++ pero cuya ejecución se realizó desde *Python*. Con esto se pudo demostrar, que para iguales operaciones de punto flotante, C++ demoraba 3 veces menos en terminar la operación en comparación con el tiempo que tomó el intérprete de *Python*. Esto era de esperarse al tener presentes las diferencias entre la ejecución de un código compilado, que ejecuta instrucciones directamente sobre la CPU, y entre otro cuyas operaciones pasan por otra entidad de software que debe interpretar cada instrucción.

Finalmente, durante el estudio y desarrollo quedaron varias ideas para seguir mejorando esta biblioteca, y marcar diferencias con las que comúnmente utilizadas. Una de ellas es la aplicación de técnicas de búsqueda aproximada de vecinos cercanos, basadas en estructuras de árboles. Según algunas pruebas preliminares, la ganancia en tiempo experimentada, en comparación con la pérdida de precisión de los vecinos encontrados, es sobresaliente. Es de extrañarse la no inclusión de dichas técnicas en las bibliotecas de recomendación analizadas.

8. REFERENCIAS

[1] Said, A., & Bellogín, A. (2014, October). *Comparative recommender system evaluation: benchmarking recommendation frameworks*. In Proceedings of the 8th ACM Conference on Recommender systems (pp. 129-136). ACM.

[2] Zeno Gantner and Steffen Rendle and Christoph Freudenthaler and Lars Schmidt-Thieme, *MyMediaLite: A Free Recommender System Library*, Proceedings of the 5th ACM Conference on Recommender Systems (RecSys 2011)

[3] Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer IEEE Magazine, 42(8), 30-37.

[4] Schafer, J. B., Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative filtering recommender systems. In The

adaptive web (pp. 291-324). Springer Berlin Heidelberg.
[5] *MyMediaLite Recommender System Library*, Disponible en la Web: http://www.mymedialite.net/
[6] *Lenskit, Open-Source Tools for Recommneder Systems*, Disponible en la Web: http://lenskit.org/
[7] *LibRec, A Java Library for Recommender Systems*, Disponible en la Web: http://www.librec.net/
[8] *LightFM*, Disponible en la Web: https://github.com/lyst/lightfm
[9] *Mrec recommender systems library*, Disponible en la Web: https://github.com/Mendeley/mrec
[10] *Rrecsys: Environment for Assessing Recommender Systems*, Disponible en la Web: https://cran.r-project.org/web/packages/rrecsys/index.html
[11] *Recommenderlab: Lab for Developing and Testing Recommender Algorithms*, Disponible en la Web: https://cran.r-project.org/web/packages/recommenderlab/index.html