

**ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO**

**MÁSTER UNIVERITARIO EN BIG DATA ANALYTICS**

TRABAJO FIN DE MÁSTER

**SISTEMA RECOMENDADOR DE LIBROS**

**JAIME NOÉ BENITO CASADO**

**CURSO 2018-2019**

**TÍTULO:** SISTEMA RECOMENDADOR DE LIBROS

**AUTOR**: JAIME NOÉ BENITO CASADO

**TITULACIÓN**: MÁSTER UNIVERSITARIO EN BIG DATA ANALYTICS

**DIRECTOR DEL PROYECTO**: CARLOS FERNANDEZ MUSOLES

**FECHA**: OCTUBRE de 2019

RESUMEN

El presente documento realiza un breve estudio de los sistemas de recomendación, así como su aplicación práctica para un dataset de libros.

Se ilustrará el proceso empleado para el desarrollo de dicho sistema de recomendación teniendo en cuenta no sólo las métricas clásicas de aprendizaje automático tales como RMSE o MAE, sino métricas típicas de estos sistemas recomendadores tales como el Hit-Rate, Average Reciprocal Hit Ranking, la cobertura o la diversidad.

Los sistemas de recomendación producen matrices de usuarios / ratings / ítems (libros en nuestro caso) muy dispersas, por lo que es buena idea utilizar técnicas de reducción de la dimensionalidad como el PCA (Principal Componente Analysis) o mediante featuring engineering. Esto hace que el cálculo de algoritmos utilizados para la mayoría de los sistemas recomendadores requiera de gran potencia computacional incluso para datasets de tamaño pequeño o mediano. Esto se puede solucionar mediante el uso de tecnologías de Big Data, donde los datos y la potencia computacional se distribuyen a lo largo de distintos clústers, para lo cual utilizaremos Databricks con Spark sobre la API de Python (Pyspark).

Para la implementación de este sistema recomendador nos basaremos en un dataset con usuarios-libros-ratings ya existente el cual limpiaremos y modificaremos para adecuarlo a nuestras necesidades mediante un proceso de feature engineering en el que nos quedaremos solamente con aquellas características (features) que son importante para el cálculo de la predicción de un determinado rating mediante el llamado EDA (Exploratory Data Analysis). Después de efectuar dicho análisis hemos podido comprobar como la lista “género” a los que un libro puede pertenecer es la característica más importante a la hora de recomendar libros similares a usuarios parecidos.

Una vez tengamos esto listo probaremos diferentes algoritmos de recomendación tales como el Item-based Collaborative Filtering, User-based Collaborative Filtering, SVD, SVD++ e incluso redes neuronales (RBMs – Restricted Boltzman Machines) evaluando cada una de las métricas más relevantes con el fin de seleccionar el algoritmo con mejor poder predictor en función del dataset transformado, que en caso aquí estudiado resultará en el SVD y por lo tanto que será el encargado de generar listas de las Top-N recomendaciones para cada usuario.

Finalmente realizaremos varias predicciones para usuarios reales con el fin de demostrar la adecuación de este sistema recomendadores de libros en el mundo real, paso previo necesario antes de la implementación de un caso de negocio real.

**Palabras clave:** Sistemas de recomendación, sistemas colaborativos basados en contenidos, sistemas colaborativos por usuario, KNN, SVD, DeepFM

ABSTRACT

This document makes a brief study of recommendation systems, as well as their practical application for a book dataset.

The process used for the development of such a recommendation system will be illustrated considering not only “classic” machine learning metrics such as RMSE or MAE, but also typical metrics of these recommendation systems such as Hit-Rate, Average Reciprocal Hit Ranking, coverage or diversity.

The recommendation systems produce very sparse matrixes of users / ratings / items (books in our case), so it is a good idea to use dimensionality reduction techniques such as PCA (Principal Component Analysis) or featuring engineering. This means that the calculation of algorithms used for most recommendation systems requires high computing power even for small or medium sized datasets. This can be solved using Big Data technologies, where data and computational power are distributed over different clusters, for which we will use Databricks with Spark on the Python API (Pyspark).

For the implementation of this recommended system we will rely on a dataset with existing user-books-ratings which we will clean and modify to suit our needs through a feature engineering process in which we will stay only with those features (features) that are important for the calculation of the prediction of a given rating through the so-called EDA (Exploratory Data Analysis). After carrying out this analysis we have been able to check how the list "genre" to which a book can belong is the most important feature when recommending similar books to similar users.

Once we have this ready we will try different recommendation algorithms such as Item-based Collaborative Filtering, User-based Collaborative Filtering, SVD, SVD++ and even neuronal networks (RBMs - Restricted Boltzman Machines) evaluating each of the most relevant metrics in order to select the algorithm with the best predictive power depending on the dataset transformed, which in the case studied here will result in the SVD and therefore will be responsible for generating lists of Top-N recommendations for each user.

Finally, we will make several predictions for real users in order to demonstrate the adequacy of this system book recommenders in the real world, a necessary preliminary step before the implementation of a real business case.

**Key words**: Recommender systems, user-based collaborative systems, item-based collaborative systems, KNN, SVD, DeepFM

**AGRADECIMIENTOS**

A mi tutor Carlos por el tiempo que me has dedicado y por todo lo que he aprendido de ti, tanto como alumno en tus clases como durante la realización del proyecto.

A mis compañeros de prácticas a lo largo del máster: mi hermano Enrique y a Guillermo por su ayuda y colaboración en todo momento que me han servido para aprender técnicamente como distintos métodos de trabajo.

Al resto de compañeros del máster por su ayuda y consejo: Jesús, José Ignacio, Jorge, Justo, Mikel, Belén y José.

A mi familia por su apoyo durante las etapas de mayor carga de trabajo durante el máster.

Índice

[1. Introducción y objetivos 12](#_Toc21936529)

[1.1 Introducción 12](#_Toc21936530)

[1.2 Planteamiento del problema 13](#_Toc21936531)

[1.3 Objetivos 14](#_Toc21936532)

[1.4 Hipótesis 15](#_Toc21936533)

[1.5 Fases del desarrollo 15](#_Toc21936534)

[2. Estado del arte 16](#_Toc21936535)

[2.1 Introducción 16](#_Toc21936536)

[2.2 Evaluación sistema recomendador 16](#_Toc21936537)

[2.2.1 Medidas de error 17](#_Toc21936542)

[2.2.1.1 MAE (Mean Absolute Error) 17](#_Toc21936547)

[2.2.1.2 RMSE (Root Mean Square Error) 18](#_Toc21936548)

[2.2.1.3 Hit Rate 19](#_Toc21936549)

[2.2.1.4 ARHR (Average Reciprocal Hit Rate) 19](#_Toc21936550)

[2.2.1.5 CHR (Cummulative Hit Rate) 20](#_Toc21936551)

[2.2.1.6 RHR (Rating Hit Rate) 20](#_Toc21936552)

[2.2.1.7 Cobertura 20](#_Toc21936553)

[2.2.1.8 Diversidad 21](#_Toc21936554)

[2.2.1.9 Novedad 21](#_Toc21936555)

[2.2.1.10 Churn 22](#_Toc21936556)

[2.2.1.11 Responsividad 23](#_Toc21936557)

[2.2.2 Evaluación final 23](#_Toc21936558)

[2.3 Algoritmos para sistemas de recomendación 24](#_Toc21936559)

[2.3.1 Introducción 24](#_Toc21936560)

[2.3.2 IBCF (Item-Based Collaborative Filtering) 24](#_Toc21936561)

[2.3.3 KNN – K Nearest Neighbors 27](#_Toc21936562)

[2.3.4 UBCF (User-Based Collaborative Filtering) 28](#_Toc21936563)

[2.3.5 PCA 30](#_Toc21936564)

[2.3.5.1 SVD 31](#_Toc21936565)

[2.4 Deep Learning en sistemas de recomendación 33](#_Toc21936566)

[2.4.1 Introducción 33](#_Toc21936567)

[2.4.2 RBM (Restricted Boltzmann Machines) 34](#_Toc21936568)

[2.4.2.1 Introducción 34](#_Toc21936569)

[2.4.2.2 RBMs condicionales 35](#_Toc21936570)

[2.4.2.3 RBMS condicionales factorizadas 36](#_Toc21936571)

[2.4.3 Deep FM 36](#_Toc21936572)

[3. Prueba de concepto 39](#_Toc21936573)

[3.1 Introducción 39](#_Toc21936574)

[3.2 EDA: Análisis exploratorio de datos 40](#_Toc21936575)

[3.2.1 Introducción 40](#_Toc21936579)

[3.2.2 Análisis y limpieza de datos 42](#_Toc21936580)

[3.2.2.1 Diferentes idiomas 42](#_Toc21936581)

[3.2.2.2 Distribución de los géneros 43](#_Toc21936582)

[3.2.2.3 Distribución de la media de ratings por usuario 43](#_Toc21936583)

[3.2.2.4 Número de ratings por usuario 44](#_Toc21936584)

[3.2.2.5 Variables independientes predictoras 45](#_Toc21936585)

[3.2.2.6 Top 10 libros más votados 45](#_Toc21936586)

[3.2.2.7 Correlación del rating con el número de ratings 45](#_Toc21936587)

[3.2.2.8 Correlación del rating con el género 46](#_Toc21936588)

[3.2.1.9 Correlación del rating con el año de publicación 47](#_Toc21936600)

[3.2.2 Resultados 48](#_Toc21936601)

[3.3 Evaluación de algoritmos clásicos 49](#_Toc21936602)

[3.3.1 Introducción 49](#_Toc21936603)

[3.3.2 Resultados y conclusiones 50](#_Toc21936604)

[3.4 Evaluación de Deep Learning 52](#_Toc21936605)

[3.4.1 Introducción 52](#_Toc21936606)

[3.4.2 Resultados RBMs 52](#_Toc21936607)

[3.5 Big Data 54](#_Toc21936608)

[3.5.1 Introducción 54](#_Toc21936614)

[3.5.2 Azure Data Lake 55](#_Toc21936615)

[3.5.3 DataBricks 56](#_Toc21936616)

[3.5.3.1 Creación del clúster 56](#_Toc21936617)

[3.5.3.2 Notebooks 57](#_Toc21936618)

[3.5.3.3 Datos 58](#_Toc21936619)

[3.5.3.4 Ejecución y resultados 59](#_Toc21936620)

[3.5.3.5 Conclusiones 62](#_Toc21936621)

[4. Conclusiones 63](#_Toc21936622)

[4.1 Conclusiones y aportaciones 63](#_Toc21936625)

[4.2 Líneas de trabajo futuras 66](#_Toc21936626)

[4.3 Business Case 67](#_Toc21936627)

[5. Presupuesto 70](#_Toc21936628)

[5.1 Introducción 70](#_Toc21936634)

[5.2 Cronograma del plan de trabajo 71](#_Toc21936635)

[5.3 Presupuesto 72](#_Toc21936636)

[Bibliografía 73](#_Toc21936637)

Índice de Figuras

[Figura 1: The Long tail 22](#_Toc21936638)

[Figura 2: Ángulo entre los vectores de Yo Robot y Starship Troopers 26](#_Toc21936639)

[Figura 3: Coseno y seno vs ángulo 26](#_Toc21936640)

[Figura 4: Representación vectorial de libros y ángulos 26](#_Toc21936641)

[Figura 5: Funcionamiento sistema KNN 27](file:///D:\UEM\TFM\Sistema_recomendador_de_libros_JBC.docx#_Toc21936642)

[Figura 6: Ejemplo UBCF 28](#_Toc21936643)

[Figura 7: SVD [9] 32](#_Toc21936644)

[Figura 8: RBM con binary hidden units y softmax visible units [5] 35](#_Toc21936645)

[Figura 9: Arquitectura de una RBM condicional [5] 35](#_Toc21936646)

[Figura 10: Arquitectura DeepFM [9] 37](#_Toc21936647)

[Figura 11: Población, muestra y unidad de análisis [1] 41](#_Toc21936648)

[Figura 12: Distribución del número de libros según el idioma 42](#_Toc21936649)

[Figura 13: Número de libros por género 43](#_Toc21936650)

[Figura 14: Distribución del número de valoraciones según sus ratings stars 44](#_Toc21936651)

[Figura 15: Distribución del número de usuarios según su número de ratings 44](#_Toc21936652)

[Figura 16: Correlación entre features y variable dependiente 45](#_Toc21936653)

[Figura 17: Top 10 libros más votados 45](#_Toc21936654)

[Figura 18: Correlación entre el rating medio y el número de ratings 46](#_Toc21936655)

[Figura 19: Correlación entre el rating y el año de publicación 47](#_Toc21936656)

[Figura 20: Correlación rating y año publicación (últimos 200 años) 48](#_Toc21936657)

[Figura 21: Resultados de predicción de algoritmos de recomendación 50](#_Toc21936658)

[Figura 22: Comparación resultados entre algoritmos de recomendación 51](#_Toc21936659)

[Figura 23: Votaciones reales usuario 43140 51](#_Toc21936660)

[Figura 24: Top-N recomendaciones 52](#_Toc21936661)

[Figura 25: Resultados ejecución RBM 53](#_Toc21936662)

[Figura 26: Resultados RBMs en función de sus hiperparámetros 54](#_Toc21936663)

[Figura 27: Azure DataLake suscription 55](#_Toc21936664)

[Figura 28: Creación clúster en DataBricks 57](#_Toc21936665)

[Figura 29: Screenshot de los notebooks del workspace 58](#_Toc21936666)

[Figura 30: Importación de datos en Databricks 58](#_Toc21936667)

[Figura 31: Tiempos y resultados modelo aleatorio 59](#_Toc21936668)

[Figura 32: Evaluación User-based KNN 60](#_Toc21936669)

[Figura 33: Evaluación Item-based KNN 60](#_Toc21936670)

[Figura 34: Resultados ejecución SVD 61](#_Toc21936671)

[Figura 35: Resultados ejecución SVD++ 61](#_Toc21936672)

[Figura 36: Comparación resultados algoritmos 62](#_Toc21936673)

[Figura 37: Top 10 recomendaciones para usuario de ejemplo utilizando DataBricks 62](#_Toc21936674)

[Figura 38: Creación canal de Youtube 68](#_Toc21936675)

[Figura 39: Costes Servidor alojamiento Google Cloud 69](#_Toc21936676)

[Figura 40: Diagrama plan de trabajo 71](#_Toc21936677)

Índice de Tablas

[Tabla 1: Rating estimado VS real y error 18](#_Toc21936678)

[Tabla 2: Rating estimado vs real y error cuadrático 18](#_Toc21936679)

[Tabla 3: Ranking y Reciprocal ranking 20](#_Toc21936680)

[Tabla 4: Ranking y Ranking predicho y valores a eliminar según el CHR 20](#_Toc21936681)

[Tabla 5: Rating y Rating Hit Rate 20](#_Toc21936682)

[Tabla 6: Ejemplo géneros por libros 25](#_Toc21936683)

[Tabla 7: Libros con géneros como dimensiones 25](#_Toc21936684)

[Tabla 8: Usuarios, Libros y clasificaciones 28](#_Toc21936685)

[Tabla 9: Similaridad del coseno entre usuarios 29](#_Toc21936686)

[Tabla 10: Precio por Byte Azure Data Lake \* 55](#_Toc21936687)

[Tabla 11: Precios DataBricks 57](#_Toc21936688)

Capítulo 1

# Introducción y objetivos

## Introducción

A lo largo de este Trabajo de Fin de Máster (TFM) se va a analizar el funcionamiento de un sistema de recomendación de libros, utilizando para ello distintos algoritmos de recomendación (Item-based collaborative filtering, User-based collaborative filtering, Singular Value Decomposition o SVD++), así como una pequeña introducción al uso de redes neuronales para generar estas recomendaciones (Restricted Boltzman Machines).

Nos centraremos no sólo en las métricas de error “clásicas” de aprendizaje automático tales como el RMSE, sino que utilizaremos otras más particulares de los sistemas de recomendación, tales como el “Hit Rate”, el Churn o la Novedad.

Se utilizará un dataset completo ya generado anteriormente obtenido de Kaggle [4] con miles de valoraciones de libros por usuarios que incluyen toda la información necesaria para el desarrollo de este TFM. De esta forma no se necesitará recolectar los datos per-se sino simplemente de adecuarlos al entorno de trabajo (mediante una limpieza de éstos, tratamiento de valores faltantes, valores anómalos, etc.)

## Planteamiento del problema

Los sistemas de recomendación de ítems / contenido son una fuente muy importante de los ingresos de grandes compañías como Amazon o Netflix.

Usando este tipo de sistemas han surgidos varias pequeñas compañías centradas en la recomendación de películas (como FilmAffinity® o IMDB ®). Sin embargo, no existe o es muy difícil encontrar páginas y/o compañías centradas en la recomendación de libros, por lo que al tratarse de un tema novedoso será éste el que se lleve a cabo durante este TFM.

Cabe preguntarse el motivo por el motivo por el que no existe una página que se dedique exclusivamente a la recomendación de libros per-se, y si bien hay compañías como “Good reads®” proporcionan algún tipo de recomendación de libros, tales como “¿Qué libro leer en verano?” basándose en los Top-sellers actuales o los libros mejor valorados para cada género, pero son recomendaciones a nivel general y no a nivel de usuario (lo que le guste al usuario “B “puede no tener nada que ver con los gustos del usuario “A” y viceversa).

Las razones de la no existencia de sistemas recomendadores de libros puede ser varias, y una de ellas podría ser que los usuarios que leen libros no estén interesados en estos sistemas, pues quizás sean usuarios que confían menos en las tecnologías o que prefieran guiarse con la intuición más que por los datos.

Otra pregunta sería si es posible aplicar la misma lógica que la usada en la mayoría de los sistemas de recomendadores de películas simplemente cambiando el conjunto de datos de películas a libros y si aplicar las mismas técnicas sería conveniente o no. Esto podría ser muy útil ya que existe una gran cantidad de literatura dedicada a la recomendación de películas, siendo el “*movie data lens*” uno de los datasets más comunes entre los utilizado en distintas aplicaciones de Data Science y Machine Learning para la enseñanza de sistemas recomendadores.

Una vez concluida la investigación llevada a lo largo de este TFM podremos responder a preguntas tales como cuales son las características más importantes a la hora de recomendar libros, es decir, es el género o el año de publicación importante o no. Adicionalmente veremos que algoritmo es el más adecuado según las métricas que queramos analizar.

Al tratarse la nuestra de una investigación cuantitativa, ésta deberá ser lo más objetiva posible, por lo que los fenómenos observados y/o medidos no deben ser afectados por el investigador (creencias, deseos, tendencias, etc.). Es fácil caer en la tentación de intentar predecir valoraciones de libros basadas en creencias puramente intuitivas del investigador tales como si al usuario “A” le han gustado varios libros de ciencia ficción, entonces le va a gustar el libro “X” porque también es de ciencia ficción, sin tener en cuenta otras variables tales como el año de la publicación (puede que la ciencia ficción del año 1900 no tenga nada que ver con la del 2000), el autor u otros.

Como parte del análisis de los datos se realizará un estudio correlacional de éstos para descubrir aquellas características más relevantes a la hora de predecir el rating de un libro para un determinado usuario. Mediremos el grado de asociación entre variables (autor del libro, el género o el año de publicación, etc.) presuntamente relacionada / correlacionada con la variable objetivo (rating). Esta correlación puede ser positiva o negativa en función de si mejoran o empeoran el rating y también puede ocurrir que no exista relación entre las variables, lo cual indica que fluctúan sin seguir un patrón sistemático entre sí. Por ejemplo, si disponemos de la información de la editorial del libro, esto puede ser algo que no tenga relación con el rating si bien podría tener relación con el autor pues podría ser que un determinado autor siempre trabajase con la misma editorial, de modo que la variable editorial no aportase ninguna información adicional valiosa. Si dos variables están correlacionadas (género del libro y rating, por ejemplo) y se conoce la magnitud de la asociación, es posible predecir, con mayor o menor exactitud, el valor aproximado que tendrá una (rating) al conocer el valor de la otra (género)

En este tipo de estudios, hay que tener en cuenta que, puede darse el caso de que dos variables estén aparentemente relacionadas pero que en realidad no sea así. A esta situación se la denomina correlación espuria (falsa). Por ello, el diseño del estudio es muy importante, ya que se puede asignar asociaciones entre variables cuando en realidad no existe tal relación. Un ejemplo clásico de esta problemática fue un estudio que relacionaba el cáncer de pulmón con el consumo de caramelos de menta: cuantos más caramelos de mentas comiese un sujeto más probabilidad tenía de contraer cáncer, por lo que aparente existía una correlación positiva entre ambas indicando que quizás los caramelos de menta provocasen cáncer. Sin embargo, esta correlación era espuria (falsa) ya que le hecho de comer más caramelos de menta simplemente se debía al hecho de que los sujetos eran fumadores y tomaban caramelos para combatir el mal aliento provocado por el tabaco, por lo que la correlación (positiva) realmente se encontraba entre los fumadores y el cáncer de pulmón.

En nuestro caso de estudio una correlación espuria podría ser la causada como se ha mencionado antes entre la editorial y el rating, porque un mismo autor puede publicar siempre con la misma editorial y si a un determinado usuario le gustan los libros de cierto autor aparentemente le van a gustar de la misma dicha editorial, pero esta correlación entre rating y editorial sería espuria ya que la verdadera sería entre la editorial y el rating del usuario.

No se debe olvidar que correlación no siempre implica causalidad.

## Objetivos

El objetivo de este TFM es crear un sistema recomendador de libros. Para ello comenzaremos con un dataset con información de libros y ratings obtenidos del repositorio de Kaggle [4], el cual limpiaremos y transformaremos para determinar las características más importantes a la hora de predecir un determinado rating para un determinado usuario.

A continuación, analizaremos que algoritmo (ICBF, UBCF, SVD, RBMs) tiene mejores resultados a la hora de recomendar un libro. Hay que tener en cuenta ciertas métricas como la “Novedad”, el “Churn de las recomendaciones” o el “Hit Rate” puedan ser más importantes en la práctica, para los sistemas de recomendación, que el cálculo de métricas de evaluación clásicas de los sistemas de aprendizaje automático tales como la exactitud o el RMSE.

Idealmente los resultados de este proyecto servirán como lanzadera para llevar esta idea (sistema recomendador de libros) al mundo empresarial y por lo tanto ver la viabilidad que tendría en un entorno real.

El resultado que se esperan tras la realización de este TFM es la obtención de un sistema de recomendación de libros con unos resultados razonablemente buenos, evaluado no sólo mediante las métricas mencionadas anteriormente (RMSE, Hit Rate, Novedad, etc.) sino también con ejemplos reales (utilizando diferentes usuarios reales para que voten determinados libros, que posteriormente nuestro sistema les proporcione recomendaciones y que finalmente estos usuarios confirmen que las predicciones tienen sentido y que les son de utilidad.

## Hipótesis

Las hipótesis son que una metodología similar a la usada para recomendar películas o ítems puede ser utilizada para recomendar libros.

Una hipótesis concreta sería, por ejemplo, el hecho de que el rating de un determinado libro para un usuario va a estar correlacionado con los géneros a los que pertenezca ese libro (ciencia ficción y/o humor, etc.)

Otra de las hipótesis a probar es que las redes neuronales funcionan para este tipo de sistemas, ya que éstas han mejorado muchas de las aplicaciones que hacen uso de los sistemas de aprendizaje automático.

## Fases del desarrollo

Las fases del desarrollo de este proyecto han sido las siguientes:

* Estudio del estado del arte de los sistemas de recomendación
* Recolección y limpieza de los datos, así como un análisis preliminar de estos (EDA)
* Implementación de los algoritmos “clásicos” de recomendación para el caso de los libros (tales como ITCF, UBCF, SVD o aproximaciones híbridas), así como su evaluación considerando las distintas métricas de calidad (RMSE, Hit Rate, Churn …)
* Uso de redes neuronales (RBMs) y comparación contra los algoritmos clásicos
* Uso de tecnologías de Big Data (Data Lake y Databricks (Spark)) para abordar el análisis de mayores conjuntos de datos, así como mayor potencia computacional
* Conclusiones

Capítulo 2

# Estado del arte

## Introducción

En este capítulo re realizarán una revisión a la literatura existente de los sistemas de recomendación, detallando las métricas más importantes y el funcionamiento de los algoritmos que se utilizan.

## Evaluación sistema recomendador



Uno de los problemas de los sistemas de recomendación es que es difícil decir como de buenos son. En teoría se podría evaluar un sistema de recomendación como cualquier otro sistema de aprendizaje automático con partición del conjunto de datos entre entrenamiento y evaluación, ya se mediante un particionado simple (train / test) o mediante un K-fold cross validation, para hacer predicciones (basándonos en los ratings previos de los usuarios) y medir la exactitud (accuracy) de dichas predicciones.

Sin embargo, usando estas técnicas lo único que podemos medir es la capacidad del sistema para predecir las votaciones de las personas sobre libros que ya leyeron, pero esto no tiene sentido para un sistema de recomendaciones ya que dicho sistema quiere recomendar nuevos libros no leídos, pero que los usuarios puedan encontrar interesantes. Esto es casi imposible de realizar porque nunca estaremos completamente seguros si el usuario va a leer el libro recomendado o no, es decir, difiere de otros sistemas de aprendizaje automático como el típico ejemplo de predicción del precio de las casas ya que éstas tarde o temprano se van a vender y de esta forma se puede calcular como de bueno es el sistema de aprendizaje automático en función de la diferencia entre el precio estimado y el precio de venta real.

### Medidas de error

Existen varias métricas para calcular el error de la predicción de un sistema de aprendizaje automático tales como el MAE, el RMSE o la exactitud. Sin embargo, estas métricas pueden no ser las más adecuadas para los sistemas de recomendación

En 2006, Netflix lanzó un concurso donde ofrecía una recompensa de 1 millón de dólares para el primer equipo que consiguiese una mejora del 10% sobre el RMSE existente en Netflix. Aunque el equipo ganador (BellKor) se hizo con el premio en 2009, Netflix nunca llegó a usar los resultados de su investigación ya que se dieron cuenta de que el RMSE no importa mucho en el mundo real, sino que lo que importa son las películas que se ponen delante de los usuarios dentro de un sistema recomendador Top-N y como los usuarios reacciones antes estas películas cuando ven que se les están recomendando. [6]

Por lo tanto, los sistemas de recomendación no sólo se basan en métricas de error como el MAE o el RMSE sino en otras más efectivas como el “Hit Rate”, el “Average Reciprocal Hit Rate”, el Churn o la diversidad.

A continuación, se describirán cada una de estas medidas de error.



#### MAE (Mean Absolute Error)

El *MAE* (Mean Absolute Error) es la diferencia absoluta entre el valor real y el predicho. Matemáticamente se corresponde con la siguiente fórmula

|  |  |
| --- | --- |
|  |  |

Siendo:

* **n =** el número de muestras (ratings) en el conjunto de test que queremos evaluar
* **y =** son cada una de las predicciones (estimaciones de las votaciones)
* **x =** son cada una de las votaciones reales dadas por los usuarios

El *MAE* suma cada una de las diferencias entre cada error y cada estimación y lo divide entre el conjunto de muestras para calcular la media. Queremos que este valor sea lo más bajo posible

A modo de ejemplo, supongamos que tenemos 5 ratings en nuestro conjunto de test como muestra la siguiente tabla:

|  |  |  |
| --- | --- | --- |
| Rating estimado | Rating real | Error |
| 5 | 4 | 1 |
| 4 | 4 | 0 |
| 3 | 5 | 2 |
| 2 | 3 | 1 |
| 1 | 2 | 1 |

Tabla 1: Rating estimado VS real y error

Como se puede observar para el primer rating nuestro sistema a predicho un rating de 4, pero realmente fue un 5, por lo que el error absoluto sería de 1, haciendo lo mismo para el resto de los ratings podemos calcular el MAE. Para el ejemplo de la Tabla 1, el MAE quedaría de la siguiente manera:

|  |  |
| --- | --- |
|  |  |

#### RMSE (Root Mean Square Error)

El *RMSE* (Root Mean Square Error) o error cuadrático medio es otra medida de exactitud (Accuracy) de los sistemas de aprendizaje automático más popular que el *MAE* porque penaliza los outliers de las predicciones y se representa mediante la siguiente fórmula:

|  |  |
| --- | --- |
|  |  |

Para el ejemplo anterior el RMSE sería el siguiente:

|  |  |  |
| --- | --- | --- |
| Rating estimado | Rating real | Error2 |
| 5 | 4 | 1 |
| 4 | 4 | 0 |
| 3 | 5 | 4 |
| 2 | 3 | 1 |
| 1 | 2 | 1 |

Tabla 2: Rating estimado vs real y error cuadrático

|  |  |
| --- | --- |
|  |  |

Como podemos ver, el error *RMSE* es mayor que el *MAE* ya que penaliza cuando el sistema se equivoca bastante con respecto al valor real.

#### Hit Rate

El cálculo de esta métrica es muy sencillo ya que simplemente genera las Top-N recomendaciones para todos los usuarios del conjunto de test.

Si una de las recomendaciones (de las top-N) recomendaciones es de algo que el usuario realmente ha valorado entonces ese elemento se considera como un hit, ya que el sistema ha sido capaz de mostrar al usuario algo que él realmente consideró lo suficiente interesante para leer por sí mismo y eso es considerado como un éxito. El Hit Rate se calcula sumando cada uno de estos “hits” de cada usuario para todos los usuarios entre el número total de usuarios.

Aunque el concepto del Hit Rate es fácil de entender su medición es más compleja ya que no podemos usar las mismas técnicas de división train / test o cross validation que se usan para la calcular la exactitud (Accuracy) ya que no estamos midiendo la exactitud de valoraciones individuales, sino sobre las listas Top-N para cada usuario.

Tampoco se debería calcular el Hit Rate directamente sobre cada lista del Top-N sin hacer el particionado ya que esto sería hacer trampas ya que no se puede evaluar un sistema usando los datos de entrenamiento.

Una solución a este problema es el llamado “Leave-one-out cross validation” en el que se computan las top-N recomendaciones para cada usuario de los datos de entrenamiento y se eliminan esos ítems para ese usuario dentro de sus datos de entrenamiento. Estamos probando la habilidad del sistema de recomendación de recomendar el ítem que ha sido dejado fuera de la lista Top-N que se ha creado para ese usuario dentro de la fase de test.

Sin embargo, el HR (hit rate) con “Leave-one-out” suele ser muy pequeño y difícil de medir a menos que se disponga de un dataset muy grande (aunque en los sistemas del mundo real suele ser necesario)

Es importante tener en cuenta que el RMSE y el Hit Rate no están siempre relacionados ya que hay sistemas recomendadores con bajo RMSE pero alto Hit Rate.

#### ARHR (Average Reciprocal Hit Rate)

El ARHR (Average Reciprocal Hit Rate) es una variación del Hit Rate pero que tiene en cuenta la posición en la que aparece los hits de la lista Top-N, es decir, si el hit aparece en las primeras posiciones de la lista tendrá más importancia que si aparece al final. En lugar de sumar el número de hits, sumamos el recíproco del rango, es decir, si el sistema ha predicho correctamente un ítem en la posición 4, el valor será de ¼, si fue la primera posición entonces sería 1 y así sucesivamente, es decir, el ARHR puede expresarse matemáticamente mediante la siguiente fórmula:

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
| Ranking | Reciprocal ranking |
| 4 | 1/4 |
| 3 | 1/3 |
| 2 | 1/2 |
| 1 | 1 |

Tabla 3: Ranking y Reciprocal ranking

#### CHR (Cummulative Hit Rate)

El CHR (Cummulative Hit Rate) es utilizado para eliminar aquellos hits que aparecen debajo de algún umbral ya que la idea es que no deberemos prestar importancia a aquellos ítems valorados por el usuario que realmente no le gustaron. Si, por ejemplo, nuestro umbral son las 3 estrellas (de 5 posibles), descartaríamos los hits de la segunda y cuarta fila de la tabla de abajo ya que el rating predichos fueron de 2 y 1 respectivamente. Finalmente, sobre las predicciones de los rankings “válidos” se calcularía el Hit Rate “normal”

|  |  |
| --- | --- |
| Ranking | Ranking predicho |
| 4 | 5 |
| ~~3~~ | ~~2~~ |
| 1 | 5 |
| ~~6~~ | ~~1~~ |

Tabla 4: Ranking y Ranking predicho y valores a eliminar según el CHR

#### RHR (Rating Hit Rate)

El RHR (Rating Hit Rate) simplemente tiene en cuenta la distribución de los hits de los ítems que le gustaron al usuario. A continuación, se suma el número de ratings para cada tipo de clasificación en nuestra lista de top-N recomendaciones y dividimos por el número total de libros de cada clasificación en nuestra lista top-N.

Un ejemplo de esta métrica es la mostrada a continuación:

|  |  |
| --- | --- |
| Rating | Rating Hit Rate |
| 5 | 0.0005 |
| 4 | 0.003 |
| 3 | 0.02 |
| 2 | 0.002 |
| 1 | 0.0004 |

Tabla 5: Rating y Rating Hit Rate

#### Cobertura

Esta métrica trata de encontrar buenas recomendaciones en nuestra lista top-N estableciendo un umbral que sólo permite buenas recomendaciones para cada usuario en la lista top-N y sumándolas y dividiéndolas por el número de usuarios. Es el porcentaje de pares (usuario, ítem) que pueden ser predichos por el sistema recomendador.

Hay que tener en cuenta que, en nuestro caso, el dataset de libros contiene muchos sin ninguna valoración.

La cobertura es importante porque nos permite ver como de rápido aparecen en nuestras recomendaciones los nuevos ítems del catálogo, pero a veces mejorar la exactitud (accuracy) puede implicar un deterioro en la cobertura.

#### Diversidad

La Diversidad es una medida de la variedad de los ítems de nuestro sistema de recomendación.

Un ejemplo de Sistema con baja diversidad sería aquel que solo recomendase el siguiente libro de una serie de libros que el usuario ya estuviese leyendo (como recomendar “El retorno del rey” si el usuario ya ha leído “La comunidad del anillo” y “Las dos torres”), pero que no recomendase libro de otros autores.

Aunque esta parezca una métrica subjetiva, se puede medir mediante la siguiente fórmula:

|  |  |
| --- | --- |
|  |  |

\*S = Similaridad media entre pares de recomendación

Si miramos la similaridad entre cada posible par de una lista Top-N de recomendaciones podemos calcular la media de como de similares son los productos de la lista entre sí (en nuestro caso los libros) y de esta forma calcular “S” y por lo tanto la Diversidad (que es el opuesto de la media de la similaridad). La mayoría se los sistemas de aprendizaje automático incorporan paquetes para calcular esta métrica de similaridad de manera casi automática.

La diversidad en el caso de los sistemas de recomendación no es siempre una cosa buena, ya que esta puede ser grande es sistemas que recomienden ítems de manera completamente aleatoria. De hecho, normalmente un Sistema con alta diversidad indica malas recomendaciones.

#### Novedad

La novedad (Novelty) mide como de populares son los ítems que estamos recomendando. Recomendaciones aleatorias producen altos índices de Novedad (ya que la mayoría de las recomendaciones no estarán dentro del Top-N)

Aunque la “novedad” se puede medir, lo que hacemos con ella es subjetivo, ya que la gente confiará más en el sistema recomendador si aparecen algún ítem que les hagan pensar que es una buena recomendación para ellos y que, por tanto, que el sistema recomendador es bueno.

Si sólo recomendamos cosas de las que nunca han odio hablar el usuario podría pensar que el sistema recomendador no funciona bien para ellos y por consiguiente dichos usuarios podrían no hacer caso de nuestras recomendaciones.

Adicionalmente los ítems populares lo son porque le suelen gustar a una buena parte de la población por lo que cabe pensar que serían buenas recomendaciones para un buen segmento de la población y si nuestro sistema no los recomienda deberíamos preguntarnos el porqué.

Teniendo todo este en cuenta, deberíamos tener un balance entre aquellos ítems recomendados que puedan resultarle familiares al usuario y aquellos completamente nuevos de los que nunca hubiese oído hablar de ellos, ya que los primeros le dan confianza al usuario y los segundos les son más útiles y que pertenecerían al llamado “the Long tail”. La forma que tiene la “long tail” es la siguiente:

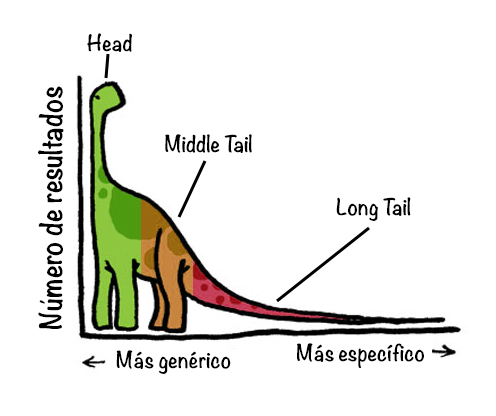


Figura 1: The Long tail [[1]](#footnote-1)

Donde el eje “Y” representa la popularidad y el eje “X” los ítems. La mayoría de los productos (libros) se encontrarán en el eje “X”. La mayoría de los ratings vendrán de un pequeño número de libros, pero todo el conjunto de todos los libros de la “Long tail” también sumarán un alto número de valoraciones. Los sistemas de recomendaciones ayudarán a las personas a descubrir a nuevos autores (que serán los que se encuentra en la “Long tail”) y, por lo tanto, nuevos libros.

#### Churn

En el caso de los sistemas de recomendación el “Churn” simplemente se refiere a la frecuencia con que cambian las recomendaciones a un usuario, es decir, como de sensible es el sistema ante un nuevo comportamiento por parte del usuario (o un nuevo rating en un nuevo libro). Es decir, el Churn para un sistema de recomendación sería complementario a la robustez de un sistema de aprendizaje automático.

A veces es conveniente mostrar a los usuarios nuevas recomendaciones en su lista de Top-N recomendaciones para darle un aire fresco al sistema de recomendación. De otra forma, mostraría siempre las mismas recomendaciones a los usuarios, aunque estos no las utilicen.

Al igual que la Diversidad y la Novedad, un Churn alto no tendría que ser por sí mismo una cosa buena ya que un recomendador aleatorio tendría un alto Churn pero dichas recomendaciones no serían buenas.

Por lo tanto, todas estas métricas deben ser evaluadas conjuntamente ya se necesitan entender las ventajas y desventajas de cada una de ellas

#### Responsividad

La responsividad sirve para medir como de rápido influencian en nuestro sistema recomendador las clasificaciones de los nuevos usuarios.

En principio, una buena responsividad sería una cosa positiva, pero ello implicaría la creación de un sistema complejo, caro y difícil de mantener, por lo que deberemos tener un balance entre la responsividad y la simplicidad.

### Evaluación final

Con tantas métricas (RMSE, Hit Rate, responsividad, etc.) es difícil decidir en cuál centrarse a la hora de evaluar un sistema de recomendación y efectivamente la respuesta depende, hasta tal punto que puede variar en función de la cultura, es decir, hay culturas en las que prefieran una mayor diversidad y novedad mientras que en otras culturas se prefieran las recomendaciones con las que ya están familiarizadas.

También el tipo de negocio (generalmente hacer más dinero) puede alterar el equilibrio de las métricas de evaluación.

La única forma de medir realmente que métrica es la más importante en nuestro sistema de recomendación sería mediante los A/B testings con nuestros clientes finales y evaluar como estos reaccionan a nuestras recomendaciones. En el caso de este TFM, se realizarán varias pruebas con sujetos conocidos, los cuales evaluarán algos de los libros de nuestro catálogo y se les proporcionará una lista de sus Top-N recomendaciones, y éstos proporcionarán su feedback para saber si dicha lista es relevante y tiene sentido para ello.

El uso y testeo de nuevos algoritmos de recomendación también ser debe realizar mediante estos experimentos controlados donde se puede observar si los cambios realmente causan a los usuarios la compra / lectura de nuevos libros. Esta parte no se desarrollará en este TFM ya que nuestros recursos son limitados.

Ninguna de las métricas anteriores es más importante que el comportamiento real de los usuarios en el mundo real, es decir, podemos tener un sistema que produzca una precisión muy alta con una bajo RMSE, pero si los usuarios no encuentran nuevos libros que comprar / leer gracias a nuestro sistema, entonces este será inútil desde un punto de vista práctico.

En el mundo real, también es importante mirar como de complejo es un nuevo algoritmo que pretende reemplazar a la ya existente, es decir, si el nuevo algoritmo es más complejo, pero no hay forma de medir como los usuarios responden a dicho algoritmo o las mejoras producidas son mínimas, entonces la recomendación sería descartar dicho algoritmo nuevo y seguir con el más simple ya que los sistemas complejos son siempre más difíciles de mantener.

También es importante tener en cuenta las velocidades de entrenamiento y de cálculo de predicciones de estos algoritmos ya que para un sistema real seguramente los usuarios deseen conocer sus recomendaciones una vez han valorado un cierto número de libros en el menor tiempo posible. Esto puede ocasionar que descartemos el uso de algoritmos más complejos y que quizás proporcionen unos resultados ligeramente superiores por estos motivos.

Las métricas “offline” tales como exactitud, novedad y diversidad son indicadores de cómo funciona el algoritmo de recomendación “offline”, pero no se puede cantar victoria hasta ver el impacto en el mundo real con usuarios reales. Youtube llama a esto el “problema subrogado” [6]. Según esto una buena exactitud no necesariamente hace buenas recomendaciones, y según ellos “hay más arte que ciencia” en la selección de los videos recomendados. Netflix llegó a una conclusión similar traes el concurso del millón de dólares.

Para evaluar estos sistemas lo mejor son los A/B testings para ver qué es lo que realmente impacta en nuestro sistema de recomendación. Como hemos mencionado anteriormente también se podría preguntar a nuestros usuarios directamente por feedback de nuestras recomendaciones y si estas son relevantes o no, e incluso que las evalúen, que es lo que se conoce como “calidad percibida” y aunque suena como una buena idea es difícil de medir (pues los usuarios podrían confundir el clasificar el ítem o la clasificación recomendada por nuestro sistema). Además, esto requiere trabajo extra por parte de nuestros usuarios sin una compensación clara para ellos.

Por todos estos motivos, es mejor realizar A/B testings y ver como los usuarios “votan” con su dinero (mediante la compra / lectura de libros) la calidad de nuestras recomendaciones.

## Algoritmos para sistemas de recomendación

### Introducción

Existen varias técnicas y métodos dedicados al estudio de los sistemas de recomendación. El primero de ellos es el llamado “Filtrado colaborativo”, que es una técnica utilizada por algunos sistemas recomendadores y simplemente hace referencia al hecho de que ítem con características similares tendrán aceptaciones parecidas por parte de los usuarios y que usuarios con gustos similares en el pasado, los seguirán teniendo en el futuro, pudiendo de estar forma predecir nuevos ítems que podrán ser recomendados a dichos usuarios.

También es común en uso de técnicas de aprendizaje automático y Machine Learning tales como PCA (SVD) o incluso la aplicación de redes neuronales (RBM).

### IBCF (Item-Based Collaborative Filtering)

Comenzaremos con la aproximación más simple, la recomendación basada en contenidos (Item-Based Collaborative Filtering) que es recomendar ítems simplemente basándonos en los atributos de esos ítems, en lugar de tratar de usar el comportamiento de los usuarios. Esto puede ser efectivo para recomendar libros del mismo género a persona que le gustan los libros de ese género.

Para recomendar libros basados en los atributos lo primero que necesitamos es un dataset con los libros clasificados por género / temática (política, ciencia ficción, fantasía, infantil, novela negra, intriga, históricos, amor …)

Un ejemplo muy sencillo de filtrado colaborativo basado en contenido es recomendar libros con temática similar a los que le gustan al usuario.

Para medir la “Similaridad” entre dos libros simplemente basándonos en sus géneros lo primero q necesitamos es clasificar los libros por géneros de la siguiente manera: para cada libro, le asignaremos una lista con todos los géneros que aplican a ese libro:

|  |  |
| --- | --- |
| Libro | Géneros |
| Yo, Robot | Ciencia ficción |
| Las fundaciones | Ciencia ficción | Acción |
| First blood | Acción |
| El ocho | Misterio | Ficción | Intriga | novela histórica |
| Starship troopers | Ciencia ficción | Acción |
| El Flautista de Hamelín | Infantil |

Tabla 6: Ejemplo géneros por libros

Sin embargo, esta solución no es muy eficiente para los sistemas de recomendación y necesitaremos seguir una estrategia similar al “one-hot encoding” para transformar la lista de categorías en dimensiones, de la siguiente manera:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Libro | Ciencia Fic | Acción | Intriga | Infantil | Novela hist | … |
| Yo, Robot | 1 | 0 | 0 | 0 | 0 |  |
| Las fundaciones | 1 | 1 | 0 | 0 | 0 |  |
| First blood | 0 | 1 | 0 | 0 | 0 |  |
| El ocho | 0 | 0 | 1 | 0 | 1 |  |
| Starship troopers | 1 | 1 | 0 | 0 | 0 |  |
| El Flautista de Hamelín | 0 | 0 | 0 | 1 | 0 |  |

Tabla 7: Libros con géneros como dimensiones

Mediante la métrica de la Similaridad de coseno podemos calcular como de parecidos son dos libros entre sí. Esta métrica es aplicable a otros sistemas de recomendación (no sólo a sistemas basados en contenido).

Para hacerlo más sencillo de visualizar vamos a suponer que cada libro sólo tiene 2 géneros posibles (Ciencia ficción y acción) (aunque estos resultados pueden ser extrapolados a *n* número de géneros). Podemos dibujar cada libro en 2D asignando el valor 1 si el libro tiene ese género en su lista géneros posibles y 0 en caso contrario.

Por ejemplo, el libro de Starship troopers está considerada como de ciencia ficción y de acción, por lo que para nuestro ejemplo nos daría las coordenadas (1,1), mientras que Yo robot sería simplemente ciencia ficción (coordenadas (1,0)). Con estas coordenadas ya estamos en disposición de dibujar los vectores correspondientes resultando en el siguiente diagrama:

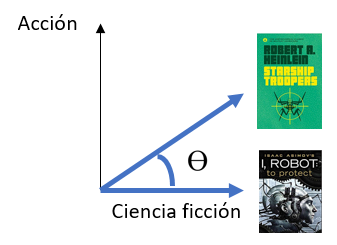


Figura 2: Ángulo entre los vectores de Yo Robot y Starship Troopers

Si miramos al ángulo de estos dos vectores (que representan los libros), podemos ver que forman un ángulo de 45ᵒ entre ellos, pero como queremos ver la “Similaridad” como número (entre 0 y 1) usaremos el coseno para determinarla.

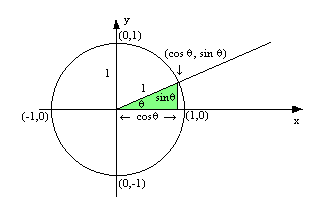


Figura 3: Coseno y seno vs ángulo

Como se puede ver en la figura de arriba en coseno es igual a 0 cuando el ángulo theta es 90ᵒ y toma el valor 1 cuando theta es 0ᵒ. Dado que en el caso de nuestro ejemplo el ángulo theta (formado por ambos vectores (que representan libros)) toma el valor 45ᵒ, el valor del coseno es de 0.707, es decir, la similaridad (según el criterio del coseno) entre los libros Starship troopers y Yo, Robot es de 0.707, dado que ambos comparten el género de ciencia ficción.

Siguiendo esta misma lógica podemos añadir otros libros, como First blood perteneciente solamente al género de la acción (coordenadas (0,1)) o las fundaciones, con géneros de ciencia ficción y acción (coordenadas (1,1)) y calcular otras similaridades:

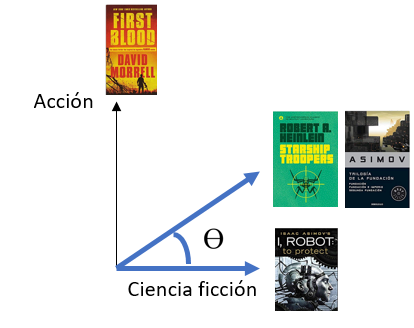


Figura 4: Representación vectorial de libros y ángulos

Como el ángulo formado entre Starship troppers y las fundaciones es 0, podemos fácilmente ver que la similaridad del coseno es 1. Sin embargo, si comparamos First Blood con Yo, Robot vemos que el águlo es de 90ᵒ y por tanto su similaridad es 0, es decir, es poco o nada probable que a alguien que le guste Yo Robot le guste también First Blood, ya que no tienen nada en común.

La fórmula de la similaridad de coseno se puede aplicar como hemos dicho a un número *n* de dimensiones (géneros) resultando en la siguiente ecuación:

|  |  |
| --- | --- |
|  |  |

La función del coseno tiene buenas propiedades matemáticas, pero se podrían usar otras distancias para medir la similaridad como la distancia euclídea:

|  |  |
| --- | --- |
|  |  |

Aunque para el caso de los géneros de los libros (de forma análoga a los géneros de los libros) se ha demostrado que la métrica de la similaridad del coseno ofrece buenas características, tanto por su sencillez, como por la capacidad de computación necesaria.

### KNN – K Nearest Neighbors

El KNN (K-Nearest-Neighbors) o de los k vecinos más próximos es un algoritmo de agrupación muy sencillo que se puede utilizar para transformar las similaridades entre los libros en predicciones reales.

La idea detrás de este algoritmo es sencilla, se comienza por calcular la similaridad de cada uno de los libros que ha clasificado el usuario y el libro que queremos predecir.

Después tenemos que seleccionar el número de “vecinos más próximos”: K que son aquellos con la similaridades más altas con relación al libro que estamos intentando predecir.

El concepto de KNN es, por ejemplo, seleccionar los 50 libros cuyos géneros son los más relacionados con el libro que queremos evaluar para ese usuario. Una vez que tenemos los vecinos más cercanos tomamos una media ponderadas de las similaridades del libro cuyo rating estamos tratando de predecir ponderándolo por el peso del rating que el usuario le ha dado. Es decir, explicado con un diagrama sería el siguiente:

Similaridades entre el libro y todos los otros usuarios que lo han clasificado

Ordenar

Top 50 libros más próximos

Media ponderada

Predicción del rating

Figura 5: Funcionamiento sistema KNN

### UBCF (User-Based Collaborative Filtering)

El UBCF (User-Based Collaborative Filtering o filtrado colaborativo basado en usuarios) es una implementación específica de un sistema de filtrado colaborativo basado en vecinos.

La idea es muy sencilla, ya que simplemente hay que empezar buscados a usuarios similares a uno mismo basado en el historial de ratings, algo así como las “almas gemelas” y entonces recomendarles contenido que no hayan leído aún.

En ejemplo gráfico pudiera ser el siguiente: Rocío y Nacho han leído el libro de “Yo, Robot” y además le han dado una buena puntuación. Rocío, además ha leído y dado una buena clasificación al libro “Dune” pero Nacho aún no, por lo que parece obvio recomendar a Nacho la lectura de “Dune”:



Figura 6: Ejemplo UBCF

Esto de forma matemática se puede hacer mediante tablas de ratings donde pondremos todos los ratings de todos los usuarios de nuestro sistema, en una especia de array de 2 Dimensiones, donde los libros estarán en un eje, los usuarios en el otro y los ratings en cada una de las celdas correspondientes. Un ejemplo de este comportamiento puede verse en la siguiente tabla, en el que además se ha añadido el usuario Jaime, que a priori, para no tener nada en común ni con Roció ni con Nacho:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Usuario \ Libro | Yo, Robot | Dune | La Fundación | La máquina del tiempo | Un mundo feliz |
| Rocío | 5 | 4 |  |  |  |
| Nacho | 5 |  |  |  | 1 |
| Jaime |  |  | 5 | 5 |  |

Tabla 8: Usuarios, Libros y clasificaciones

Si estos fuesen lo únicos 5 libros de nuestro dataset, podría describir al usuario Rocío como una vector 5-D de la manera (4,5,0,0,0) y Nacho como (5,0,0,0,1), donde en este caso el “0” representaría el valor faltante. La conversión a vectores numéricos nos permite calcular la similaridad del coseno entre 2 usuarios (también se podría usar otras medidas de similaridad, pero en nuestro caso utilizaremos la del coseno).

Si solo tuviésemos estos 3 usuarios (Rocío, Nacho y Jaime) podríamos crear una nueva matriz de 2D con los cálculos de la similaridad del coseno para cada para de usuarios, de la siguiente manera:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Rocío | Nacho | Jaime |
| Rocío | 1 | 0 | 1 |
| Nacho | 0 | 1 | 0 |
| Jaime | 1 | 0 | 1 |

Tabla 9: Similaridad del coseno entre usuarios

Como podemos ver, cada usuario es un 100% similar consigo mismo y tanto Rocío como Nacho tiene un 100% (1 de similaridad del coseno) entre ellos pues ambos han dado el mismo ranking a los libros que tienen en común y ambos tienen una similaridad de 0 con Jaime pues éste no ha leído ningún de los libros de los otros dos y viceversa.

Es importante tener en cuenta que un 100% de similaridad no significa que a ambos usuarios les gusten los mismos libros, sino que a los libros que los dos han leído, les han asignado la misma clasificación, es decir, podría ser el caso que ambos odien los mismos libros.

También existe otro problema a que es que, si 2 usuarios han leído 100 libros cada uno, pero sólo tienen 1 en común entre ellos y los 2 le han asignado la misma puntuación, entonces la similaridad del coseno entre ellos sería del 100%. Esto es conocido como el problema de datos dispersos (Sparse data / matrix) y ocurre de manera frecuente en los sistemas de recomendación, ya que puede conducir a resultados extraños. Para solucionar parcialmente este problema se puede optar por forzar un número mínimo de libros en común antes de considerarlos para el cálculo de la similaridad y de esta forma evitar estos resultados extraños.

En el caso de que intentemos generar recomendaciones para nuevos usuarios, tendría que mirar a estas matrices de similaridad para ver como de similar es este nuevo usuario con nuestra base de datos de usuarios y entonces podemos elegir aquellos usuarios más similares como nuestro Top-N vecinos. En el ejemplo anterior Rocío sería un vecino de Nacho, pero Jaime no. Adicionalmente y como es lógico se deben excluir las comparaciones de cada usuario consigo mismo. En situaciones con datos menos dispersos estaríamos en disposición de trabajar con nuestros datos y obtener los K vecinos más similares

En nuestro ejemplo, Rocío es el único usuario similar a Nacho, pero normalmente tendríamos más, con lo que simplemente tendríamos que listar aquellos candidatos que se parezcan más a Nacho.

Como adicionalmente queremos recomendar solamente aquellos libros que le gusten a los usuarios (y no aquellos que los dos odien), normalizaremos los datos a una escala de 0-1, donde el “1” representarán las 5 estrellas del rating. Además de multiplicar por su similaridad del coseno para calcular la similaridad entre ambos. También se podría normalizar modificando ratings bajos (1 o 2 estrellas) a valores negativos para que éstos vayan al final de nuestros resultados de similaridad ya que en principio no existe un modo estándar de normalizar los valores de las recomendaciones por lo que deberemos experimentar con las distintas implementaciones y ver cuál es la que funciona mejor.

En el ejemplo la similaridad entre Nacho y Rocío seguirá siendo 1 pues la similaridad del coseno entre ellos era 1 y además al haberle dado 5 estrella ambos serian 1 también, resultado en 1x1 = 1.

Finalmente, deberemos ordenar las recomendaciones de nuestros candidatos por sus puntuaciones finales.

Además de filtrar los candidatos para que no aparezcan en su misma lista de “vecinos”, de pueden aumentar los criterios de filtrado como que su similaridad sea mayor que cierto umbral o con un número mínimo de votaciones.

En nuestro ejemplo, de User-Based Collaborative Filtering tendrías que el libro “Dune” puede ser recomendado al usuario Nacho basado en las recomendaciones de sus vecinos más cercanos (Rocío).

Por lo tanto, para hacer un sistema de filtrado colaborativo basado en usuario (UBCF) necesitamos realizar los siguientes pasos:

1. Matriz usuarios -libros (en los ejes) y sus clasificaciones en cada una de las celdas.
2. Matriz usuarios-usuarios con sus similaridades
3. Buscar usuarios similares
4. Generaremos los “vecinos” o usuarios similares viendo que usuarios votaron cosas similares al que nosotros estamos intentando recomendar
5. Ordenaremos los candidatos según el grado de similaridad con nuestro usuario
6. Filtramos para tener sólo los usuarios similares (vecinos más cercanos)
7. Del total de libros recomendamos, eliminaremos aquellos que nuestro usuario ya haya visto (para sólo recomendarle nuevos libros)

### PCA

Los métodos de filtrado colaborativo (como Item-based KNN o User-based KNN) funcionan bien para sistemas de recomendación. Sin embargo, han sido altamente criticados por tener una escalabilidad limitada ya que el computo de matrices para dataset muy grandes en término de usuarios o de ítems requiere mucha potencia de computación. En teoría esto podría solucionarse utilizando Spark, que es capaz de distribuir el cálculo de las matrices a través de los clústeres.

Un problema real de los sistemas de recomendación son los datos dispersos o los registros con mucho “ruido” lo que hace que los sistemas de recomendación colaborativos requieran de datasets muy grandes para funcionar de manera razonablemente buena.

Una aproximación para solucionar este tipo de problemas es la utilización de métodos de machine Learning para entrenar los modelos y hacer predicciones (de los ratings de los usuarios a nuevos libros). Esta es una de las ideas principales por la que la librería “*surprise*” del paquete *scikit-learn* de Python fue construida.

Lo métodos de factorización de matrices necesitan trabajar con datos numéricos ya que transforman los atributos y dimensiones en matrices con la idea de describir el comportamiento de los usuarios según las combinaciones de features.

Por ejemplo, si al usuario Nacho está definido como 60% ciencia ficción y 40% historia entonces sería una buena idea recomendarle un 60% de libros de ciencia ficción y un 40% los libros de historia.

Las matrices descriptoras serán de 2D con los usuarios representados como columnas y los libros como filas. Sin embargo, las matrices resultantes serán muy dispersas, por lo que es buena idea utilizar técnicas de reducción de la dimensionalidad como el PCA (Principal Componente Analysis). En nuestro caso las dimensiones serán muy altas (como el número de libros que un usuario ha podido clasificar) y las reduciremos a un número menor de dimensiones como por ejemplo los géneros (que pueden describir un libro).

El Análisis de Componentes Principales es un método estadístico que usa una transformación ortogonal para convertir una serie de variables posiblemente correlacionadas en un conjunto de variables linealmente no relacionadas denominadas componentes principales. De este modo, se capturan las regularidades estadísticas de un dataset, de modo que las componentes principales son ortogonales (perpendiculares) entre sí y determinan las direcciones de máxima variabilidad de los datos, transformando un espacio con correlaciones lineales en un espacio sin esas correlaciones. Así, logramos dos ventajas: [1]

* 1. Que las nuevas variables no están correlacionadas entre sí, por lo que un algoritmo de aprendizaje automático puede explotar los datos sin temor a concentrarse en grupos de variables correlacionadas que influyen de manera excesiva en el aprendizaje, y por tanto impactan en su calidad.
  2. Que las nuevas variables son sustancialmente menos que las originales, de modo que los algoritmos de aprendizaje son sustancialmente más eficientes sobre el dataset y además podemos visualizar mejor el conjunto de datos.

La transformación PCA se construye de modo que la primera componente tiene la varianza máxima posible, es decir, captura el máximo de variación de los datos que es posible capturar con sólo una dimensión. Cada una de las siguientes componentes se van construyendo de modo que maximizan la variabilidad de los datos, pero son al mismo tiempo ortogonales o perpendiculares a las anteriores. El resultado es un conjunto de componentes, vectores o ejes que constituyen un nuevo sistema de representación de los datos, con ejes que son siempre perpendiculares entre sí y que además explican cómo varían los datos.

Los autovectores y autovalores pueden describir la varianza de los datos y su vector ortogonal que define los datos y que proporciona unos resultados similares. Los autovectores son los llamados “componentes” principales del análisis (PCA) y describen nuestros datos. Adicionalmente sirven para encontrar features ocultas que son inherentes a los datos, es decir, sin la necesidad de interpretar los datos, estos algoritmos son capaces de encontrar las características más importantes de los datos, por lo que también se puede considerar al PCS como una herramienta de extracción de características.

Normalmente, el cerebro humando no es capaz identificar gráficamente más de 3 Dimensiones, sin embargo, el PCA puede trabajo con decenas de ellas y reducirlas a un número menor, por ejemplo, pasando de 5D a 2D, con lo que sería hacer una representación visual de los datos (aunque no sabríamos decir que representan realmente estas 2 dimensiones), y esto sin perder apenas información.

#### SVD

Existen múltiples algoritmos para el cálculo de las componentes principales, se los cuales uno de los más populares es la Descomposición en Valores Singulares o Valores Propios o Autovectores (Singular Value Decomposition - SVD). Se trata de una técnica algebraica de operación de matrices disponible en muchos paquetes de software matemático y en casi todos los entornos de minería de datos. [9]

El PCA generalmente involucra una serie de cómputos costosos, agravados por el hecho de que de antemano no se conoce cuál es el número óptimo de dimensiones final K. Típicamente es necesario hacer pruebas con varios K << N, y aplicar a continuación los algoritmos de aprendizaje para ver cuál es el K óptimo. En otras palabras, en problemas de aprendizaje predictivo, para calcular el K óptimo, el método más lógico es el indirecto:

* 1. Hacemos pruebas con distintos valores de K, por ejemplo, con K = 2, 3, 4, etc.
  2. En cada prueba, entrenamos un algoritmo de aprendizaje y calculamos su efectividad.
  3. Seleccionamos el K que maximiza la efectividad del algoritmo de aprendizaje.

Cuando entran datos nuevos, no hay que cambiar el K sino aplicar el modelo PCA sobre los nuevos datos, y luego se aplica el algoritmo de aprendizaje, de manera encadenada. De ese modo, los nuevos datos están representados con las mismas dimensiones que los datos sobre los que hemos entrenado, y el modelo que hemos construido se puede aplicar sobre ellos porque se está utilizando el mismo sistema de representación que el usado durante el aprendizaje.

Una representación típica de un SVD es la que se muestra a continuación, donde se utilizan nuevas matrices que, multiplicadas entre sí, usando diferentes pesos aprendidos mediante la técnica de gradient descent, dan lugar a una representación más sencilla de los datos, pero con esencialmente la misma funcionalidad.

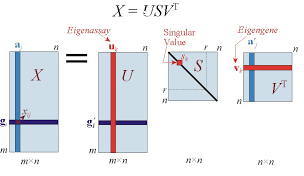


Figura 7: SVD [9]

Como hemos mencionado, para el caso de los sistemas de recomendación sería pensar que éstos son sistemas que se dedican a la búsqueda de patrones, aunque estos son muy complejos y estarán basados en el comportamiento de los usuarios.

## Deep Learning en sistemas de recomendación

### Introducción

Cabe preguntarse porque si porqué hoy en día que se le da tanta importancia a la inteligencia artificial y que hay muchas investigaciones que tratan de usar las redes neuronales para los sistemas de recomendación tienen sentido o es todo “hype”. Es decir, aunque las redes neuronales sean muy adecuadas para ciertos problemas (reconociendo patrones (al igual que el cerebro humando), el reconocimiento de imágenes o prediciendo secuencias de eventos) quizás no funcionen correctamente para los sistemas de recomendación.

Otra pregunta interesante sería si hacer recomendaciones es realmente un problema de reconocimiento de patrones, de la misma forma que cuando se utiliza TensorFlow, que al principio parece no tener mucho que ver con las redes neuronales, pero que sin embargo sirven de gran ayuda para el cálculo de estas mediante el uso de multiplicación de matrices y la adición de tensores que representar a las neuronas artificiales, sus pesos y su sesgo (Bias).

Las redes neuronales son fundamentalmente cálculos de matrices y hoy en día existen varias técnicas de factorización de matrices para los sistemas de recomendación que funcionan de forma similar.

Una de las principales razones por las que usar las redes neuronales a los sistemas de recomendación es que podremos utilizar muchas de sus ventajas, gracias a los rápidos avances en el mundo de la Inteligencia Artificial y el Deep Learning.

Amazon tiene un sistema open source llamado “DSSTNE” que permite la ejecución de grandes redes neuronales que permiten trabajar con datos dispersos (como lo son las matrices de los sistemas de recomendación) en un clúster de manera eficiente [7]. Adicionalmente existen formas de usar Tensorflow en un clúster y usar las ventajas proporcionadas por las GPUs.

Existen varias investigaciones en cursos para el uso de nuevas topologías de redes neuronales que puedan conducir a nuevas ideas acerca de cómo hacer mejores recomendaciones usándolas. Hoy en día existen ya algunos casos de algunas aproximaciones usando redes neuronales que pueden mejorar el rendimiento de los SVD, aunque por un margen pequeño.

En el siguiente apartado veremos cómo se pueden aplicar las redes neuronales para el caso de los sistemas de recomendación.

### RBM (Restricted Boltzmann Machines)

#### Introducción

La mayoría de los enfoques para filtrado colaborativo no pueden manejar bien grandes dataset, pero esto se puede solucionar mediante el uso de las RBM, que son unos modelos de dos capas unidireccionales que pueden ser usados para modular datos tabulares, como los ratings de libros por parte de usuarios.

Se ha demostrado en [5] que las RBM’s pueden ser usadas con éxito para el dataset del problema del millón de dólares de Netflix [6] y que contiene 100 millones de ratings usuarios / películas. Las RBM’s pueden mejorar los modelos incluso de los SVD tuneados.

Si además combinamos linealmente los resultados de los RBM’s con los de los modelos SVD, entonces se pueden incluso obtener unas mejoras del 6% (en términos de error) con respecto al sistema de scoring the Netflix [5]

Una práctica común para los sistemas de filtrado colaborativo es la asignación de vectores de dimensión baja de features para cada usuario y para cada libro. El rating de cada usuario para cada libro es modelado por el producto escalar de los dos vectores de features, resultando en una matriz NxM de ratings con “N” usuarios y “M” libros y que a su vez pueden ser modelados por la matriz “X”, que es el producto de los vectores UxV, siendo U una matriz NxC con las filas las features de los usuarios y V una matrix CxM con las columnas las features de los libros. El rango de X = C- número de features asignados a cada usuario o libro.

Aproximaciones de rango bajo basadas en la minimización de la suma de la suma de los cuadrados de la distancia mediante el uso de SVD. Sin embargo, en la mayoría de los casos de filtrado colaborativo, los datos son muy dispersos, lo que hace que los problemas sean no convexos y por lo tanto aquellas aproximaciones genuinas del SVD no vayan a funcionar bien. Sin embargo, este problema se puede solucionar mediante el uso de RBMs

El primer problema al que debemos enfrentarnos al aplicar RBMs es a tratar de forma eficiente los ratings faltantes, que dan lugar a matrices de usuarios-ratings muy dispersas.

Si tenemos “M” libros y “N” usuarios con ratings de 1 a K, entonces si todos los “N” usuarios hubiesen clasificado el mismo set de “M” libros entonces podrías tratar cada usuario como un caso único de entrenamiento para una RBM que tiene M “softmax” neuronas visibles simétricamente conectadas con un set binario de neuronas ocultas (hidden units). Cada neurona oculta puede aprender a modelar la dependencia entre los ratings de los diferentes libros.

Cuando la mayoría de los ratings no están presentes, como es el caso de los sistemas de recomendación, tendremos que usar diferentes RBMs para cada usuario. Cada RBM con el mismo número de neuronas ocultas, pero una RBM tendrá sólo neuronas visibles “softmax” para aquellos libros que ha valorado, por lo que, de esta forma, la RBM tendrá menos conexiones si el usuario ha valorado pocos libros.

Adicionalmente, cada RBM tendrá sólo un caso de entrenamiento, pero todos los pesos y su sesgo (Bias) estarán unidos por lo que, si dos usuarios han valorado los mismos libros, entonces se utilizarán los mismos pesos entre las unidades de softmax visibles para ese libro y sus neuronas ocultas. Sin embargo, las neuronas ocultas serán distintas para cada usuario.

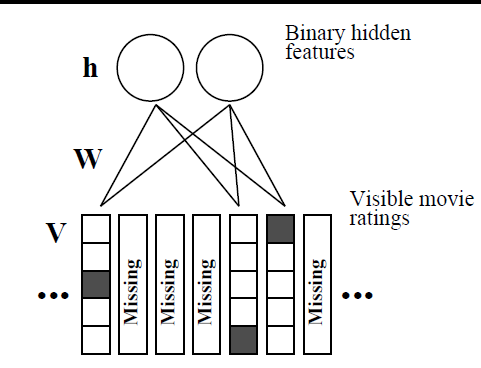


Figura 8: RBM con binary hidden units y softmax visible units [5]

#### RBMs condicionales

Si añadimos un peso “w” a cada uno de los “K” ratings posibles de cada feature oculta y usando la sobre parametrización del softmax, las RBM se pueden usar para aprender los ratings faltantes e influenciar de esta manera sus características ocultas. La estructura de las RBMs condicionales es la que se puede ver a continuación:

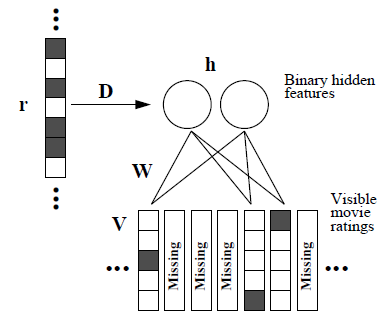


Figura 9: Arquitectura de una RBM condicional [5]

Las RBMs condicionales han sido usadas con éxito para el modelado de datos temporales, y han mostrado una mejora significativa en el dataset de Netflix.

En lugar de usar RBMs condicionales, podemos imputar los ratings faltantes mediante modelos RBM “ordinarios” pero, aunque esto sólo funciona bien en pequeños datasets.

#### RBMS condicionales factorizadas

Una desventaja de los modelos RBMs es que su parametrización actual resulta en un alto número de parámetros libres del modelo. Por ejemplo, para F = 100 (número de neuronas ocultas), M =177770 y K = 5 tendríamos 9 millones de parámetros libres. Esto podría causar overfitting, aunque se puede solucionar parcialmente utilizando pesos que decaigan con el tiempo. Sin embargo, si incrementamos el número de features ocultas o el número de libros, entonces el aprendizaje de la matrix de pesos “W” sería muy difícil.

La reducción del número de parámetros libres del modelo simplemente reduciendo el número de neuronas ocultas no soluciona el problema ya que entonces este modelo no podría contener la información necesaria de cada usuario en cada estado oculto. Este problema se soluciona mediante la factorización de la matriz de pesos “W” en 2 productos de matrices de rango bajo. Este se traduce en que las RBMs condicionales factorizadas converger mucho más rápido que las RBM condicionales sin factorizar y ambas produciendo unos resultados similares.

### Deep FM

Existe una investigación reciente que tiene resultados prometedores acerca del uso de Deep Learning en los sistemas de recomendación, y aunque su uso en estos sistemas de recomendación aún no es muy popular, tiene unos resultados prometedores. Esta investigación tiene que ver con la factorización de matrices, proceso similar al SVD con la diferencia de que también pueden utilizar datos categóricos. Además, demostró que se pueden encontrar relaciones entre cualquier combinación de features dada. [9]

Las Factorization Machines (FM) son unos modelos nuevos de clasificación que combinan las ventajas de los SVM (Support Vector Machines) y los modelos de factorización. Las FM a diferencias de los SVM modelan todas las interacciones entre variables usando parámetros factorizados y por este motivo son capaces de hacer estimaciones incluso en problemas con gran cantidad de datos dispersos (Sparse matrix), como los sistemas de recomendación, donde los SVM suelen fallar. [8]

Las Factorization Machines tiene un propósito más general que la SVD ya que además puede encontrar relaciones entre features que SVD ni siquiera considera.

La idea de las “Deep factorization machines” es combinar la potencia de Factorization Machines con la potencia de las redes neuronales (Deep Learning) para crear un sistema de recomendación más potente.

Esta aproximación supera en rendimiento al uso de Factorization Machines o redes neuronales individualmente, aunque no por un amplio margen.

Esta investigación es de 2017 por lo que todavía no se ha probado en sistemas con grandes cantidades de datos y consecuentemente no se sabe si podrá ser usados en sistemas de producción [8].

La motivación detrás del DeepFM es combinar las ventajas de las FM y de las Deep neural networks ya que las FM funcionan bien con un bajo número de interacciones entre variables, las redes neuronales lo hacen cuando el grado de las interacciones es alto, por lo que DeepFM intenta agrupar las ventajas de ambos mundos. Para ilustrar el grade de interacciones entre las features utilizaremos los siguientes ejemplos:

**Ejemplo 1**: Queremos construir un sistema de recomendación para ver que App nos tenemos que descargar en un momento preciso. Para una App de envío de comida a domicilio será más popular por la tarde antes de la hora de la cena y en este caso las features involucradas serían de orden 2 ya que hay una relación entre la categoría de la aplicación (comida) y la hora local para el usuario. Para este ejemplo de orden bajo la FM funcionaría bien.

**Ejemplo 2**: A los usuarios adolescente masculinos suele gustarles los juegos RPG. En este caso tenemos una interacción de orden 3: género y edad del usuario y categoría de juegos RPG. En este caso una red de Deep Learning podría funcionar mejor que las FM.

DeepFM combina lo mejor de ambos enfoques y puede obtener buenos resultados tanto para órdenes de interacción bajos como altos

Un pequeño problema con los sistemas de recomendación es que éstos no sólo pueden perpetuar el mismo sesgo que exista en los datos con los que han sido entrenados, sino que incluso lo pueden amplificar. Este se ha podido ver en el segundo ejemplo donde el sistema de recomendación perpetua los estereotipos (recomendando contenido de juegos RPG a adolescentes masculinos)

Las DeepFM pueden representares mediante la siguiente arquitectura:

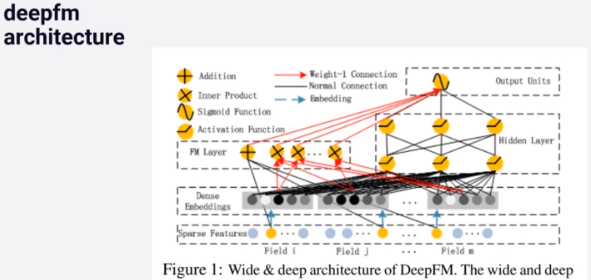


Figura 10: Arquitectura DeepFM [9]

El funcionamiento de esta arquitectura es más sencillo de lo que parece y como se puede se trata de un sistema híbrido, donde los datos de las features del dataset de entrenamiento se alimentar en paralelo a las FM y a la red neuronal (de dos niveles (hidden layers)).

Las salidas de las FM y de la red neuronal se combinan en el sigmoid que se utiliza como salida para la predicción.

Como se puede ver en el dibujo anterior de la arquitectura, las FM es “ancha” y por lo tanto adecuada para encontrar relaciones de bajo orden, mientras que la red neuronal es “profunda”, lo que significa que es más adecuada para encontrar relaciones de orden alto. Como ambas trabajan de manera conjunta, tendrán las ventajas de los dos mundos (aunque es según el problema en cuestión deberemos poner el énfasis más en uno o en la otra).

Las DeepFM son un ejemplo de sistemas de aprendizaje conjunto (ensemble Learning) para los sistemas de recomendación ya que como hemos dichos combinan las FM con las redes neuronales profundas. Estos sistemas, combinan otros más sencillos que funcionan mejor juntos que las implementaciones individuales de cualquier de ellos.

Los sistemas de aprendizaje conjunto pueden hacer que sistemas muy complejos sean manejables simplemente dividiéndolo en “módulos” que pueden ejecutarse en paralelo y combinando los resultados al final del proceso.

Con esta aproximación, además, si uno de los módulos deja de funcionar correctamente, lo podemos deshabilitar y seguir usando el resto de los componentes.

En el caso de las DeepFM se trata de la combinación de 2 tecnologías que se ha demostrado que funcionan correctamente (las Factorization Matrix y las Deep Neuronal networks), aunque su comportamiento combinado no se ha demostrado aún que mejore los resultados de los sistemas recomendadores. por lo que deberemos probar sus resultados, descartándolos en caso de que estos sean peores que los obtenidos mediante nuestros algoritmos “clásicos”.

Cabe recordar el hecho de que le premio de Netflix se ganó una un enfoque similar, donde se combinaban SVD con RBM que se demostró que combinados funcionaban realmente bien. DeepFM simplemente remplaza la parte del SVD con algo más genérico, que son las FM y las RBMs con redes neuronales profundas más generales también.

Capítulo 3

# Prueba de concepto

## Introducción

A lo largo de este capítulo realizaremos una prueba de concepto donde demostraremos la aplicación práctica de un sistema recomendador de libros sobre le dataset de Kaggle [4] y donde haremos uso de todos los conceptos analizado en el capítulo 2 del estado del arte, ya sea mediante la aplicación de algoritmos de recomendación como para la evaluación de los mismo con el ánimo de decidir qué algoritmo funciona mejor para nuestro dataset, así como la comprobación empírica del correcto funcionamiento de nuestro sistema recomendador mediante una recomendación real sobre un usuario real con gustos muy parecidos al desarrollador de este TFM.

Lo primero será comenzar con la limpieza, tratamiento de valores faltantes y el análisis exploratorio de los datos, siguiendo de la implementación, modelado y evaluación de cada uno de los algoritmos de recomendación, teniendo en cuenta el volumen de los datos (lo que hará que tengamos que dividir éstos en 2 datasets, uno de menor tamaño pero con una muestra representativa de éstos para trabajar en local de manera y otro casi completo que ejecutaremos en una plataforma de Big Data y procesamiento distribuido como DataBricks. Terminaremos el capítulo con las conclusiones obtenidas.

## EDA: Análisis exploratorio de datos



### Introducción

En este capítulo nos centraremos en los resultados obtenidos del análisis exploratorio de los datos (EDA como es conocido popularmente por sus siglas en inglés) a partir del dataset original obtenido del repositorio de Kaggle [4] que cuenta con miles de valoraciones de libros por usuarios que incluyen toda la información que se deseaba probar en las hipótesis. De esta forma no se necesitará recolectar los datos per-se sino simplemente adecuarlos al entorno de trabajo (mediante una limpieza de datos, tratamiento de valores faltante, valores anómalos, etc)

El análisis exploratorio de datos analiza datasets para resumir las características principales de dicho conjunto de datos, normalmente apoyándose en métodos visuales. No tiene porqué usar modelos estadísticos (aunque se puede emplear algunos estadísticos como el rango intercuartílico para identificar valores anómalos y poder eliminarlos o corregirlos con la media (para valores numéricos) o la moda (para valores categóricos)).

Este análisis nos permite identificar aquellos registros que cumplen con unos ciertos criterios de calidad. Adicionalmente nos va a permitir la formulación de hipótesis que nos puedan dirigir a la creación / adquisición de nuevas colecciones de datos y nuevos experimentos.

El EDA engloba al denominado IDA (Initial Data Analysis) ya que no sólo chequea y reduce los datos de manera conveniente para que se adapten a las hipótesis de partida, sino que permite cálculos más avanzados tales como correlaciones entre las variables dependientes del modelo (colinealidad), la relación entra la variable independiente y las dependientes pudiendo eliminar aquellas que no aporten nada, tales como los ZIF (Zero Influence Factors), identificación y corrección / eliminación de outliers, así como el cálculo de otras medidas estadísticas.

Muchas de las variables del dataset van a ser numéricas (como el género, autor) será necesario un tratamiento previo para convertir dichas variables categóricas a numéricas y de esta forma poder aplicar la mayoría de los métodos estadísticos utilizados para la medida de la correlación o para la predicción.

En este proceso se debe buscar el máximo control para lograr que otras explicaciones posibles, distintas a la propuesta en el estudio (hipótesis) sean desechadas, se excluya la incertidumbre y se minimice el error. Se debe tratar de trabajar con el binomio: causa-efecto. En nuestro caso y dado que disponemos de un dataset bastante grande donde diferentes usuarios con gustos similares (ha votado más o menos lo mismo para los distintos libros) podemos comprobar si los resultados “recomendados” para el usuario “A” son los mismos (o muy parecidos) que los recomendados para el usuario “B” y en caso contrario, analizar los posibles motivos de dichas diferencias (quizás una votación extra o faltante cambie por completo los resultados de nuestro algoritmo de predicción, en cuyo caso estaríamos ante un sistema poco robusto (ligeros cambios en el dataset de entrada provocan grandes cambios en la predicción)).

Los estudios cuantitativos siguen un patrón estructurado, de tal manera que las decisiones críticas son efectuadas antes de la recogida de datos. En nuestro caso dicha recogida de datos ya ha sido efectuada por otros investigadores anteriormente, es decir, nuestro punto de partida serán los dataset completos de valoraciones de libros por parte de los usuarios o dataset de características de cada uno de los libros (tales como género, autor, editorial, año publicación, ediciones, etc.) de modo que nuestra labor en este punto se limitará a “limpiar” este dataset para tratar con los valores faltantes y anómalos, ya sea eliminado completamente aquellos registros en los que la ausencia de valores dificulte el aprendizaje automático así como la corrección de valores anómalos (donde es posible la eliminación completa del registro como otras técnicas estadísticas tales como la sustitución por la media, mediana, etc)

Como veremos en capítulos posteriores el tamaño del dataset de datos es bastante grandes como parar poder trabajar con él “en local”, por lo que necesitaremos una sub-selección representativa de nuestros datos para ser más ágiles, aunque posteriormente debamos utilizar tecnologías Big Data para sacar las conclusiones basándonos en los resultados de la muestra completa. Será aquí donde entran el juego los distintos conceptos de población, muestras y unidades de análisis.

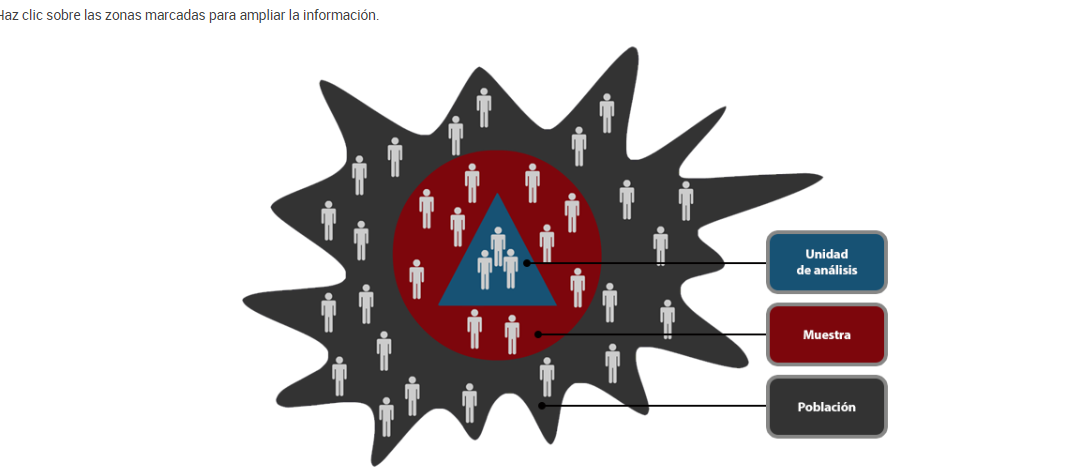


Figura 11: Población, muestra y unidad de análisis [1]

Para seleccionar la muestra, lo primero que hay que hacer es definir las unidades de análisis (usuarios y libros). Su definición dependerá del planteamiento del problema a investigar (recomendación de nuevos libros a usuarios existentes en función de sus ratings anteriores) y de los alcances del estudio (nuevas metodologías de aprendizaje automático y revisión de las existentes para sistemas de recomendaciones aplicadas al mundo de los libros).

1. La *población* es el conjunto de casos que concuerdan con una serie de especificaciones tales como usuarios que hayan valorado al menos 5 libros en los últimos 10 años.
2. La *muestra* serán todos los usuarios que hayan valorado al menos 5 libros, pero de la lista de libros disponibles en nuestro conjunto de datos y que se encuentren en nuestro dataset inicial.
3. En cuanto a las *unidades de análisis* se pude considerar a los usuarios (cada uno con su lista de valoraciones correspondientes) y los libros (con el número de usuarios que los han valorado, así como cada una de las puntuaciones).

### Análisis y limpieza de datos

A continuación, mostraremos los resultados reales de nuestro dataset (y su análisis), para lo cual analizaremos la distribución de nuestros datos para algunas de las dimensiones (variables independientes del modelo), correlaciones entre variables e influencia con la variable dependiente (rating).

Hay que tener en cuenta que los sistemas recomendadores son un tipo particular de sistemas de aprendizaje automático, pues la variable dependiente (ratings) va a depender completamente de los usuarios, es decir, a pesar de compartir completamente el resto de features tales como géneros, autores, número de páginas, etc, la valoración de un libro será completamente diferente dependiendo del usuario.

Para arreglar parcialmente este problema utilizaremos el promedio de las valoraciones (ranking\_average) y veremos que variables dependientes pueden influenciar en ella.

#### Diferentes idiomas

En el archivo "books.csv" se puede observar cómo existe información del lenguaje de cada libro (a pesar de que "goodreads" sea una web sólo de luenga inglesa). Podemos ver como el datase de "books" contiene libros en diferentes idiomas. La razón de esto es que normalmente existen múltiples ediciones del mismo libro en diferentes idiomas

A continuación, se muestra la distribución del número de nuestro dataset en función del idioma en el que estos están escritos:

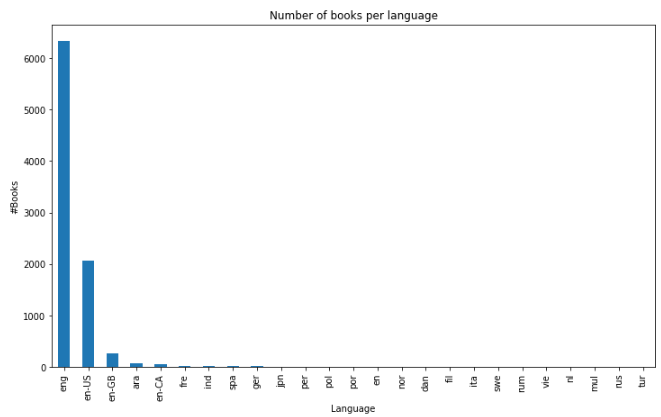


Figura 12: Distribución del número de libros según el idioma

Como se puede observar claramente, los libros escritos en inglés (tanto británico como estadounidense) predominan claramente sobre el resto. Para el posterior análisis de este dataset sólo se considerarán aquellos libros escritos en inglés.

#### Distribución de los géneros

La extracción de los géneros de los libros no es trivial, ya que estos no están normalizados y los usuarios pueden elegir texto libre en sus "tags". Estos "tags" se utilizarán para analizan los géneros de los libros. Sin embargo y dado que no existe un proceso de normalización para dichos tags. Por lo tanto, siendo pragmáticos sólo elegiremos las "tags" de la forma ('self-help', 'self help' etc. to 'Self Help') ya que son muy similares a las proporcionadas por la fuente del dataset (goodreads).

Adicionalmente, del total de géneros posibles sólo elegiremos unos predefinidos por nosotros que englobarán la mayoría de las categorías posibles de un libro. Esto reducirá el número de muestras de nuestro dataset, pero no será algo por lo que debamos preocuparnos ya que el tamaño original es demasiado grande para poder procesarse en ordenadores locales.

Es importante recordar también que un libro puede pertenecer a varias categorías, es decir, un libro como “Las fundaciones” pertenecerá al género de la ciencia ficción, pero también al de acción, y por lo tanto se le contará dos veces en el conteo de libros por géneros.

Una vez “normalizados” los géneros de los libros, podemos visualizar como se distribuyen en nuestro dataset, tal y como se muestra en la siguiente figura:

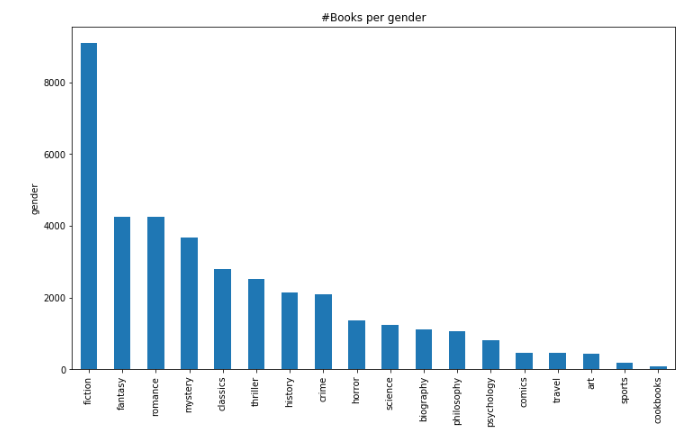


Figura 13: Número de libros por género

#### Distribución de la media de ratings por usuario

Debemos tener en cuenta que los usuarios pueden tener diferentes tendencias al evaluar un libro, ya que algunos le dan puntuación máxima a un libro mediocre mientras que otros sólo le darán el máximo al libro perfecto para ellos. Estas tendencias se pueden ver claramente en el gráfico inferior, donde además podemos ver como esta gráfica tiene esta sesgado positivamente, es decir, los usuarios suelen valorar más libros con 4 o 5 estrellas que con bajas puntuaciones.

Una hipótesis que puede explicar este comportamiento es que un determinado usuario sólo leerá un libro que esté predispuesto a leer, es decir, si al usuario “A” le gustan los libros de ciencia ficción, seguramente leerá un libro de ciencia ficción que le haya recomendado alguien (amigo, internet …) y por lo tanto es probable que le guste y le asigne una puntuación alta. Este mismo usuario no leerá libros de otras temáticas que no le gusten y por lo tanto será poco probable que le asigne bajas puntuaciones. Estas tendencias serán importantes para el filtrado colaborativo que veremos más adelante, pues suelen combinar el rating medio del usuario con un rating para un libro específico.

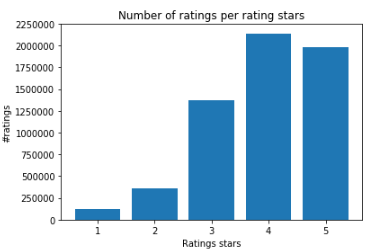


Figura 14: Distribución del número de valoraciones según sus ratings stars

#### Número de ratings por usuario

A continuación, mostraremos la distribución del número de ratings por usuario, para posteriormente eliminar usuarios con pocos números de ratings (perjudiciales para nuestros algoritmos de aprendizaje automático) ya que generarán matrices de datos aún más dispersas afectando tanto a la eficacia como a la eficiencia.

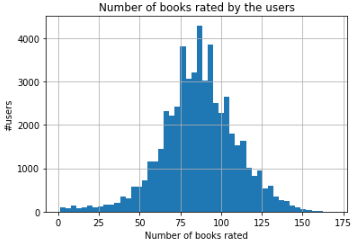


Figura 15: Distribución del número de usuarios según su número de ratings

#### Variables independientes predictoras

Lo primero que tenemos que preguntarnos es que variables independientes (predictoras / features) influencian o están correladas con nuestra variable dependiente (que es el ranking del libro).

En un vistazo rápido visualizaremos la matriz de correlaciones entre la variable “average\_rating” con algunas de las otras variables dependientes. Como podemos ver, existen pequeñas correlaciones entre las features y la variable dependiente, pero algunas correlaciones son altas entre features lo que implica que deberemos eliminarlas a la hora de calculas las predicciones para no afectar a los algoritmos de aprendizaje automático. Esto implica que el rating depende más fuertemente de otras features (como los géneros)

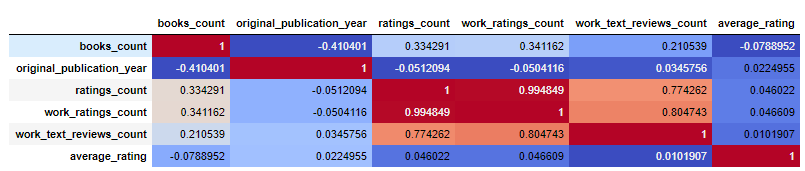


Figura 16: Correlación entre features y variable dependiente

#### Top 10 libros más votados

En el siguiente dibujo mostraremos los 10 libros más votados de todos nuestro dataset y podemos comprobar como efectivamente se trata de libros muy conocidos.

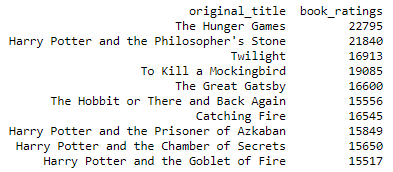


Figura 17: Top 10 libros más votados

#### Correlación del rating con el número de ratings

A continuación, demostraremos que los libros más populares también serán los más votados, es decir, existe una correlación positiva entre ambas cosas.

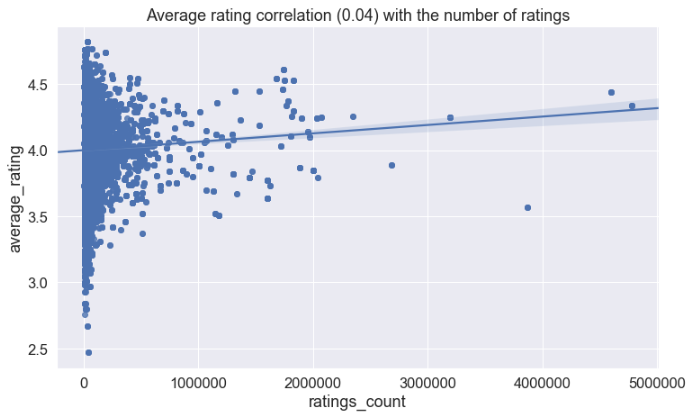


Figura 18: Correlación entre el rating medio y el número de ratings

#### Correlación del rating con el género

Para calcular la correlación del rating de un libro con el género de este utilizaremos la métrica de “F-Score” ya que los géneros son variables categóricas.



##### F-score

Esta medida se calcula de la siguiente manera:

|  |  |
| --- | --- |
|  |  |

La cobertura y la precisión se complementan, por lo que típicamente subir una implica bajar la otra. Es decir, existe un balance entre ambas medidas, por lo que no deben considerarse de manera aislada sino en conjunto. Para promediarlas, no se suele utilizar la media aritmética ni la geométrica, sino la medida F. Esta medida es una suerte de media ponderada promueve que la cobertura y la precisión sean altas a la vez, y que no estén desequilibradas. Es decir, la medida F tiene un valor alto cuando las dos son altas, y bajo cuando alguna de las dos es baja.

##### Cobertura (Recall)

Es el número de ejemplos de la clase clasificados correctamente sobre el número total de ejemplos de la clase. Esta medida indica cuantos ejemplos de la clase somos capaces de encontrar. Se calcula mediante la siguiente fórmula:

|  |  |
| --- | --- |
|  |  |

##### Precisión

Se corresponde con el número de ejemplos de la clase clasificados correctamente sobre el número total de ejemplos clasificados en esa clase. Esta medida aproxima cuanto acertamos entre los ejemplos que hemos puesto en la clase.

|  |  |
| --- | --- |
|  |  |

##### Resultados

Tras computar la medida F (F-score) entre nuestros géneros y el rating medio, podemos ver como esta medida es la siguiente (obtenida mediante el uso de los métodos proporcionados por la librería de scikit-learn.



#### Correlación del rating con el año de publicación

A continuación, intentaremos demostrar si existe algún tipo de correlación entre la valoración de un libro y el año / década en el que este se ha publicado. En primer lugar, visualizaremos la correlación de todos los años y el rating medio:

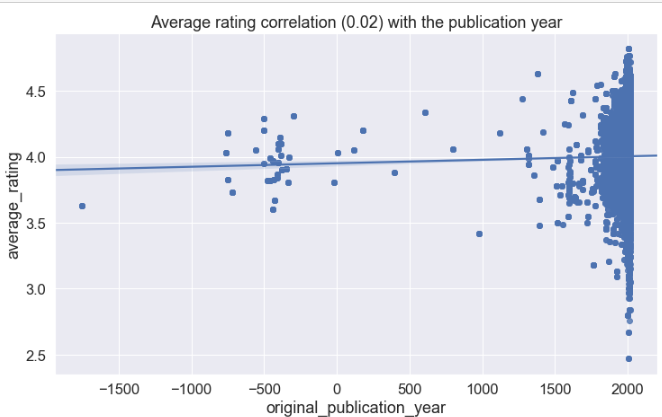


Figura 19: Correlación entre el rating y el año de publicación

Como podemos ver, la mayoría de los libros han sido publicados a lo largo del último siglo / 200 años, lo que distorsiona nuestra visualización. Además, se puede observar cómo aparentemente no existe una correlación entre el rating y el año de publicación.

Para intentar tener una idea más precisa de la correlación entre el año de publicación (original) y el rating vamos a visualizar solamente aquellos libros publicados en los últimos 200 años.

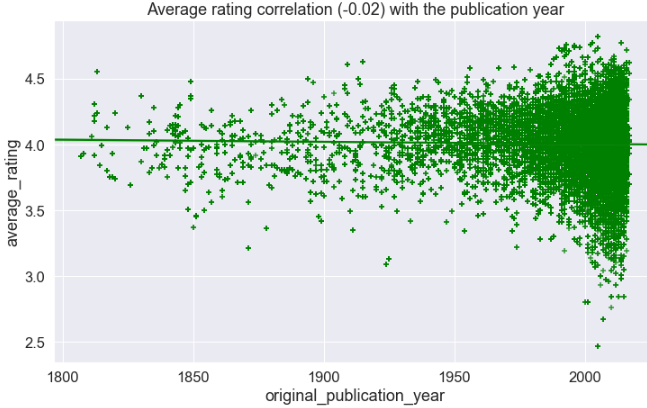


Figura 20: Correlación rating y año publicación (últimos 200 años)

Como podemos ver tampoco existe ninguna correlación entre el año de publicación y el rating medio.

### Resultados

El resultado del EDA será no sólo los insights obtenidos sino el dataset limpio necesario para poder implementar nuestros algoritmos de recomendación.

En este caso se han obtenido dos dataset, uno de menor tamaño para poder trabajar en local ya que la potencia del ordenador local (i7-8750H con 16 GB RAM y GTHX 1050Ti) no es suficiente a la hora de calcular los algoritmos de recomendación. Este problema se soluciona mediante el uso de tecnologías de Big Data, tales como Azure Data Lake para el almacenamiento de los datos y Databricks (Community edition) para el procesamiento utilizando procesamiento distribuido en memoria (Spark (PySpark ya que utilizaremos la API de Python)).

El problema de estas tecnologías es la necesidad del uso de internet y que en el caso del Databricks Community edition que cada día hay que crear el clúster y añadir las librerías necesarias, por lo que utilizaremos un dataset más pequeño, pero que contenga una muestra significativa de los datos donde los resultados puedan ser extrapolables al dataset completo de mayor tamaño.

En cuanto a los insights obtenidos hemos podido ver como no existe una buena correlación entra la mayoría de las variables dependientes del modelo y la variable objetivo (rating), si bien esto puede ser causado por el hecho de que los sistemas de recomendación son un caso muy particular de los modelos de aprendizaje automático, ya que aunque un mismo libro comparta todas las mismas características con otro similar los resultados entre distintos usuarios hacen que el resultado (rating) sea completamente distinto de uno a otro.

Se ha demostrado que no existe correlación entre el año de publicación y el rating, o entre el idioma y el rating.

También se ha visto como los usuarios suelen tener un sesgo positivo hacia los libros, ya que sólo leerán aquellos libros que presupongan que les van a gustar. O que el número de ratings esta correlacionado con el rating del libro.

Finalmente hemos calculado la medida F (F-score) entre los géneros de los libros y su rating, y serán estas las features que utilizaremos para los apartados posteriores de aprendizaje automático de sistemas recomendadores (tales como el filtrado colaborativo)

## Evaluación de algoritmos clásicos

### Introducción

Una vez hemos transformado el dataset original, mediante la eliminación / corrección de valores faltantes en la parte del EDA, así como de la sub-selección de solamente los atributos necesarios, estaríamos en disposición de aplicar los algoritmos de aprendizaje “clásicos” tales como es User-based KNN, Item-based KNN, SVD o SVD ++ además de un sistema recomendador aleatorio con el que podamos comparar si efectivamente los resultados obtenidos por dichos algoritmos arrojan buenos resultados o no. Deberemos tener en cuenta todas las métricas de evaluación vistas en el capítulo 2, desde las métricas más comunes utilizadas en los sistemas de aprendizaje automático tales como RMSE o MAE, así como las métricas más típicas de los sistemas recomendadores (Hit Rate, Cumulative Hit Rate, Average Reciprocal Hit Rate, Cobertura o Diversidad). Es importante tener en cuenta que un sistema con un buen RMSE no tiene porque ser el mejor en términos de Hit Rate y viceversa.

Como también estudiamos en el capítulo 2, los sistemas recomendares generan matrices muy dispersas de usuarios / ratings / ítems (libros) lo que requiere alto poder computacional. Es por este motivo que el notebook encargado del análisis, limpieza y exploración de los datos van a generar 2 datasets prácticamente iguales, pero uno de ellos de menor tamaño para poder trabajar en local pero que contiene una muestra representativa de los datos.

Para el análisis del dataset de mayor tamaño se utilizarán herramientas de Big Data y procesamiento distribuido tales como DataBricks. Posteriormente se comprobarán que los resultados de ambos procesos arrojar resultados similares en cuanto a métricas de evaluación, lo que nos servirá para determinar el algoritmo óptimo para un sistema recomendador de libros.

En este apartado no sólo veremos que algoritmo es más óptimo para nuestros sistema recomendador de libros desde el punto de vista las métricas de evaluación, sino que además generaremos una lista de las top-N (top-10) recomendaciones para un usuario cuasi-real con unos gustos muy similares a los del investigador con el fin de demostrar empíricamente que el sistema recomendar de libros funciona correctamente, ya que la lista de libros recomendados deberá contener aquellos libros que el usuarios haya podido leer, pero no haya votado aún o libros en los que el sujeto realmente estaría dispuesto a leer.

### Resultados y conclusiones

En el mundo real existen muchas aplicaciones capaces de generar recomendaciones. En nuestro caso particular usaremos la librería “*surprise*” de Python que nos permite obtener recomendadores y sus métricas de calidad más importantes

En nuestro sistema real con nuestro dataset de libros hemos implementado ambas soluciones de filtrado colaborativo (IBCF o Item-based KNN y UBCF o User-based KNN), así como soluciones basadas en el análisis de componentes principales (SVD y SVD++). Posteriormente, hemos analizado los resultados obtenidos, comparándolos siempre contra el valor de un generador de predicciones aleatorio y esos son los resultados obtenidos:

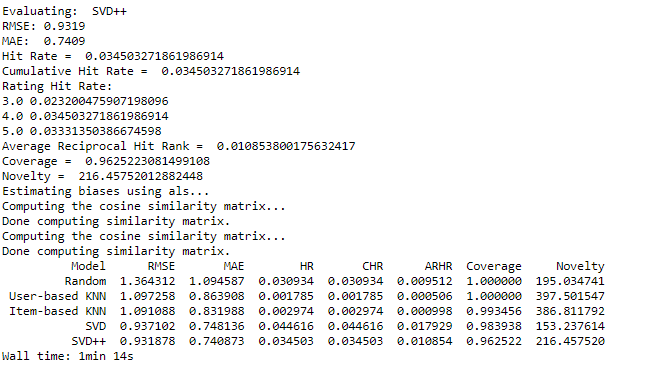


Figura 21: Resultados de predicción de algoritmos de recomendación

En la figura anterior, se muestra un pantallazo correspondiente al a ejecución del algoritmo SVD++ para nuestro dataset reducido (pero representativo) de los ratings de libros por usuario donde se muestra el tiempo de ejecución del algoritmo (1min 14 segundos), así como las métricas de error de los sistemas de aprendizaje automático (RMSE y MAE) y cada una de la métricas más representativas de los sistemas de recomendación (Hit Rate, Cummulative Hit Rate, Average Rate Hit Ranking, Coverage y Novelty). Aunque el Rating Hit Rate se calcula para cada uno de los algoritmos utilizados no se ha considerado para la comparación final entre los algoritmos ya que resulta en varios valores (uno para cada puntuación). Adicionalmente se muestran algunos de los pasos que los algoritmos realizan internamente para el cálculo de dichas métricas, tales como el cálculo de la similaridad del coseno.

Esta misma ejecución se ha realizado para todos los algoritmos considerados (Random, Item-based KNN, User-based KNN, SVD y SVD++) y esta es la comparación entre los resultados obtenidos para cada una de las métricas evaluadas:

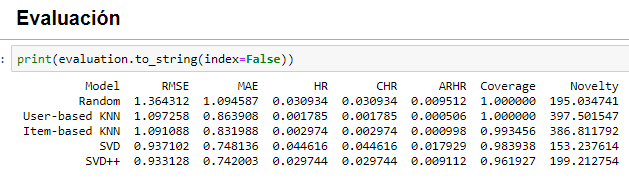


Figura 22: Comparación resultados entre algoritmos de recomendación

Como hemos mencionado anteriormente estas métricas han sido obtenidas utilizando el dataset de menor tamaño (pero representativo de nuestros datos) y serán analizadas y comparadas en apartados posteriores (“Big Data” – utilizando un dataset de datos mayor y “Deep Learning” donde además se compararán con las redes neuronales).

Para el caso de estudio de este dataset se puede ver cómo los algoritmos SVD y SVD ++ son los que mejores resultados producen en términos de RMSE y MAE (menores) y con mejores resultados en métricas típicas de sistemas de recomendación tales como el Hit Rate o el ARHR.

A modo de ejemplo se demostrará como realmente funciona nuestro sistema recomendador, para lo cual se ha seleccionado un usuario del dataset con unos gustos similares a los del desarrollador de este TFM a modo de comprobar si realmente recomendados estuviese dentro de lo que un usuario pudiese aceptar como “recomendables”. En primer lugar, se han mirado usuarios con votaciones altas para libros de interés parecido, siendo el usuario “43140” un candidato para esta prueba ya que sus votaciones reales son las siguiente:

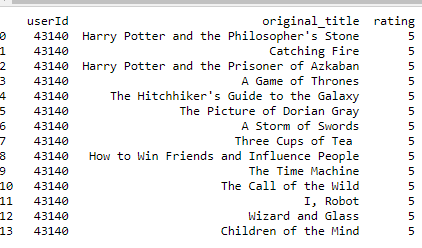


Figura 23: Votaciones reales usuario 43140

A continuación, vamos a calcular el Top-N (top 10) de libros que le recomendaría a dicho usuario (aplicando el algoritmo SVD ya que era el que mejores resultados producía en términos de recomendación), resultando en las siguientes recomendaciones:

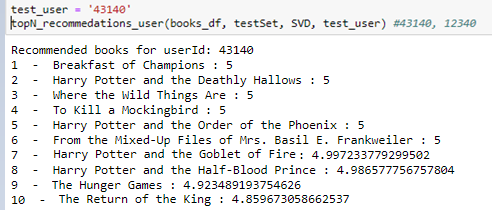


Figura 24: Top-N recomendaciones

Como se puede ver la lista recomendada (además del rating esperado de recomendación) tiene bastante sentido en para este usuario en función de sus valoraciones previas, ya que aparecen varios libros de series similares (Harry Potter o Juego de tronos) que, y esto es importante, el usuario en cuestión no ha valorado aún, pero que posiblemente haya leído, pero no ha valorado, de esta forma el usuario confiará en nuestro sistema de recomendación.

## Evaluación de Deep Learning

### Introducción

Tal y como vimos en el capítulo 2, las redes neuronales han demostrado obtener buenos resultados en otros sistemas de aprendizaje automático, aunque aún tengan poco uso (hay varias investigaciones en curso) en los sistemas recomendadores. Una de las principales razones por las que usar las redes neuronales a los sistemas de recomendación es que podremos utilizar muchas de sus ventajas, gracias a los rápidos avances en el mundo de la Inteligencia Artificial y el Deep Learning.

A lo largo del siguiente epígrafe mostraremos los resultados de una red neuronal del tipo RBM (Restricted Boltzman Machines)

### Resultados RBMs

Tras implementar una red neuronal del tipo RBM para nuestro dataset de libros podemos comprobar cómo tanto el RMSE como el MAE son más bajos que para los sistemas aleatorios, tal y como se puede ver a continuación:

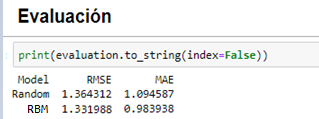


Figura 25: Resultados ejecución RBM

Esto quiere decir que efectivamente estas redes neuronales funcionan para los sistemas de recomendación, sin embargo, estos errores son ligeramente mayores que los obtenidos usando otros algoritmos tales como SVD o SVD++, y eso sin tener en cuenta el tiempo de cálculo que necesitan unos y otros, ya que para el dataset analizado de unos 5.000 libros y 10.000 ratings el cálculo de la mayoría de las métricas y predicciones varía desde los pocos segundos / minutos obtenidos mediante SVD, SVD++, IBCF, UBCF a los minutos / horas de las RBMs por lo que a priori podríamos pensar que no es buena idea utilizar estas redes neuronales para los sistemas de recomendación.

Los resultados de las RBMs varían significativamente en función de sus hiperparámetros, tales como:

1. Número de épocas
2. Número de dimensiones ocultas
3. Ratio de aprendizaje
4. Tamaño del batch

Sin embargo, para todas las configuraciones usadas los resultados han sido similares en comparación con los otros algoritmos, es decir, algo peores en términos del MAE y RSME y mucho peores en términos de eficiencia.

A continuación, se muestran los resultados de la ejecución de la RBM para las siguientes configuraciones de hiperparámetros (épocas, dimensiones ocultas, ratio aprendizaje y batch)

* RBM\_1: 20, 50, 0.01, 50
* RBM\_2: 20, 100, 0.01, 50
* RBM\_3: 20, 50, 0.05, 50
* RBM\_4: 20, 100, 0.05, 50
* RBM\_5: 20, 50, 0.01, 100
* RBM\_6: 20, 100, 0.01,100
* RBM\_7: 20, 50, 0.10, 100
* RBM\_8: 20, 100, 0.01, 100

Produciendo los siguientes resultados (comparados también con los resultados obtenidos por los algoritmos “clásicos” de recomendación utilizado en capítulos anteriores:

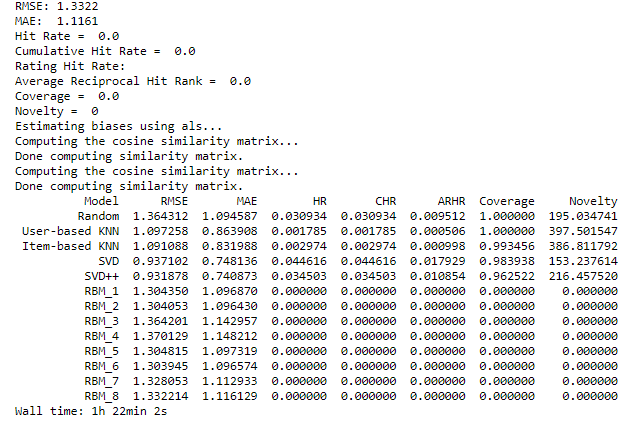


Figura 26: Resultados RBMs en función de sus hiperparámetros

Como podemos observar los resultados son mejores que en un sistema aleatorio, pero sin embargo peores que el resto de los algoritmos (SVD o Item-based KNN), esto se puede deber a que el dataset no sea lo suficientemente grande, a que la topología de la red neuronal no es la óptima o a que los hiperparámetros elegidos nos han sido lo óptimos. En cualquier caso, el uso de redes neuronales en sistemas de recomendación es algo que hay que tener en cuenta para el futuro próximo, porque si bien aún no está muy estudiado puede tener unos buenos resultados, algo que se explicará con más detalle en el capítulo de “líneas de investigación futuras”.

## Big Data



### Introducción

Tal y como se ha comentado en el epígrafe del análisis exploratorio de los datos, dado el volumen del dataset es difícil trabajar en local, a pesar de contar con un ordenador potente[[2]](#footnote-2) no es suficiente a la hora de calcular los algoritmos de recomendación con grandes cantidades de datos. Este problema se solucionó parcialmente en capítulos anteriores mediante el uso de técnicas de limpieza del dato “muy estrictas” así como de reducción de ambos datasets (libros y ratings) hasta ajustarlos a un tamaño capaz de ser procesado localmente, pero que a la vez produjese buenos resultados en términos de análisis y aprendizaje automático.

Este problema puede solucionarse mediante el uso de tecnologías de Big Data, tales como Azure Data Lake para el almacenamiento de los datos y Databricks (Community edition) para el procesamiento utilizando procesamiento distribuido en memoria (Spark (PySpark ya que utilizaremos la API de Python)).

El problema de estas tecnologías es la necesidad de estar conectados a una red de internet y que en el caso del Databricks Community edition que cada día hay que crear el clúster y añadir las librerías necesarias. Adicionalmente el clúster gratuito proporcionado por Databricks Community es el menos potente de todos por lo que si un fuese capaz de procesar todo nuestro dataset deberíamos de hacer un upgrade mediante un pay-per-use.

### Azure Data Lake

Azure Data Lake es el servicio de almacenamiento proporcionado por Microsoft dentro de su nube (Azure). Es capaz de almacenar grandes cantidades de datos a un precio bastante bajo, tal y como podemos ver en la siguiente tabla:



Tabla 10: Precio por Byte Azure Data Lake \*

(\* <https://azure.microsoft.com/en-us/pricing/details/data-lake-storage-gen1/>)

El Data Lake es capaz de trabajar con roles y permisos por lo que nuestros datos no serán accesibles públicamente y hemos elegido DataLake Generación 1 para trabajar.

Este es el aspecto de nuestra suscripción de Azure DataLake:

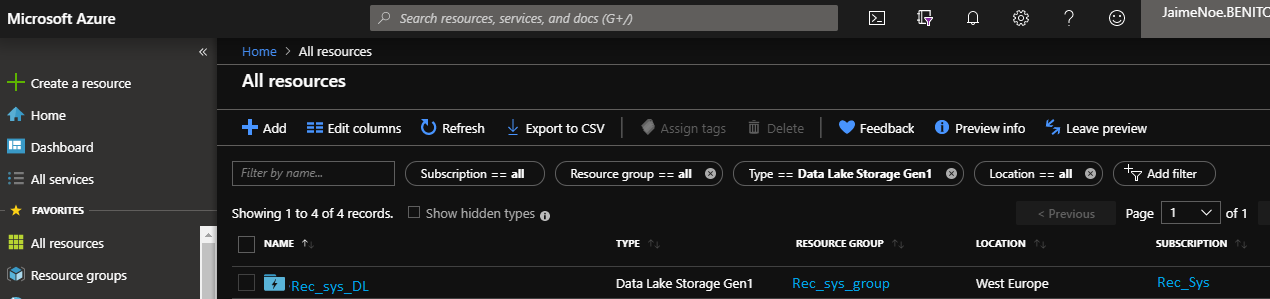
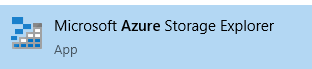


Figura 27: Azure DataLake suscription

Como podemos ver este DataLake se ha creado bajo la nueva suscripción “Rec\_sys” y se le ha asignado al Resource group “Rec\_sys\_group” lo que permitirá acceder sólo a miembros de estos grupos y adicionalmente necesitaremos las credenciales de acceso para que Databricks sea capaz de acceder a él.

El DataLake puede verse como un sistema de archivos en la nube y puede accederse bien mediante la web o mediante la aplicación “Microsoft Azre Storage Explorer”, que simula a al explorador de fichero de Windows, pero sobre las carpetas y ficheros del DataLake



### DataBricks

DataBricks es una empresa fundada por los creadores originales de Apache Spark y que surgió del proyecto AMPLab en la Universidad de California, Berkeley, los cuales participaron en la creación de Apache Spark, un marco de computación distribuida de código abierto construido sobre Scala y que es hasta 10 veces más rápido que MapReduce cuando trabaja en disco y hasta 100 veces si trabaja en memora.

Aunque está optimizado para trabajar en Scala la API de Python proporciona unos resultados similares en términos de velocidad de procesamiento (ligeramente superiores) pero nos permite utilizar una mayor cantidad de librarías de machine Learning. Adicionalmente nuestros programas los hemos creados utilizando un Jupyter notebook en local, los cuales se pueden importar directamente en Databricks. Por este motive elegiremos “Python” como lenguaje de nuestros notebooks de DataBricks.

Para poder usar Databricks lo primero que necesitamos es crearnos una cuenta con cualquier email y registrarnos, proceso transparente y rápido que sólo tendremos que realizar una vez.

#### Creación del clúster

Para empezar a trabajar necesitaremos crearnos un clúster, ya que en la edición “free” (Community) este se destruye por inactividad pasados los 30 minutos. Adicionalmente cada vez que se destruya, tendremos que crear uno completamente nuevo, esto incluye la instalación de librerías, que en nuestro caso solo será añadir “scikit-surprise” que es muy típica de los sistemas de recomendación y por eso no viene instalada por defecto, como otras librerías tales como pandas o random.

Este es el precio a pagar por usar los clústers gratuitos. Si utilizamos la opción de pay-per-use entonces podemos elegir la potencia del clúster, sólo necesitaríamos instalar las librerías una vez ya que el clúster sería el mismo, aunque este podría estar encendido o apagado (conviene apagar el clúster cuando lo estemos utilizando para evitar gastos extras y el arranque del clúster es bastante rápido (de 1 – 5 minutos dependiendo de la potencia de éste).

Algunos de los precios de los clústers de Databricks son los siguientes:

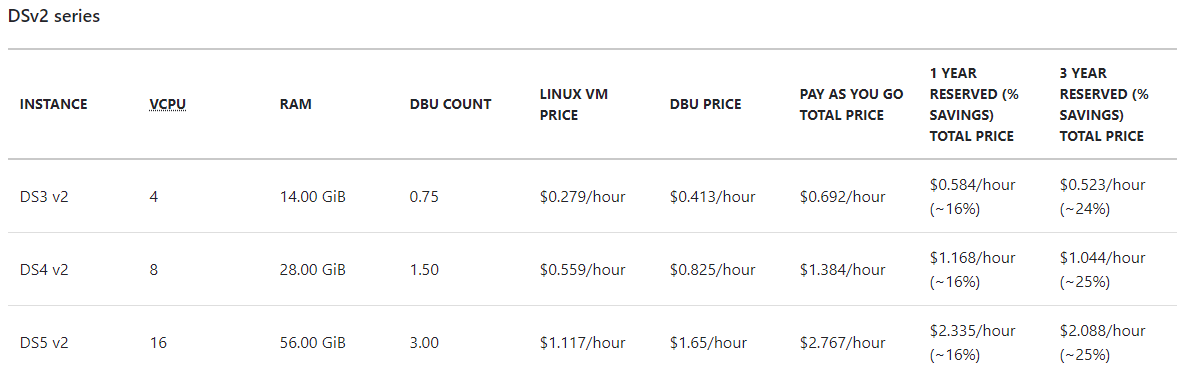


Tabla 11: Precios DataBricks

Nosotros elegiremos el clúster gratuito, que se crea por defecto y que incluye las siguientes versiones:

1. Spark 2.4.3
2. Scala 2.11
3. Python 3

Además de 6 GB de memoria, 0.88 cores y 1 DBU, tal y como se puede ver a continuación:

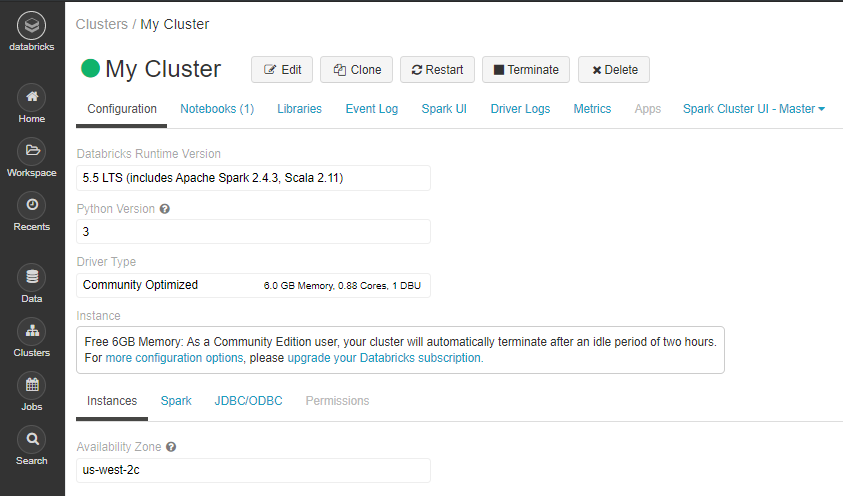


Figura 28: Creación clúster en DataBricks

#### Notebooks

A pesar de que el clúster se destruya después de cada ejecución, esto no ocurre ni con los notebooks, ya que una vez importados o creados los podremos reutilizar tantas veces como queramos. Estos son accesibles a través del workspace tal y como se puede observar en el siguiente pantallazo:

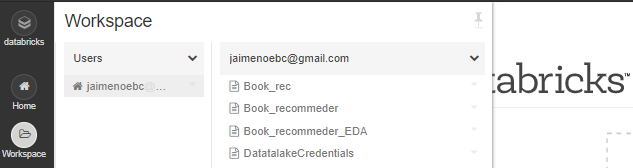


Figura 29: Screenshot de los notebooks del workspace

Como podemos ver tenemos varios notebooks:

1. Book\_recommender\_EDA (para el análisis exploratorio de los datos)
2. Book\_recommender (para el sistema de recomendación)
3. DataLakeCredentials (con las configuraciones para conectar Databricks y ADL)

Los dos primeros notebooks son exactamente los mismos que se utilizaron en el capítulo 4 tanto para el análisis exploratorio de los datos como para la predicción y cálculo de recomendaciones, con la única diferencia de los ficheros de entrada, ya que ahora es posible utilizar un volumen de datos mucho mayor (concretamente se ha pasado de usar 10.000 rating en local a 100.000 en DataBricks).

#### Datos

Además de utilizar DataLake como fuente de datos, DataBricks permite cargar los datos localmente de forma manual, tal y como se detalle a continuación:

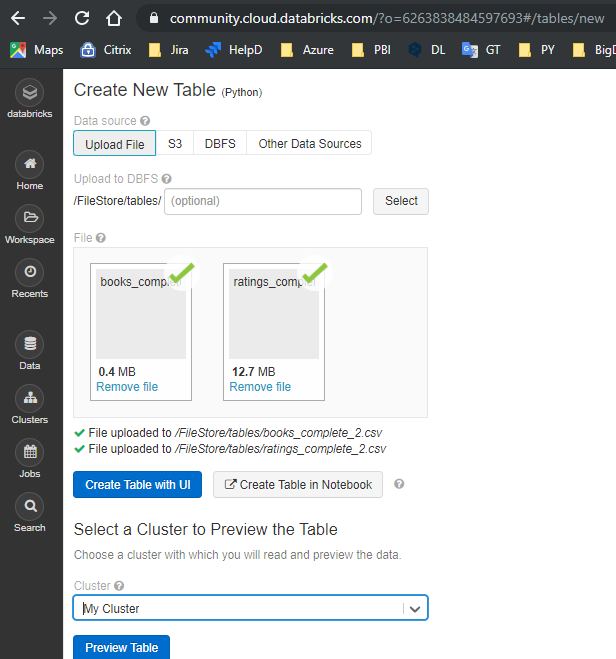


Figura 30: Importación de datos en Databricks

Esta opción nos ahorra el script de configuración entre el DataLake y Databricks donde se comparten las credenciales y funciona de manera razonablemente bien para ficheros pequeños como los utilizados a lo largo de este proyecto. La opción de usar DataLake es mejor para grandes ficheros de datos y por el hecho de que luego es más sencillo compartirlos con otras aplicaciones.

#### Ejecución y resultados

Una vez importados los datos y los notebooks estamos en disposición de ejecutar tanto nuestros procesos de análisis y limpieza de datos como de predicción y recomendación. Lo primero que nos ocurre es que vemos que no es posible utilizar todos los datos ya que el clúster gratuito no es lo suficientemente potente y produce el siguiente error:



Eso no es un problema porque se han solucionado parcialmente utilizando un subset de los datos más complete que el utilizado en local y que además contiene una muestra representativa de todos los datos válidos.

Los resultados del EDA son prácticamente los mismos que los obtenidos en el capítulo 4 por lo que no repetiremos los resultados obtenidos y nos centraremos en aprendizaje automático y en los resultados de los distintos modelos de recomendación para un conjunto mayor de datos. Los resultados obtenidos han pasado de los pocos segundos / minutos a minutos / horas tal y como vemos a continuación

##### Random

El modelo aleatorio produce los siguientes resultados en términos de error, métricas de recomendación y tiempos de ejecución:



Figura 31: Tiempos y resultados modelo aleatorio

##### User-based KNN

El algoritmo de User-based KNN mejora el RMSE respecto al modelo aleatorio, pero, sin embargo, tiene unos ratios de recomendación peores (Hit Rate más bajos), esto sin tener en cuenta que el tiempo de ejecución es mucho mayor, tal y como podemos observar en el siguiente pantallazo de la ejecución:

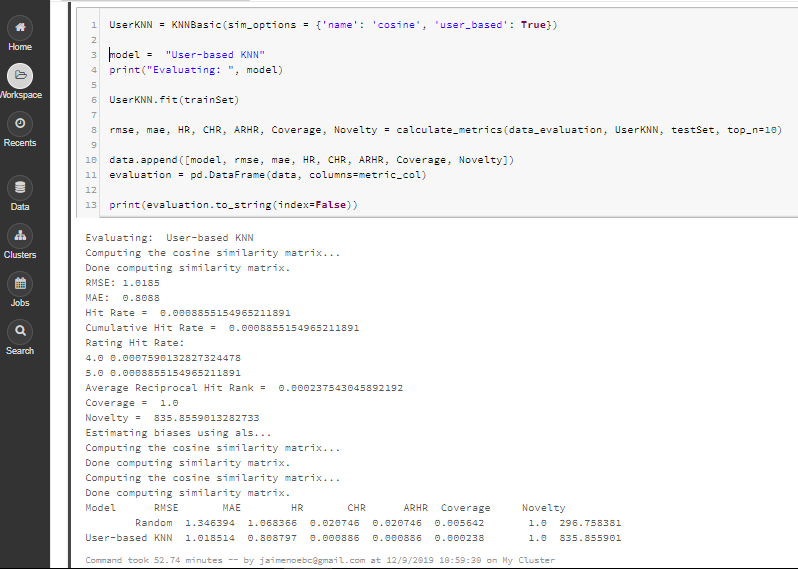


Figura 32: Evaluación User-based KNN

##### Item-based KNN

El algoritmo de Item-based KNN, proporciona unos resultados similares al User-based KNN, es decir, mejora el RMSE respecto al modelo aleatorio, pero, sin embargo, tiene unos ratios de recomendación peores (Hit Rate más bajos), tal y como podemos observar en el siguiente pantallazo de la ejecución:

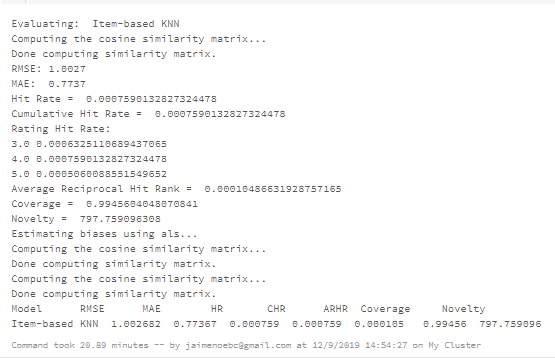


Figura 33: Evaluación Item-based KNN

##### SVD

El algoritmo de SVD (Singular Value Decomposition) se caracteriza por ser más eficiente que otros algoritmos de aprendizaje automático típicamente utilizados en sistemas de recomendación y que además proporciona unos buenos resultados en términos de RMS (batiendo claramente a otros modelos analizados anteriormente (Random, Item-based KNN, User-based KNN) tanto en términos de error (RMSE) como en términos de recomendación (Hit Rate, ARHR, etc). Además, los tiempos de ejecución son menores por lo que sería uno de nuestros candidatos favoritos para nuestro modelo de sistema de recomendación de libros. Los resultados obtenidos para este algoritmo son los siguientes:

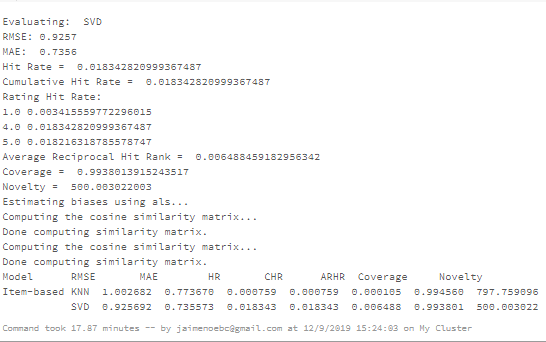


Figura 34: Resultados ejecución SVD

##### SVD++

El SVD++ es una extensión del algoritmo "SVD" pero teniendo en cuenta los ratings implícitos, por ello sus resultados son muy similares al SVD tanto en términos de eficiencia (velocidad de la ejecución) como de eficacia (RMSE y Hit Rate), tal y como podemos ver:

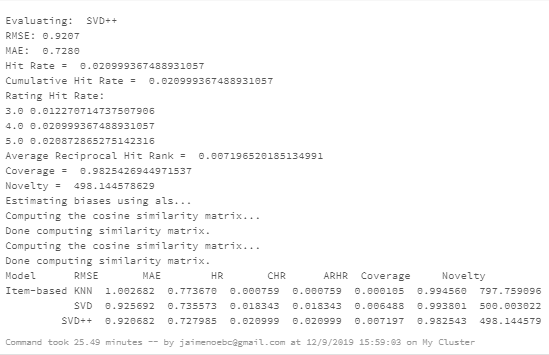


Figura 35: Resultados ejecución SVD++

#### Conclusiones

Como se puede observar a través de los datos de las métricas de evaluación los resultados son similares a los obtenidos procesando los datos localmente con un dataset menor ya que prácticamente sólo cambia el tiempo de ejecución que va desde los pocos segundos (20s segundos – 1 min) que tarda la ejecución de cada algoritmo en local hasta los varios minutos (20-50 min) que tarda cada ejecución en el DataBricks, debido principalmente a que el tamaño del dataset es mucho mayor (el dataset de ratings contiene 10.000 registros en local y 100.000 en el DataBricks). Conviene recordar que no es posible trabajar con el dataset de mayor tamaño en local ya que Python utiliza toda la memoria RAM disponible hasta dejar inutilizado completamente el ordenador (Python y Jupyter no son capaces de gestionar y limitar la memoria RAM disponible por lo que utilizará toda la que el programa y los datos le soliciten hasta no poder continuar o termine satisfactoriamente).

La comparación de las principales métricas de evaluación utilizando DataBricks y un mayor volumen de datos son los siguientes:

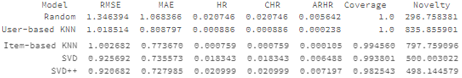


Figura 36: Comparación resultados algoritmos

Como se puede ver el algoritmo SVD++ es el que mejores resultados produce en términos de RMSE (al igual que en el caso del dataset de menor tamaño), pero en este caso el SVD ++ también es mejor en términos de Hit Rate (al contrario que en el caso anterior que el mejor era el SVD, aunque en ambos casos los resultados son muy similares (especialmente entre el SVD y el SVD++).

En cuanto a el Top 10 recomendaciones también tiene bastante sentido tal ya que se trata de libros de ciencia ficción y fantasía (similares a los libros con buenos ratings para el usuario analizar) tal y como podemos ver a continuación:

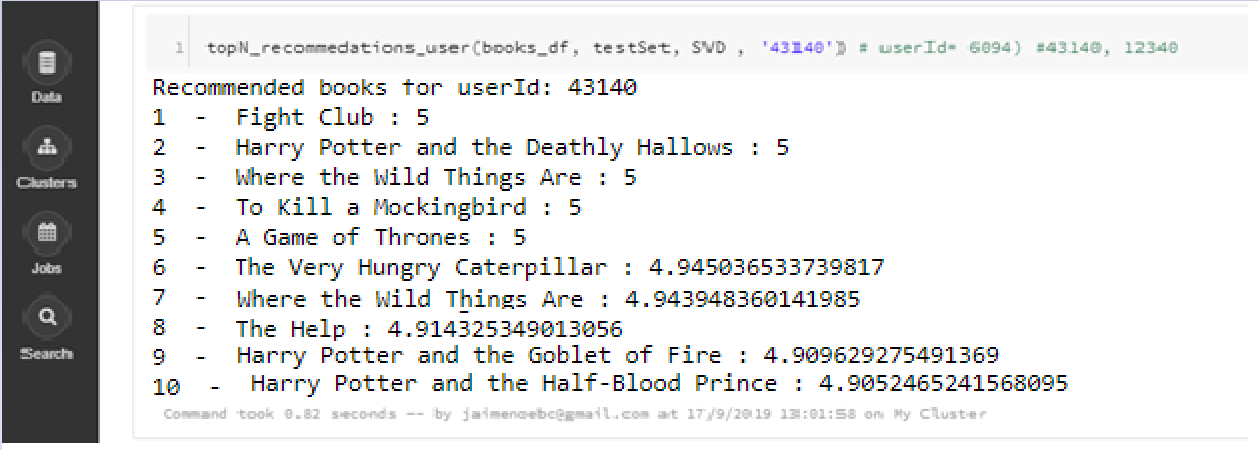


Figura 37: Top 10 recomendaciones para usuario de ejemplo utilizando DataBricks

Capítulo 4

# Conclusiones



## Conclusiones y aportaciones

Los sistemas de recomendación son sistemas complejos que generan matrices muy dispersas para la representación de usuarios, libros y ratings. Esto hace que el cálculo de algoritmos utilizados para la mayoría de los sistemas recomendadores requiera de gran potencia computacional incluso para dataset de tamaño pequeño o mediano.

En el caso de los libros, los datos son aún más dispersos que para otros sistemas de recomendaciones como las películas, ya que evidentemente existen muchos más libros que películas, por lo que se requerirán de datasets de gran tamaño para poder establecer las relaciones usuario-libro-rating.

A la hora de trabajar con datasets de gran tamaño se recomienda utilizar los servicios Cloud, principalmente para el procesamiento y cálculo de algoritmos, siendo DataBricks la plataforma utilizada en una parte de este TFM. DataBricks es una empresa privada y como la mayoría de los servicios Cloud cuenta con suscripciones y servicios de Pay-per-use para la utilización de sus clústers de programación distribuida (como Spark), lo que permite programación distribuida y cálculos en memoria y además cuenta con APIs de alto nivel para poder utilizar Python sobre Spark (y no sólo Scala). Se recomienda el uso de Python ya que cuenta con un mayor número de librería de machine learning y aprendizaje automático (tales como “surprise”) lo que facilita la implementación de muchos algoritmos de recomendación. Adicionalmente la API está desarrollada de tal manera que resulta óptima para la ejecución del DAG (Directed Acyclic Graph) sobre los clústers de DataBricks.

En cuanto a las aportaciones del presente TFM han sido las siguientes:

* 1. Estudio de mercado: lo primero que hemos visto al empezar a desarrollar el proyecto es la nula o muy baja existencia de sistemas de recomendación de libros, así como dataset de estos. Esto no sucede con otros sistemas como de películas, utilizado ampliamente para el estudio de los sistemas de recomendación. Tal y como veremos más adelante, esto será bueno para el desarrollo de nuestro Business Case.
  2. En cuanto a las conclusiones obtenidas respecto al análisis de los datos, hemos podido ver como los usuarios tienen un sesgo positivo a la hora de valorar los libros que han leído, y esto se debe principalmente a que se tarda mucho más en leer un libro que en ver una película por lo que, en general, alguien sólo leerá un libro si cree que este le puede gustar. Hasta la fecha la única forma de recomendar libros era mediante la confianza de otra persona de nuestro entorno con la que creemos tener gustos similares, o en los que podemos confiar en su criterio o mediante los llamados “Top-sellers” donde se asume que aquellos libros muy leídos / vendidos van a gustar a la mayoría de los usuarios. Esto se demostró viendo que existe una correlación positiva entre en número de veces que se ha votado un libro y su puntuación media. Aunque esto es cierto para la mayoría de los usuarios, no es válido para aquellos usuarios que se salen de “lo normal”. Por este motivo, nuestro sistema de recomendación de libros realizará una aportación importante a la hora de recomendar libros para gente con gustos específicos.
  3. Hemos podido comprobar que el número medio de libros leídos por los usuarios oscila entre 30 y 130, es decir, bastante menor que el número medio de otros datasets como el de las películas ya que como hemos mencionado anteriormente en la sociedad actual es más rápido y probable leer un libro que ver una película.
  4. Para este dataset se ha trabajado con un dataset de libros en inglés, en los que libros que aparezcan en otros idiomas han sido descartados, aunque si bien estos eran la minoría. Es importante tener en cuenta que puede haber cuestiones culturales que pueden hacer que los resultados obtenidos no sean directamente extrapolables para otros idiomas.
  5. Hemos vistos como el año de publicación del libro no tiene relación con su rating medio, es decir, que los libros actuales no son considerados mejores para el público objetivo de la actualidad, pero tampoco al revés.
  6. También hemos podido observar cómo los libros más leídos (votados) son aquellos que tienen su equivalente en película (Harry Potter, Hunger Games, etc.) por lo que el hecho de que un libro sea llevado a la gran pantalla ya nos está diciendo que dicho libro ha sido un éxito en términos de ventas (lecturas) y ambas cosas, libro y película, se complementan desembocando en una simbiosis múltiple.
  7. De la mayoría de las características de un libro hemos podido comprobar como los géneros son los que tienen un mayor poder predictivo y por eso ha sido la concatenación de éstos lo que se ha utilizado para el entrenamiento, cálculo y predicción de algoritmos de recomendación. Sin embargo, el término “género” es difícil de asignar a un determinado libro, de hecho, en el dataset original no existía esta característica, sino un campo “tag”, texto libre, done el usuario escribía todo lo que pensaba y/o quería decir en relación con el libro que acababa de valorar. El primer problema al que hemos tenido que enfrentarnos ha sido la normalización de dichos “tags” para traducirlas en géneros. Adicionalmente diferentes usuarios pueden clasificar los mismos libros con diferentes géneros, por lo que para obtener todos los géneros a los que puede pertenecer un libro será necesario la concatenación de éstos. Es importante asignar un número mínimo de coincidencias ya que de otra forma un usuario podría cambiar los géneros de un libro según su propio criterio. También por este motivo, para la parte de entrenamiento de nuestros algoritmos se han seleccionado sólo aquellos libros con más de 10 votaciones y aquellos usuarios que han votado al menos 10 libros. Esta última parte no se tiene en cuenta a la hora de las predicciones.
  8. Otra cosa que hay que tener en cuenta a la hora de implementar los sistemas de recomendación es que no sólo influyen las métricas clásicas de aprendizaje automático (que también) como el RSME, sino otras más propias de dichos sistemas de recomendación tales como el Hit Rate, Average Ranking Hit Rate, cobertura o diversidad, en función de los resultados buscados, ya que a veces querremos modelos más conservadores o más innovadores.
  9. Adicionalmente se debe prestar menos atención a los rankings bajos a la hora de recomendar libros a usuarios similares, ya que el hecho de que dos usuarios odien los mismos libros no significa que a ambos les van a gustar los mismo.
  10. Los resultados obtenidos en términos de predicción y evaluación de modelos, hemos podido ver como los algoritmos de SVD y SVD++ son los que obtienen mejores resultados tanto en error cuadrático medio (RMSE – menor que 1 star), como en Hit Rate y similares. Adicionalmente son más rápidos que otros como las redes neuronales (RBMs) por lo que serán los que utilizaremos, ya que este sistema podría ser utilizado como piedra angular de un sistema recomendador real tal y como veremos en la parte de “Business Case” donde dicho sistema recomendador deberá proporcionar las Top-N recomendaciones a golpe de click (una vez el usuario ha valorado un número mínimo de libros).
  11. Los resultados de un dataset de menor tamaño (10.000 valoraciones) son extrapolables a los resultados obtenidos utilizando otro de mayor tamaño (100.000 ratings) si éste ha sido elegido de manera cuidadosa para ser una muestra representativa de los datos.
  12. Los sistemas de Big Data tales como Azure Data Lake y DataBricks son necesarios y los seguirán siendo en el futuro de los sistemas recomendadores, ya que como hemos mencionado anteriormente estos sistemas generan matrices de datos (usuarios/ratings/libros) muy dispersas que hacen que los cálculo de las métricas de evaluación, así como de nuevas recomendaciones sean muy pesado, por lo que el uso de herramientas para el procesamiento distribuido es una buena idea.
  13. Las redes neuronales estudiadas (RBMs – Resitricted Boltzman Machines) funcionan de manera razonablemente (mejor que un sistema recomendador aleatorio), sin embargo, son pesadas de calcular y requieren de la configuración de muchos hiperparámetros para finalmente producir unos resultados peores que otros algoritmos tales como el SVD o el SVD++, por lo que desaconseja el uso de dichas RBMs para nuestro sistema recomendador y tal y como se verá en el apartado de “líneas de trabajo futura” se propone la experimentación con otro tipo de redes neuronales con diferentes topologías y funcionalidades ya que es un campo que tiene resultados prometedores en todas las aplicaciones relacionadas con el aprendizaje automático y reconocimiento de patrones (como podría interpretarse que es el caso de un sistema recomendador de libros)
  14. Finalmente, y para comprobar que efectivamente nuestro sistema recomendador funciona correctamente se han elegido varios usuarios del dataset con gustos parecidos a los del investigador, tanto en libros leídos como en ratings, uno de dichos usuarios con votaciones reales. Posteriormente se han calculado la lista de Top-10 recomendaciones y se ha podido comprobar empíricamente como dicha lista concuerda con los libros que el investigador podría leer (o que ya ha leído pero que aún no ha votado) lo que otorga bastante credibilidad a nuestro sistema recomendador.

## Líneas de trabajo futuras

La primera línea de investigación futura debería centrarse en la aplicación de nuevos modelos de redes neuronales y no solamente la utilización de las RBMs, concretamente la utilización de las DeepFM parece una buena línea de investigación futura, ya que tal y como se especifica en el paper [9] los resultados son bastante prometedores en el cálculo del CTR y quizás puedan ser extrapolables a los sistemas de recomendación. Para ello también es recomendable utilizar un clúster de pago del DataBricks Community ya que las redes neuronales necesitan una gran potencia computacional.

Otra línea de trabajo futura sería mediante la utilización de otros dataset con diferentes lenguajes, ya que quizás los algoritmos óptimos de recomendación sean diferentes en función del lenguaje (así como de la cultura).

Adicionalmente se podría poder trabajar con datasets de mayor tamaño en cuanto a las características de los libros y de esta forma aumentar la parte del análisis exploratorio de datos con el fin de descubrir nuevas relaciones entre las nuevas features y la variable objetivo (rating para cada usuario). Quizás características tales como el número de páginas son relevantes a la hora de predecir un rating pues habrá usuarios que prefieran leer libros de menor tamaño y otros que se decantarán, por lo contrario. El uso de nuevos datasets con nuevas características puede implicar cambios en los algoritmos predictores.

## Business Case

Aunque el business case podría tratarse como una línea de trabajo futura, se va a tratar en un epígrafe propio ya que es cierta importancia para el desarrollo del Máster de Business Analytics MBI.

Esta línea de trabajo futura tendría un enfoque más de negocio, es decir, la implementación de una página web para que los usuarios pudiesen votar los libros que deseen y en función de estas valoraciones proporcionarles predicciones, es decir, algo parecido a lo que ofrecen páginas como “FilmAffinity” Pero para los libros en lugar de películas.

Esta solución requeriría de estrategias de posicionamiento de web, implementación del front-end así como una API para comunicarse con nuestro sistema de back-end de cálculo de predicciones, los cuales debería ser rápidos en las predicciones (quizás también en el cálculo de modelos para usuarios primerizos) ya que los usuarios seguramente no quieran estar esperando mucho tiempo a que el sistema les proporcione las lista de libros recomendados.

Esto seguramente implique la no utilización de redes neuronales ya que estas suelen ser más lentas, aunque como hemos visto los algoritmos SVD y SVD++ producen muy buenos resultados y además son muy rápidas.

En este aspecto, deberemos tener cuidado con la utilización de imágenes de imágenes protegidas para los libros en los que los usuarios puedan clicar para seleccionar el rating.

Adicionalmente sería bueno incluir otras opciones en nuestra página web ya que solamente un sistema de valoraciones podría no tener el gancho necesario y podría ser buena idea incluir opciones extras como las listas de libros con mejores valoraciones globales, o los mejore libros según el género (mejores libros de comedia, ciencia ficción, etc.), los libros más veces leídos / valorados o los libros del último año / década (y / o por género). La posibilidad de ver libros de los mismos autores, etc.

La idea de este apartado es desarrollar cuantitativamente el caso de negocio mediante el análisis de costes, ingresos y márgenes.

Lo primero que deberemos tener en cuenta es que nadie o casi nadie va a pagar por una recomendación de libros si encima tienen que “trabajar” primero valorando libros dentro de nuestra web. Los usuarios no están acostumbrado a este tipo de pagos (existen varias páginas de recomendación de libros totalmente gratuitas tales como “FilmAffinity” o “IMDB” lo que hace que los usuarios no quieran pagar por un sistema que les proporciones lista de recomendaciones de libros).

Una forma de ingresos sería mediante la adición de publicidad (copiando el modelo de las webs de películas mencionadas anteriormente) pero si bien es aconsejable que antes de monetizar nuestra web adquiramos una audiencia elevada, para ello puede ser recomendable la creación de un blog y de un canal de Youtube utilizando el marketing de contenidos. Dicho blog y canal tendrán las siguientes características.

1. Blog: Creamos un blog donde vamos a utilizar la táctica de marketing de contenido. En el blog no haremos publicidad durante los primeros meses, mientras ganamos subscriptores. Todas las semanas publicaremos uno o varios artículos comentamos libros, preferiblemente nuevos. Intentaremos darle visibilidad nuestro blog enviando las noticias agregadoras como menéame, donde nos hemos creado varias cuentas para intentar votar positivo a nuestros envíos (para evitar ser baneados debemos crear las cuentas días antes de enviar nuestro primer envío y tener actividad en el blog). Este blog puede estará alojado dentro de nuestra página web.
2. Canal de Youtube: La misma idea del marketing de contenidos que en blog, pero con un enfoque ligeramente diferente, ya la idea es comentar los libros más destacadas de la semana. Se haría un video a la semana con resúmenes de los nuevos libros, nos interesa que nos futuros lectores. Una vez tengamos suficientes seguidores en el canal, sacaremos nuestras primeras promociones para redirigir clientes a nuestra web.

El coste del canal de Youtube será 0€ (simplemente habrá que invertir cierto tiempo del desarrollador) para que dicho canal para que esté activo y sea visitado por el mayor número de usuarios posibles.

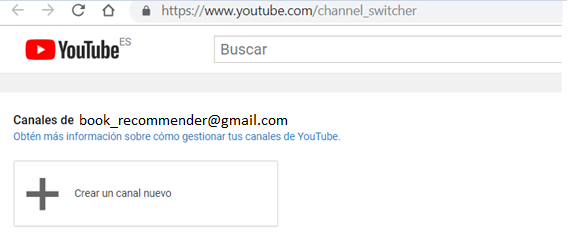


Figura 38: Creación canal de Youtube

Uno de los costes que generará este business case será el ser el del servidor de alojamiento, ya que, aunque el costo de desarrollo de web sea gratuito, necesitaremos un servidor para alojar dicha web. Para ello se ha decidido usar [Google Cloud](https://cloud.google.com/) (3) con un precio de 1.219,82$ (1.092,69€) durante 3 años, ya que, dentro de las ofertas disponibles en el cloud, la de GCP es la más barata:

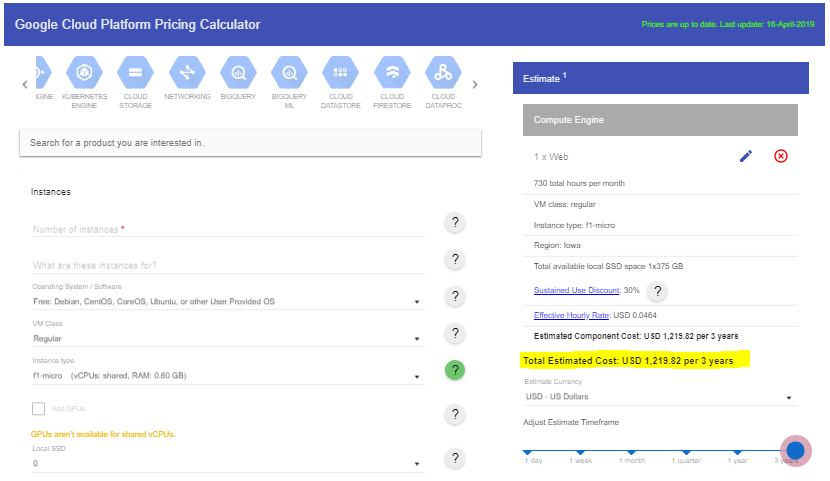


Figura 39: Costes Servidor alojamiento Google Cloud [[3]](#footnote-3)

En cuanto al tema de los ingresos como hemos mencionado, lo haremos mediante el uso de publicidad, preferiblemente relacionada con el tema de los libros de forma que un usuario no sólo pueda valorar un libro, sino que también comprarlo. Esto estaría especialmente recomendado para las recomendaciones, ya que son los libros que seguramente nuestros usuarios no han leído aún, pero que puede que quieran leer y por lo tanto comprar. La inclusión de dicha funcionalidad nos proporcionaría unos ingresos extra ya que muchas veces páginas como Amazon o La casa del libro proporcionar un cierto porcentaje si el ítem (libro en nuestro caso) ha sido recomendado o redirigido a partir de un enlace concreto.

Teniendo en cuenta tanto los ingresos como los costes, se puede ver como este caso de negocio no va a tener grandes márgenes sobre todo al principio. Sin embargo, al tratarse de un producto innovador estamos convencidos que tendrá una gran aceptación por parte de la comunidad de internet (si bien quizás necesitaríamos diferentes sistemas recomendadores para distintos idiomas como el español (utilizando otro dataset diferente tanto para la parte del análisis exploratorio de datos como las predicciones)). Por este motivo, otra opción sería crecer hasta que una empresa de mayor tamaño nos comprase.

Capítulo 5

# Presupuesto



## Introducción

Al tratarse de un proyecto relativamente pequeño no se requerirá de grandes equipos, sino simplemente de un investigador principal y un doctor tutor de TFM.

Para la realización de dicho presupuesto se debe tener en cuenta no solo el tiempo empleado, sino también los materiales (compra de equipos, alquileres, etc)

Lo primero que haremos será visualizar el plan de trabajo para entender el tiempo empleado para la realización de dicho proyecto, desglosado en sus diferentes secciones (tales como recogida y análisis de datos, procesamiento, evaluación, etc) lo que facilitará futuras investigaciones al proporcionar estimaciones de los tiempos empleados para cada una de ellas. Adicionalmente se puede observar aquellas áreas con margen de mejora (presumiblemente aquellas a las que se haya dedicado menos tiempo).

## Cronograma del plan de trabajo

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tarea | Week 1-2 | Week 3-4 | Week 5-6 | Week 7-8 | Week 9-10 | Week 11-12 | Week 13-14 | Week 15-16 | Week 17-18 | Week 19-20 | Week 21-21 | Week 23-24 |
| Estudio estado del arte |  |  |  |  |  |  |  |  |  |  |  |  |
| Recolección de datos |  |  |  |  |  |  |  |  |  |  |  |  |
| Limpieza del dato: Tratamiento de anomalías y valores faltantes |  |  |  |  |  |  |  |  |  |  |  |  |
| Aplicación de algoritmos clásicos |  |  |  |  |  |  |  |  |  |  |  |  |
| Evaluación algoritmos clásicos |  |  |  |  |  |  |  |  |  |  |  |  |
| Aplicación redes neuronales y Deep Learning |  |  |  |  |  |  |  |  |  |  |  |  |
| Evaluación algoritmos Deep Learning |  |  |  |  |  |  |  |  |  |  |  |  |
| Comparación algoritmos |  |  |  |  |  |  |  |  |  |  |  |  |
| Consideración de otras métricas (Hit Rate, ACHR, RHR vs RMSE) |  |  |  |  |  |  |  |  |  |  |  |  |
| Comparaciones nuevas métricas |  |  |  |  |  |  |  |  |  |  |  |  |
| Big Data |  |  |  |  |  |  |  |  |  |  |  |  |
| Conclusiones y Resultados |  |  |  |  |  |  |  |  |  |  |  |  |
| Memoria |  |  |  |  |  |  |  |  |  |  |  |  |

Figura 40: Diagrama plan de trabajo

Como se puede observar el proyecto se ha completado en 24 semanas repartidas entre cada una de las fases típicas de un proyecto de aprendizaje automático incluyendo desde el tiempo empleado para la documentación para la base teórica del estado del arte de los sistemas de recomendación así como la recolección de los datos, el análisis de éstos y la implementación y evaluación de los diferentes algoritmos usados para el sistema recomendador de libros, todo ello explicado detalladamente en el presente documento.

## Presupuesto

En este último apartado se incluye el presupuesto total del proyecto, desglosado en sus diferentes secciones.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **CONCEPTO** | **NÚMERO** | **COSTE** | **TOTAL** | **JUSTIFICACIÓN DEL GASTO** |
| **COSTES DIRECTOS** | **GASTOS DE EJECUCIÓN** | ADQUISICIÓN DE MATERIAL INVENTARIABLE | | | | |
| Ordenadores | 1 | 1000 | 1000 | Necesidad de ordenadores razonablemente potentes para el cálculo de algoritmos de Machine Learning |
| Programas de software | 1 | 100 | 100 | Se utilizará programas open source cuando se pueda y sino programa con licencia como Office u otros |
| ALQUILER DE MATERIAL INVENTARIABLE | | | | |
| Azure Data Lake & DataBricks (almacenamiento y procesamiento en el cloud) | 1 | 150 | 150 | DataBricks utilizaremos la versión community (gratuita) y DataLake tenemos un crédito de 150 € por ser estudiantes |
| MATERIAL FUNGIBLE | | | | |
| Material de oficina | 1 | 100 | 100 | Papel, fotocopias, post-it, carpetas, encuadernaciones, rotuladores, etc |
| **GASTOS DE PERSONAL** | SALARIOS | | | | |
| Salario Ingeniero / mes = 6 meses al 50 % | 3 | 2694 | 8082 | 1 ingeniero para desarrollar el TFM durante 6 meses con una dedicación del 50% cada mes |
| Salario Doctor/tutor (mes) = 6 meses al 10 % | 0.6 | 4289 | 2573.4 | 1 Doctor tutor para supervisar el TFM durante 6 meses con una dedicación del 10% cada mes |
| TOTAL NETO DEL PROYECTO |  |  |  |  | 12005.4 |  |
| COSTES INDIRECTOS | 21% DEL COSTE DEL PROYECTO |  |  |  | 2521.13 |  |
| **TOTAL DEL PROYECTO** |  |  |  |  | **14526.53** |  |

Se concluye finalmente que el presupuesto total de este proyecto asciende a la cantidad de 14.526,53 Euros.

# Bibliografía

|  |  |
| --- | --- |
| [1] | C. Fernández Musoles, J.M. Gomez Hidalgo. “Apuntes Análisis y aprendizaje automático”. Máster Big Data Analytics. UEM: 2018-2019 |
| [2] | F.Kane*.* “*Building Recommender Systems with Machine Learning and AI*”. Sundog education.Udemy. Disponible en web: <https://www.udemy.com/building-recommender-systems-with-machine-learning-and-ai/> [último acceso, 17 de Septiembre de 2019]. |
| [3] | R. Rakesh. “Evaluating Recommendation Systems”. Disponible en web: [<https://medium.com/fnplus/evaluating-recommender-systems-with-python-code-ae0c370c90be>/](https://www.udemy.com/building-recommender-systems-with-machine-learning-and-ai/) [último acceso, 17 de Septiembre de 2019]. |
| [4] | P. Spachtholz. “*Book* *Recommender: Collaborative Filtering, Shiny*”. Disponible en web: <https://www.kaggle.com/philippsp/book-recommender-collaborative-filtering-shiny> [último acceso, 17 de Septiembre de 2019]. |
| [5] | R. Salakhutdinov, A. Mnih, G.Hinton. “Restricted Boltzmann Machines for Collaborative Filtering”. University of Toronto, 6 King’s College Rd |
| [6] | X. Amatrainin, J.Basilico. “*Netflix recommendations: beyond the 5 stars”.* Disponible en web: <https://medium.com/netflix-techblog/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429> [último acceso, 17 de Septiembre de 2019]. |
| [7] | K. Chung. “*Generation recommendations at Amazon scale with Apache Spark and Amazon DSSTNE*”. Jul 2016. Disponible en web: <https://aws.amazon.com/es/blogs/big-data/generating-recommendations-at-amazon-scale-with-apache-spark-and-amazon-dsstne/> [último acceso, 17 de Septiembre de 2019]. |
| [8] | S. Rendle. “*Factorization Machines*”. Osaka University, Japan. pp. 1–6 |
| [9] | H.Guo , R. Tang, Y. Ye , Z. Li , X. He “*DeepFM: A Factorization-Machine based Neural Network for CTR Prediction*”. Noah’s Ark Research Lab, Huawei, China. Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17) |
| [10] | M.E. Wall, A. Rechtsteiner, L.M. Rocha. “*Singular value decomposition and principal component analysis*”. Los Alamos, New Mexico 87545, USA |
|  |  |

1. (\* <https://www.40defiebre.com/que-es/long-tail>) [↑](#footnote-ref-1)
2. (i7-8750H con 16 GB RAM y GTHX 1050Ti 4GB) [↑](#footnote-ref-2)
3. [*https://cloud.google.com/*](https://cloud.google.com/) [↑](#footnote-ref-3)