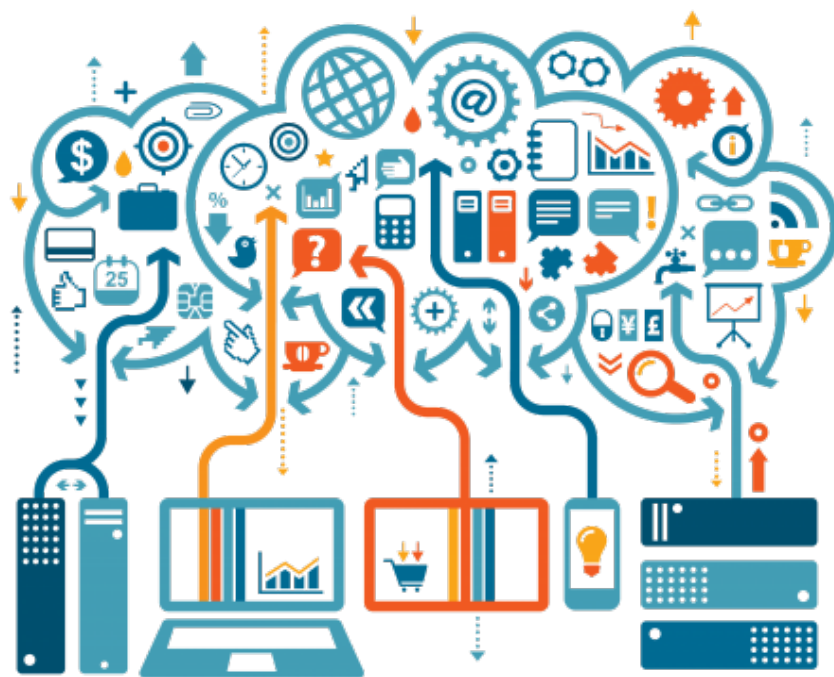


RECOMENDADOR DE PELÍCULAS

Raul Pingarrón
raul.ping4rr0n@gmail.com



APACHE SPARK

Introducción

Este ejemplo muestra cómo implementar y validar un sistema de recomendación basado en **filtrado colaborativo** utilizando el dataset de MovieLens compuesto por:

- `movies.dat`, contiene el listado de 3883 películas siendo el formato de éste fichero como sigue `movieID::Title::Genres`
- `ratings.dat`, contiene la matriz de más de medio millón de valoraciones de películas por distintos usuarios, con formato `userID::movieID::rating::timestamp`

Notar que, dado el carácter ilustrativo del ejemplo, no se ha implementado un mecanismo exhaustivo de validación del modelo (validación cruzada de n iteraciones, etc). Además, para la validación del mismo se han utilizado métricas muy sencillas y directas (estimación del error medio absoluto y estimación del error medido sobre la recomendación final mostrada).

Descripción del algoritmo utilizado y sus parámetros

Para implementar el modelo de recomendación se ha utilizado una aproximación basada en **filtrado colaborativo**. De esta manera la recomendación para un usuario se realiza basándose en los usuarios que más se parecen a él y recomendándole los ítems que mejor han valorado esos usuarios. Los usuarios son similares si sus vectores de características están cercanos entre sí de acuerdo con una medida de distancia.

En concreto se ha utilizado el **algoritmo ALS** (Alternating Least Squares) que proporciona la librería **MLlib** de **SPARK** (disponible para Scala, Java y Python). Dicho algoritmo intenta encontrar factores o características latentes en los datos que determinan cómo un usuario evalúa un producto y para ello, en lugar de utilizar el método SVD (Singular Value Decomposition), utiliza el método de factorización de matrices (ya que en la matriz de ratings usuario-producto va a haber ratings no presentes que SVD no es capaz de ignorar).

Básicamente ALS va modelando la matriz dispersa de ratings de usuario-producto mediante el producto de dos sub-matrices densas de factores: una para usuarios y otra para productos. Estas matrices de factores representan las características latentes que están ocultas en los datos y que el algoritmo trata de “descubrir” para poder realizar la recomendación.

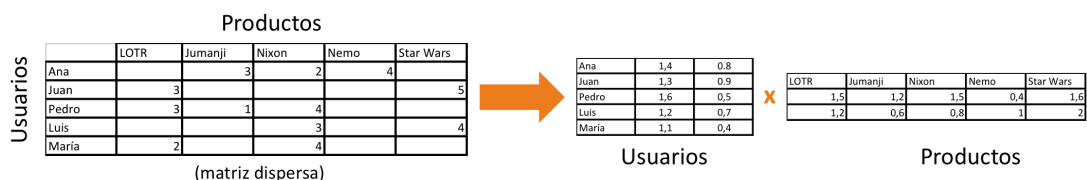


Figura 1: Factorización de matrices en ALS

En este caso, con el dataset que se tiene, si dos usuarios valoran una misma película con un valor elevado es porque puede que les guste el género de la misma (o que ése sea su género favorito). Este sería el factor latente y al descubrirlo podríamos ser capaces de realizar una buena recomendación ya que las características del usuario estarían cercanas a las del producto (la película en este caso).

El algoritmo ALS es iterativo de manera que en cada iteración fija una matriz de factores y resuelve (como un problema de mínimos cuadrados) para la otra de manera alternativa en cada iteración, y este proceso continúa hasta que el algoritmo converge (de ahí su nombre de *Alternating Least Squares*).

ALS es, además, un algoritmo que permite la paralelización masiva y que funciona muy bien con matrices muy densas. Esto lo hace ideal para utilizarlo con un framework de procesamiento masivamente distribuido y paralelo como Apache Spark, como el que se ha utilizado.

Parámetros importantes del algoritmo ALS que vamos a variar para evaluar el modelo:

- **rank:** *especifica el número de factores o características latentes en el modelo. Este número se presume, es decir, ya que en realidad no vamos a saber a priori cuantos factores latentes vamos a tener en el dataset tenemos que averiguarlos. Se supone que cuantos más factores latentes se utilicen mejor será la predicción (hasta un punto determinado para no producir overfitting).*
- **iterations:** *número de iteraciones que el algoritmo realizará. Normalmente el algoritmo converge a una solución razonable en 20 iteraciones o menos.*
- **lambda:** *es el parámetro de regularización del algoritmo. **Una de las bondades de ALS es que hace uso de la técnica de la regularización para reducir el sobreajuste**, ya que el overfitting es uno de los principales problemas que nos podemos encontrar en los sistemas de recomendación. La regularización consiste en introducir un factor que varía la función de coste y que reduce la importancia o quita peso a las características latentes.*

El paper del algoritmo se puede leer en el siguiente enlace:

<https://datajobs.com/data-science-repo/Recommender-Systems-%5BNetflix%5D.pdf>

Medida del error utilizada para la validación del modelo

La librería MLlib de Apache SPARK incluye una serie de métricas de evaluación que se pueden utilizar para los distintos algoritmos que incorpora. En concreto, para validar un modelo basado en ALS y comparar sobre la recomendación final mostrada podemos utilizar la clase RegressionMetrics que implementa métricas de evaluación como RMSE, el MSE (o ECM), el **MAE (o Error Medio Absoluto)**, o el coeficiente de determinación (o R cuadrado).

En enlace con las llamadas a la clase se puede encontrar en:

<https://spark.apache.org/docs/2.1.0/api/python/pyspark.mllib.html#pyspark.mllib.evaluation.RegressionMetrics>


El MAE expresa la diferencia más directa entre dos variables continuas (como los ratings de las predicciones) y por ello es fácilmente interpretable; es decir, el MAE nos va a indicar cuán de cerca va a estar la predicción sobre el resultado final.

Otra de las cosas que se han hecho en la fase de carga y parseado de los datos es filtrar del fichero ratings_all.txt aquellas valoraciones que tienen un 0 (ya que significa que son películas no valoradas) y también he decidido dividir el dataset en 70% para entrenamiento, 20% para validación y 10% para test.

Experimentación

Se han ejecutado varias variantes del código correspondientes a variaciones de los hiper-parámetros del modelo bajo un clúster de Spark2 bajo YARN (distribución HDP 2.6):

```
spark-submit --num-executors 4 RecomendadorALS_MAE_lambda-0.1.py
```



All Applications

Cluster

- About
- Nodes
- Node Labels
- Applications
- NEW
- NEW SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- FAILED
- KILLED
- Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores I
23	0	0	23	0	0 B	160 GB	0 B	0	24	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation
Capacity Scheduler	[MEMORY]	<memory:4096, vCores:1>

Show 20 entries

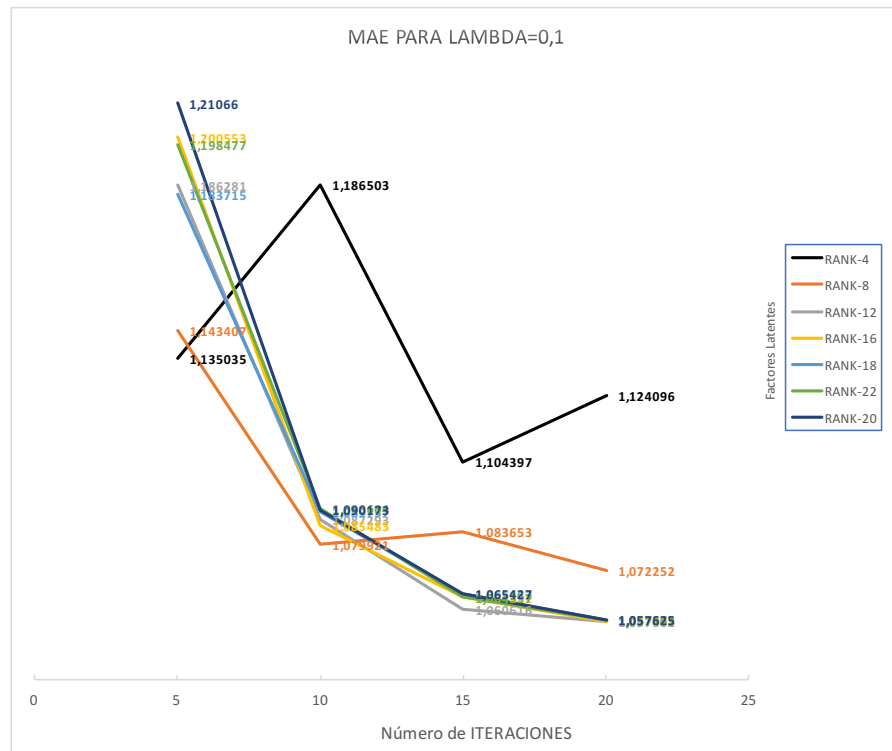
ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus
application_1495966194023_0023	raul.pingarron	RecomendadorALS_MAE_lambda-1.0	SPARK	default	0	Sun May 28 19:28:03 +0200 2017	Sun May 28 19:29:31 +0200 2017	FINISHED	SUCCEEDED
application_1495966194023_0022	raul.pingarron	RecomendadorALS_MAE_lambda-0.5	SPARK	default	0	Sun May 28 19:27:47 +0200 2017	Sun May 28 19:29:19 +0200 2017	FINISHED	SUCCEEDED
application_1495966194023_0021	raul.pingarron	RecomendadorALS_MAE_lambda-0.1	SPARK	default	0	Sun May 28 19:21:39 +0200 2017	Sun May 28 19:23:12 +0200 2017	FINISHED	SUCCEEDED

Dado que el clúster tiene recursos bastante reducidos se ha tenido que jugar con los parámetros de ejecución y planificación de los jobs (número de ejecutores, memoria de los ejecutores y driver, número de particiones de los RDDs, etc) ya que durante algunas de las ejecuciones se producían situaciones de excepción por OOM perdiéndose ejecutores o llenándose la memoria del programa driver de Spark.

Resultados y conclusiones

Utilizando el error medio absoluto (MAE) para realizar la validación del modelo se observa que el valor parámetro de regularización lambda, introducido en ALS para evitar el sobre-entrenamiento, ha de ser elegido cuidadosamente ya que un valor muy elevado distorsiona la función de coste y da lugar a malos resultados (resultado con sesgo al comparar sobre el conjunto de test final).

A continuación, se muestra la gráfica comparativa del MAE para el modelo con $\lambda=0,1$ visualizando los distintos números de factores latentes e iteraciones del algoritmo:



Dos observaciones a partir de los resultados obtenidos:

- *el algoritmo converge con mayor número de iteraciones*
- *si el número de iteraciones es elevado (a partir de 15) el número de factores latentes a utilizar tiene menos peso (salvo para el caso de Rank=4 donde se observa que ese número de factores latentes es insuficiente).*

Con $\lambda=0,1$ el modelo parece ajustarse bastante bien, aunque como se observa en las gráficas los valores del RMSE son muy cercanos entre los modelos cuando estos tienen un número suficiente de iteraciones en el algoritmo.

Cabe reseñar que, para otros casos particulares, por ejemplo, si queremos tener en cuenta si nuestro modelo ofrece valores de error extremo -- pocos valores de errores muy elevados frente a la gran mayoría que da errores bajos -- el RMSE será una medida de error mejor para la validación.