

Systeme de recommandation

Spark-Scala sur la base de données Movie-Lens

Fait par:
Salma BENNISS
Marwen TOUZI

Plan

Mise en scène

Présentation du contexte

Les systèmes de recommandation

Solution technique

Réalisation

Perspectives et conclusion



Mise en scène

My New Mood est jeune start-up créé en 2015 qui vise l'univers de la nuit.

L'objectif est de créer une application qui vise à inclure un système de recommandation des établissements(bar, restaurant ..).

Dans un premier temps, la start-up nous a fourni une base de données (Movie Lens) afin de s'entraîner et de construire des modèles de recommandation performants, avant d'appliquer nos modèles sur les données réelles de l'entreprise.



Présentation du contexte

L'objectif de ce projet est d'implémenter un système de recommandation qui à partir d'une base de donnée contenant les Ids des utilisateurs, les items et les notes, permet de :

- Retourner un film qu'un utilisateur est susceptible d'aimer.
- Prédire la note qu'un utilisateur peut attribuer à un film.
- Calculer l'erreur du modèle.
- Inclure dans la recommandation les film qui n'ont pas été beaucoup notés.
- Ajouter un nouvel utilisateur.
- fonctionner dans un cadre d'une grande masse de données.

Les systèmes de recommandation (1/7)

Recommandation **non Personnalisée** (naïve) :

- recommander les films les mieux notés.
- recommander les films qui ont eu un maximum de notes.
- recommander les films qui ont un maximum de nombre de vus.



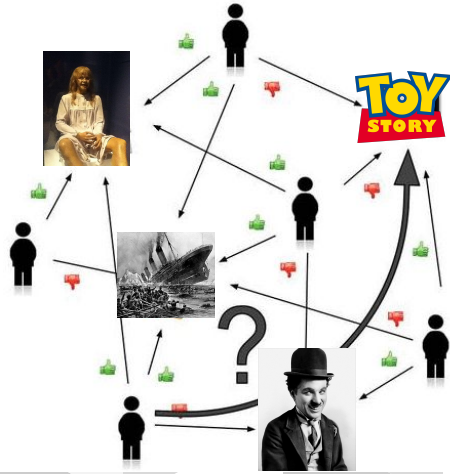
Facile à implémenter.



Ne prend pas en considération ni le goût ni les caractéristiques des utilisateurs.

Les Systèmes de recommandation (2/7)

Recommandation Sociale ou « Collaboratif Filtering »:



Elle se base sur les événements qui lient les utilisateurs avec les items comme les notes, les likes, le nombre de visite de la page de l'établissement...

Les Systèmes de recommandation (3/7)

Recommandation Sociale ou « **Collaboratif Filtering** »:

Memory based(se base sur le principe de similarité):

- User based: « Les utilisateurs qui sont similaires à vous, ont aimé aussi ... »
- Item based: « Les utilisateurs qui ont aimé ça, ont aimé aussi ... »



Facile à implémenter, génère de bon résultats



Ne résout pas les problèmes de cold start, sparsity, et scalability.
Coûteuse en mémoire et temps d'exécution.

Les Systèmes de recommandation (4/7)

Recommandation Sociale ou « **Collaboratif Filtering** »:
Model based(se base sur la factorisation de matrices):

- SVD Singular Vector Decomposition.
- ALS Alternating Least Square.
- SGD Stochastic Gradient Descent.





Plus rapide, résout le problème de sparsity.



Ne résout pas les problèmes de cold start et scalability.

Les Systèmes de recommandation (5/7)

Collaborative Filtering - Model Based

Algorithme	Les avantages 	Les inconvénients 
ALS	<ul style="list-style-type: none">- On peut l'utiliser en parallèle- Plus performant dans un cas non sparse.	<ul style="list-style-type: none">- En général, il est plus lent
SGD	<ul style="list-style-type: none">- Plus facile à implémenter et plus rapide.	<ul style="list-style-type: none">- Plus lent dans le cas d'une matrice sparse.

Les systèmes de recommandation (6/7)

Recommandation objet ou « **Content-based** »:

se base sur les caractéristiques des users/items: En gros, elle calcule la corrélation entre les caractéristiques de tous les items pour trouver l'item le plus corrélé avec l'item sélectionné.

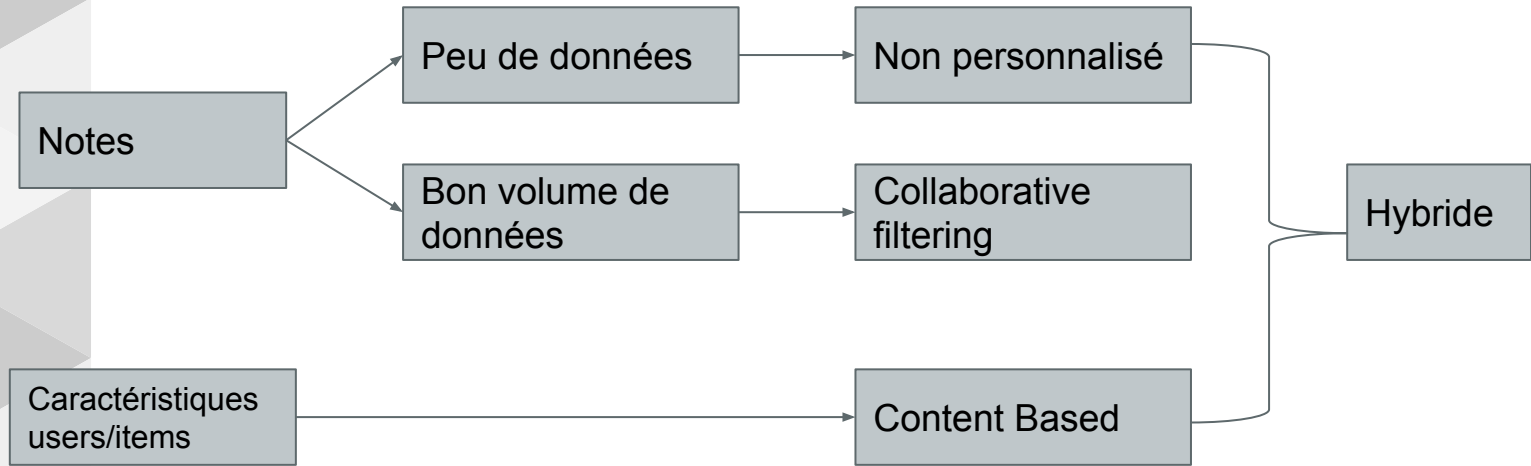


résout le problème de cold start



C'est difficile d'initialiser les 2 matrices des caractéristiques des users/items, on n'a pas résolu le problème de scalability

Les système de recommandation (7/7):



Les systèmes hybrides sont les systèmes les plus utilisés par les géants de web.



Les système de recommandation :

Choix d'algorithmes:

- On ne peut pas utiliser le content based vu qu'on n'a pas les caractéristiques des utilisateurs ou des items.
- La recommandation non personnalisée est facile et déjà implémenter dans la startup.
- Le Collaboratif filtering, Memory based est très coûteux vu qu'il calcule la corrélation entre tous les utilisateurs et entre tous les items à chaque fois qu'on a des nouvelles données.
- Le collaboratif filtering, Model based est plus rapide. On a utilisé l'algorithme ALS car il est le plus scalable et bien documenté avec les technologies qu'on va utiliser.

Solution technique(1/3)



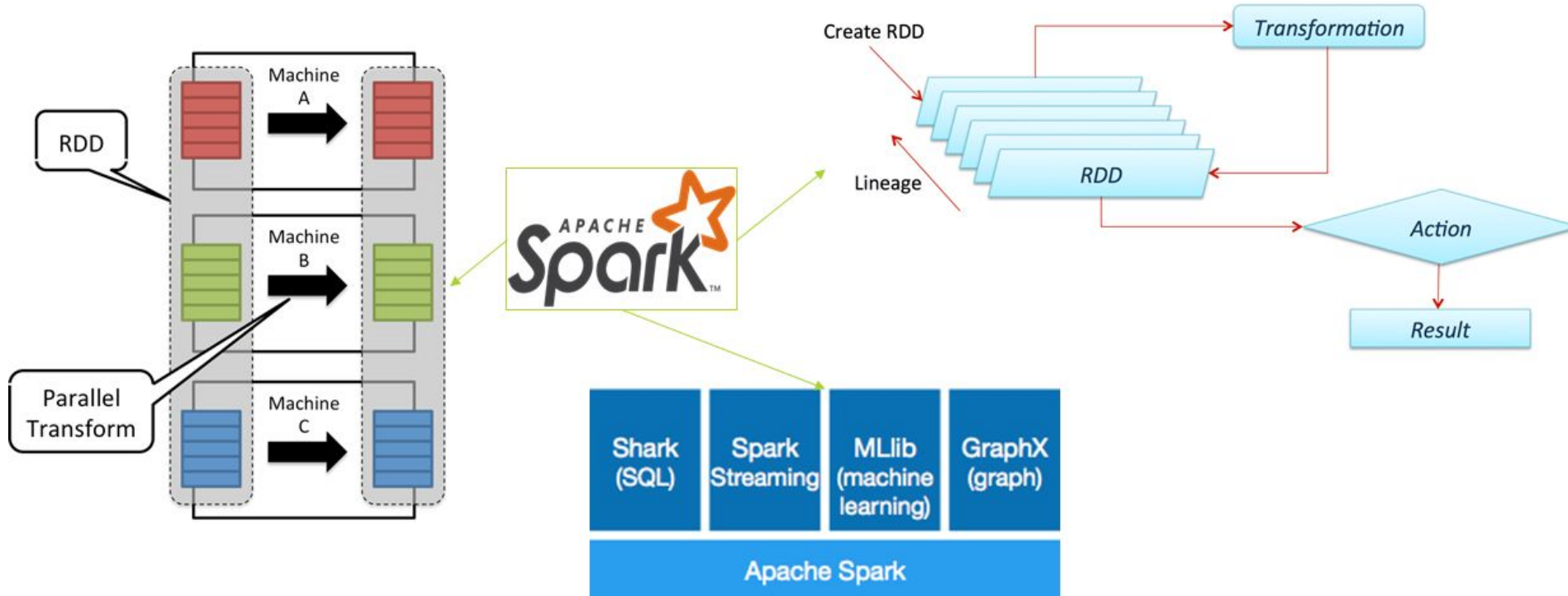
- Scala est un langage de programmation multi paradigme, c'est à dire qui intègre les paradigmes de programmation orienté objet (qui met en avant le changement d'état) et les paradigmes de programmation fonctionnelle (qui met en avant l'application des fonctions).
- Un code Scala est compilé en bytecode et exécutable sous la JVM (Java Virtual Machine). Scala est plus facile à intégrer dans un code java ce qui satisfait le besoin de la startup

Solution technique (2/3)



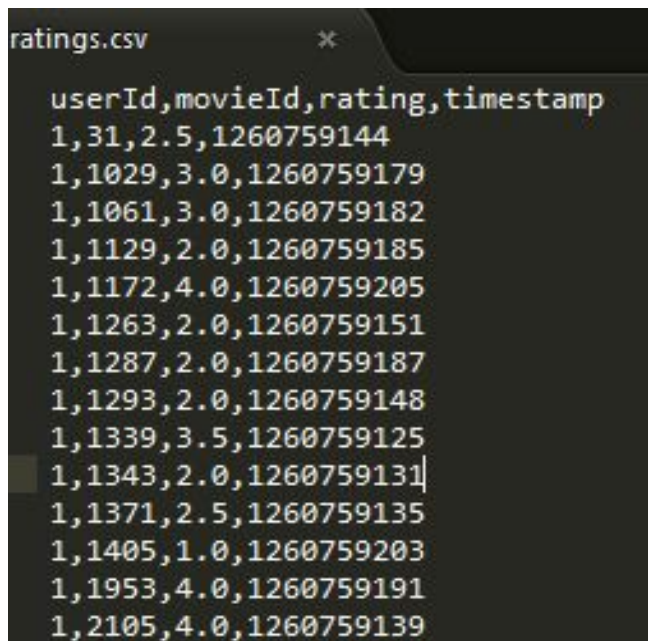
- Spark est un framework open source de calcul distribué qui se base sur la notion des RDD (Resilient Distributed Dataset) qui sont des collections des données. Notons que les opérations sur les RDD suivent implicitement le paradigme “map reduce” (on n’est pas censé programmer la partie du calcul distribué)
- Spark travaille en mémoire vive ce qui est cent fois plus rapide que Hadoop qui tourne sur disques. (On va plus perdre du temps pour transférer les données du disque dur à la mémoire vive, faire les opérations puis stocker les résultats sur le disque dur). Ce qui est nouveau chez Spark, c’est qu’on peut mettre directement des données sur la mémoire cache ce qui réduit énormément le temps d'exécution.

Solution technique (3/3)



Réalisation

Bases de données à disposition



A screenshot of a text editor window titled 'ratings.csv'. The editor displays a CSV file containing movie ratings. The first line is the header: 'userId,movieId,rating,timestamp'. The subsequent lines represent individual ratings, all from user 1, with varying movie IDs, ratings, and timestamps. The text is white on a dark background.

userId	movieId	rating	timestamp
1	31	2.5	1260759144
1	1029	3.0	1260759179
1	1061	3.0	1260759182
1	1129	2.0	1260759185
1	1172	4.0	1260759205
1	1263	2.0	1260759151
1	1287	2.0	1260759187
1	1293	2.0	1260759148
1	1339	3.5	1260759125
1	1343	2.0	1260759131
1	1371	2.5	1260759135
1	1405	1.0	1260759203
1	1953	4.0	1260759191
1	2105	4.0	1260759139

Réalisation ($\frac{1}{3}$)

étape 1: Charger et parser les données (création des RDDs)

étape 2: Collaborative filtering

on utilise la librairie Spark MLlib qui contient l'implémentation de la méthode du Collaborative Filtering utilisant la méthode ALS (Alternating Least Squares)

- Choix de paramètre rank (nombre de variable latentes à créer) du modèle ALS: construction du modèle sur une base de train et le tester sur une base de test

```
In [18]: errors //meilleur err est pour rank==24
```

```
Out[18]: Array(0.4876676022721085, 0.39997350625431854, 0.3481989677701417, 0.288097924016866)
```

Réalisation (2/3)

étape 3: Concentrer la recommandation pour les films les moins notés.

Un des grands problème des systèmes de recommandations est le fait de recommander toujours le 20% de la base qui représente les items les plus populaires (vu qu'il sont les plus notés, il seront les plus corrélés avec les autre items).

```
In [25]: //recomandation for a user:
// we want to recommend movies with minimal number of rates:
def get_counts_and_averages(x:Int,y:Array[Double])={//prend Rating Matrix
  val nratings = y.size
  var s=0.0
  for (elt<-y) {
    s=s+elt
  }
  s=s/nratings
  (x,(nratings, s))
}
```

Réalisation (2/3)

étape 4: Création d'un nouvel utilisateur avec des ratings, et fusionner ce nouveau RDD avec l'ancienne base de données => on utilisera cette fonction pour ajouter des utilisateurs à la base de données, ou pour mettre à jour la base des notes si on a un nouvel événement "review".

```
In [22]: //Adding new user rating id==0 is not used in the data base
val new_user_ID=0
val new_user_ratings = Array(
  (0,260,4), // Star Wars (1977)
  (0,1,3), // Toy Story (1995)
  (0,16,3), // Casino (1995)
  (0,25,4), // Leaving Las Vegas (1995)
  (0,32,4), // Twelve Monkeys (a.k.a. 12 Monkeys) (1995)
  (0,335,1), // Flintstones, The (1994)
  (0,379,1), // Timecop (1994)
  (0,296,3), // Pulp Fiction (1994)
  (0,858,5), // Godfather, The (1972)
  (0,50,4) // Usual Suspects, The (1995)
)
val new_user_ratings_RDD = sc.parallelize(new_user_ratings)//transform array to RDD
```

Réalisation (3/3)

étape 5: Pour un utilisateur donné, afficher les cinq films les plus susceptibles à lui plaire (Parmis les films qui n'a jamais noté) => on utilisera cette fonction dans l'application pour générer les recommandations aux utilisateurs

```
In [59]: val top_movies = new_user_recommendations_rating_title_and_count_RDD_format.filter(x=>x._3>=25)
        .map(x=>(x._1,x._2)).map(item => item.swap).sortByKey(false, 1).map(item => item.swap).take(5)
```

```
In [41]: top_movies.take(5)
```

```
Out[41]: Array((Modern Times (1936),4.265634643048782), (There Will Be Blood (2007),4.2168618496493835), ("Lives of Others,4.18043946618
4994), (Cinema Paradiso (Nuovo cinema Paradiso) (1989),4.143232017416185), ("Third Man,4.127757003770054))
```

Réalisation (3/3)

étape 6: Pour un utilisateur donné et pour un film donné, on peut estimer la note et la comparer avec la note réelle => on utilisera cette fonction dans l'application pour afficher à un utilisateur sur une page d'établissement, la note prédite grâce à notre algorithme, afin de l'inciter à donner sa propre note

```
In [45]: new_ratings_model.predict(0,260)//dans la base c'est 4! //c'est la prediction du notre us=0, it=260
```

```
Out[45]: 3.418232923178837
```

Perspectives

- Utiliser la méthode de Content-based pour faire de la prédiction.
- Ajouter les Out-put suivants :
 - Similarité Utilisateur-Utilisateur.
 - Similarité Item-Item.
- Appliquer nos algorithmes sur la vraie base des établissements.
- Passer en Prod et faire du en-ligne recommandation

Conclusion

La librairie MLlib de spark offre des méthodes d'analyses de données très performant, notamment la méthode ALS du collaborative filtering, Model based qui s'adapte parfaitement à notre système de recommandation.

Naturellement sur des grandes échelles de données, même les méthodes de factorisation de matrice peuvent devenir assez coûteuses puisqu'elles nécessitent la mise à jour du modèle, après l'ajout des utilisateurs, des items et/ou de leurs préférences. D'où l'intérêt de l'utilisation des systèmes de calcul distribué comme Spark qui se base à la fois sur les algorithmes "map reduce" et sur le traitement en mémoire.

Pour assurer et faciliter le passage en production, on peut utiliser des technologies comme le serveur du machine learning Open Source "PredictionIO".