

Techniques et outils d'analyse du Big Data

Les fondamentaux



Sommaire

- **Présentation de Pig**
- **Présentation de Hive**
- **Présentation de Mahout**

Pig



Sommaire pour Pig

- Introduction de Pig
- Les types de données
- Les opérateurs basiques
- Compteur de mots
- Les jointures

Cycle de vie des Big Data avec Hadoop

Collecte

Sqoop
Flume

Stockage

HDFS
Hbase

Traitement

MapReduce
Hive, Pig

Analyse

RHadoop
Mahout
Hive, Pig
Spark
Storm
Solr

Exploration

RHadoop
Mahout
Hive, Pig
Spark
Storm
Solr

Qu'est que Pig ?

- **Pig est un système de traitement des gros volumes de données utilisant MapReduce**
- **Pig comprend 2 modules :**
 - **Pig Latin: langage procédural de haut niveau**
 - **Pig Engine : analyse, optimise et exécute les scripts PigLatin comme une série de jobs MapReduce au sein d'un cluster.**

3 modes pour exécuter des tâches Pig

- Mode interactif
- Mode script
- Mode API

Mode interactif

- Pour interagir avec le shell Pig, entrer la commande:

pig

- Le système va afficher l'écran suivant:

```
$ pig ... Connecting to ...  
grunt>
```


Mode script

- Ensemble d'instructions du langage Pig réunies dans un fichier de type fichier.pig
- Pour exécuter un script Pig Latin sur Hadoop, utiliser la commande:
`pig -x mapreduce fichier.pig`
- Pour exécuter un script Pig Latin en mode local, utiliser la commande:
`pig -x local fichier.pig`

Mode API

- Pig dispose d'une API Java qui permet d'appeler Pig via des programmes Java.
- L'exécution du fichier au moyen de la commande:

```
java -cp <path>pig.jar WordCount
```

2 types de données dans Pig

- **Type scalaire : int, long, float, double, chararray, bytearray**
- **Types complexes:**
 - tuple
 - bag
 - map

Tuple

- Le tuple est une série ordonnée de champs de n'importe quel type
- Tuple: (field1, field2, field3, ...)
ex: (12, John, 5.3)

Bag

- Le bag est une collection de tuple non ordonnée
- Bag: {tuple1, tuple2, tuple3,...}
ex: {(1, Nancy), (2, Reims)}

Map

- Un map est une série de couples (clef#valeur)
- map [clef1#valeur1,clef2#valeur2,...] ex.
['devise'#'EUR', 'montant'#12000]
- La clé est de type chararray, elle est utilisée comme index pour trouver la valeur associée.
- La valeur est considéré comme un «bytearray».

Structure d'un programme Pig

- Un programme Pig contient au moins les étapes suivantes:
 - Chargement des données (LOAD)
 - Traitement sur les données
 - Affichage des résultats (DUMP) / Sauvegarde des données (STORE)
- Les étapes sont parallélisées par MapReduce
- Chaque ligne d'un script PIG finit par un point-virgule ;

Chargement des données

- Les données sont chargées au sein d'un bag.
- La commande de chargement est: **LOAD**

LOAD 'source' [USING fonction] [AS schemas]

- source: nom du/des fichier(s) ou répertoire(s),
- fonction: nom d'une fonction d'importation (ex: PigStorage)
- schemas: un descriptif du format des données importées (ex: (NOM1:TYPE1, NOM2:TYPE2, ...))

Fonction PigStorage (1/2)

- Charge les données sous la forme d'un bag de tuples
- `PigStorage("SEPARATEUR")`. le caractère séparant les différents champs de chaque tuple.
- Chargement du fichier 'etudiants.txt':
`A = LOAD 'etudiants.txt' USING PigStorage('|');`
... alors A contiendra le bag de tuples suivant:

```
DUPONT|André|10  
DURANT|Marc|14  
MARTIN|Christine|10
```



```
(DUPONT,André,10)  
(DURANT,Marc,14)  
(MARTIN,Christine,10)
```

Fonction PigStorage (2/2)

- Si on n'indique pas de fonction alors PigStorage est utilisé avec un séparateur de tabulation.
- Chargement du fichier 'etudiants.txt':
`A = LOAD 'etudiants.txt';`
... alors A contiendra le bag de tuples suivant:

```
DUPONT[TAB]André[TAB]10  
DURANT[TAB]Marc[TAB]14  
MARTIN[TAB]Christine[TAB]10
```



```
(DUPONT,André,10)  
(DURANT,Marc,14)  
(MARTIN,Christine,10)
```

schemas (1/2)

- Une fois le bag des données d'entrée chargé, on peut se référer à différents membres de chaque tuple en utilisant la syntaxe:

\$0, \$1, \$2, etc...

1^{er} membre du tuple

2ème membre du tuple

3ème membre du tuple

- Exemple de données chargées dans A:

(DUPONT,André,10)
(DURANT,Marc,14)
(MARTIN,Christine,10)

« A.\$0 » désigne le nom (DUPONT, ...)

« A.\$2 » désigne la note (10, 14, ...)

schemas (2/2)

- **schemas permet de:**
 - **Donner un nom aux champs des tuples chargés.**
 - **Leur attribuer un type de données explicite.**
- **La syntaxe est: (NOM1:TYPE1, NOM2:TYPE2, ...)**
Le type est optionnel.

Exemple:

```
A = LOAD 'etudiants.txt' USING PigStorage('|') AS  
(nom,prenom,note);
```

Description

- La description de la composition d'un container est obtenu avec commande:

DESCRIBE

- Exemple:

```
A = LOAD 'in.txt' Using PigStorage('|') AS \  
(infos:tuple(nom,prenom),notes:tuple(n1,n2,n3));
```

```
DESCRIBE A;
```

```
A: {infos: (nom: bytearray,prenom: bytearray),notes:  
(n1: bytearray,n2:bytearray,n3: bytearray)}
```

Traitement des données

- Les opérateurs relationnels permettent d'effectuer des opérations sur les données:
 - filtrer,
 - trier
 - grouper,
 - réorganiser,
 - ou effectuer des jointures

Filtrer les données

- L'opérateur **FILTER** permet de filtrer les éléments d'un alias selon une ou plusieurs conditions
- **Syntaxe:**
DEST = FILTER SOURCE BY EXPRESSION;
- **Exemples:**
B = FILTER A BY note > 10;
B = FILTER A BY note > 10 AND
((annee_naissance+age)<2014);

Trier les données

- L'opérateur ORDER permet de trier les éléments selon une condition.
- C'est l'équivalent du ORDER BY de SQL.
- Syntaxe:
DEST = ORDER SOURCE BY FIELD [ASC|DESC];
- Exemples:
B = ORDER A BY note DESC;
B = ORDER A BY note DESC, nom ASC;

Grouper les données

- L'opérateur **GROUP** permet de grouper les tuples selon un champs.
- Il génère des tuples constitués de deux champs:
 - la valeur unique du champs défini par **GROUP**
 - un bag qui contient une série de tuples ayant la valeur unique du champs en commun
- Syntaxe:
DEST = GROUP SOURCE BY FIELD;

Grouper les données (exemple)

Exemple

Dump A;

```
(DUPONT,André,10)  
(DURANT,Marc,14)  
(MARTIN,Christine,10)
```

Exécutons:

B = GROUP A BY note;

Dump B;

```
(10, {(DUPONT,André,10), (MARTIN,Christine,10)})  
(14, {(DURANT,Marc,14)})
```

Réorganiser les données

- L'opérateur **FOREACH ... GENERATE** permet de générer les tuples requis
- Syntaxe:
DEST = FOREACH SOURCE GENERATE EXPRESSION;
- Au sein de l'expression, on peut générer des bags, des tuples, des maps, etc.

Réorganiser les données (exemple)

Exemple

Dump A;

```
(DUPONT,André,10)  
(DURANT,Marc,14)  
(MARTIN,Christine,10)
```

Exécutons:

B = FOREACH A GENERATE note;

Dump B;

```
(10)  
(14)  
(10)
```

Affichage des résultats

- La commande pour afficher les résultats est: **DUMP**
- Elle ne sauvegarde pas les données sur le système de fichier mais se contente de les afficher à l'écran
- Syntaxe: **DUMP <nom du container>;**
- Exemple:
DUMP A;

Sauvegarde des données

- La commande de sauvegarde est: **STORE**
- Syntaxe:
STORE alias INTO 'repertoire' [USING fonction];
 - alias: représente le container à sauvegarder
 - repertoire: répertoire des fichiers
- Les données sont stockées sous formes de fichiers
« part-r-* »

Sauvegarde des données (exemple)

Exemple

Dump A;

```
(DUPONT,André,10)  
(DURANT,Marc,14)  
(MARTIN,Christine,10)
```

Exécutons:

STORE A INTO 'results' USING PigStorage(',');

Un répertoire « results/ » sera créé, contenant ici un fichier unique part-r-00000 avec pour contenu:

```
DUPONT,André,10  
DURANT,Marc,14  
MARTIN,Christine,10
```

Sauvegarde des données (exemple)

Exemple

Dump A;

```
(DUPONT,André,10)
(DURANT,Marc,14)
(MARTIN,Christine,10)
```

Exécutons:

STORE A INTO 'results' USING JsonStorage();

Un répertoire « results/ » sera créé, contenant ici un fichier unique part-r-00000 avec pour contenu:

```
{"nom": "DUPONT", "prenom": "André", "note": 10.0}
{"nom": "DURANT", "prenom": "Marc", "note": 14.0}
{"nom": "MARTIN", "prenom": "Christine", "note": 10.0}
```


Fonctions – chaînes de caractères

- Pig propose, en standard, de nombreuses fonctions pour la manipulation des types chaîne de caractère.
- **TOKENIZE**: découpe les différents mots d'une ligne. Génère un bag contenant un tuple pour chacun des mots.

```
("exemple de phrase")  
("autre exemple")
```

- Exemple, A:
B = FOREACH A GENERATE TOKENIZE(\$0);
DUMP B:

```
({"exemple", "de", "phrase"})  
({"autre", "exemple"})
```

Fonctions – chaînes de caractères

- **SIZE:** renvoie la taille d'une chaîne de caractères
- **SUBSTRING:** permet de récupérer une sous-chaîne de caractères
- **STRSPLIT:** permet de découper une chaîne de caractères à partir d'une expression régulière
- **LOWER:** convertir une chaîne en minuscules
- **REPLACE:** remplacement dans une chaîne (supporte là aussi les regexp)

La jointure

- La commande pour la jointure est: **JOIN**
- **Syntaxe:**

JOIN alias1 BY id1, alias2 BY id2;

- Alias1, alias2: représente les tables.
 - Id1, id2 : represente les champs qui doivent correspondre
-
- **Sélection des enregistrements avec les mêmes clés**

Opérations avec Pig



Hive



Sommaire pour Hive

- Introduction de Hive
- Gestion des tables
- Gestion des données
- Gestion des requêtes (Query)
- Architecture
- Jointures

Cycle de vie des Big Data avec Hadoop

Collecte

Sqoop
Flume

Stockage

HDFS
Hbase

Traitement

MapReduce
Hive, Pig

Analyse

RHadoop
Mahout
Hive, Pig
Spark
Storm
Solr

Exploration

RHadoop
Mahout
Hive, Pig
Spark
Storm
Solr

Qu'est que Hive ?

- **Entrepôt de données pour Hadoop**
- **Langage semblable à SQL, appelé HiveQL**

Utilisation de Hive

- **Requête Ad hoc:**
c.a.d. librement aménagée par l'utilisateur final
- **Analyse de grandes bases de données**
(Analyse des logs, text mining, etc...)

Limitations de Hive

N'est pas conçu pour :

- **le traitement des transactions en ligne**
- **des requêtes temps réel**

HiveQL, SQL pour Hive

HiveQL supporte

- **DDL (Create, Alter, Drop)**
- **DML (Load, Insert, Select)**
- **Fonctions utilisateurs**
- **Appel à des programmes externe MapReduce**

Gestion des tables

Utilisation d'opérations

Hive Data Definition Language (DDL)

Opération	Commande
Création	CREATE TABLE db_name;
Description	DESCRIBE db_name;
Liste	SHOW TABLES;
Modification	ALTER TABLE db_name ADD COLUMNS (field1 STRING);
Suppression	DROP TABLE db_name;

Gestion des données

Utilisation d'opérations

Hive Data Manipulation Language (DML)

Opération	Commande
Chargement des données locales	LOAD DATA LOCAL INPATH './files/data_db_name.txt' OVERWRITE INTO TABLE db_name;
Chargement des données HDFS	LOAD DATA INPATH '/user/myname/data_db_name.txt' OVERWRITE INTO TABLE db_name;

Gestion des requêtes (Query)

Utilisation d'opérations SQL

Opération	Commande
Retrieving information	SELECT from_columns FROM table WHERE conditions;
All values	SELECT * FROM table;
Some values	SELECT * FROM table WHERE rec_name = "value";
Multiple criteria	SELECT * FROM TABLE WHERE rec1 = "value1" AND rec2 = "value2";
Selecting specific columns	SELECT column_name FROM table;
Retrieving unique output records	SELECT DISTINCT column_name FROM table;
Sorting	SELECT col1, col2 FROM table ORDER BY col2;
Counting rows	SELECT COUNT(*) FROM table;
Grouping with counting	SELECT owner, COUNT(*) FROM table GROUP BY owner;

Exemple de gestion de table

- **Créer une table**
- **Charger des données dans une table**
- **Effectuer des requêtes simples**
- **Supprimer la table**

Créer une table


```
hive> CREATE TABLE employes (id INT, prenom STRING, nom STRING)
```

1ere ligne : créer une table de 3 colonnes




```
> ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;
```

2eme ligne : indique que le séparateur
entre les champs est une virgule



La commande doit finir par un
point virgule pour l'exécuter



```
hive> DESCRIBE employes;
```

Affiche la structure de la table

id	int
prenom	string
nom	string

Charger les données dans la table

```
hive> LOAD DATA LOCAL INPATH 'data/data-employees.txt
```

 Copier les valeurs à partir d'un fichier local data-employees.txt

```
> OVERWRITE INTO TABLE employees;
```

 Les enregistrements éventuels de la table employees sont supprimés

```
$ hdfs dfs -cat /user/hive/warehouse/employees/data-employees.txt
```

```
100,jean,martin
```

```
200,robert,poulain
```

```
300,michel,dunot
```

Afficher les enregistrements copiés.

Les tables Hive sont stockés par défaut dans le répertoire
/user/hive/warehouse

Effectuer des requêtes simples

```
hive> SELECT COUNT(*) FROM employes;
```



Compter le nombre d'enregistrements de la table employes

```
hive> SELECT * FROM employes WHERE prenom = "Robert";
```



Afficher les enregistrements satisfaisant un critère

Supprimer une table

```
hive> DROP TABLE employees;
```



Supprimer la table sélectionnée

```
$ hdfs dfs -ls /user/hive/warehouse/  
$
```

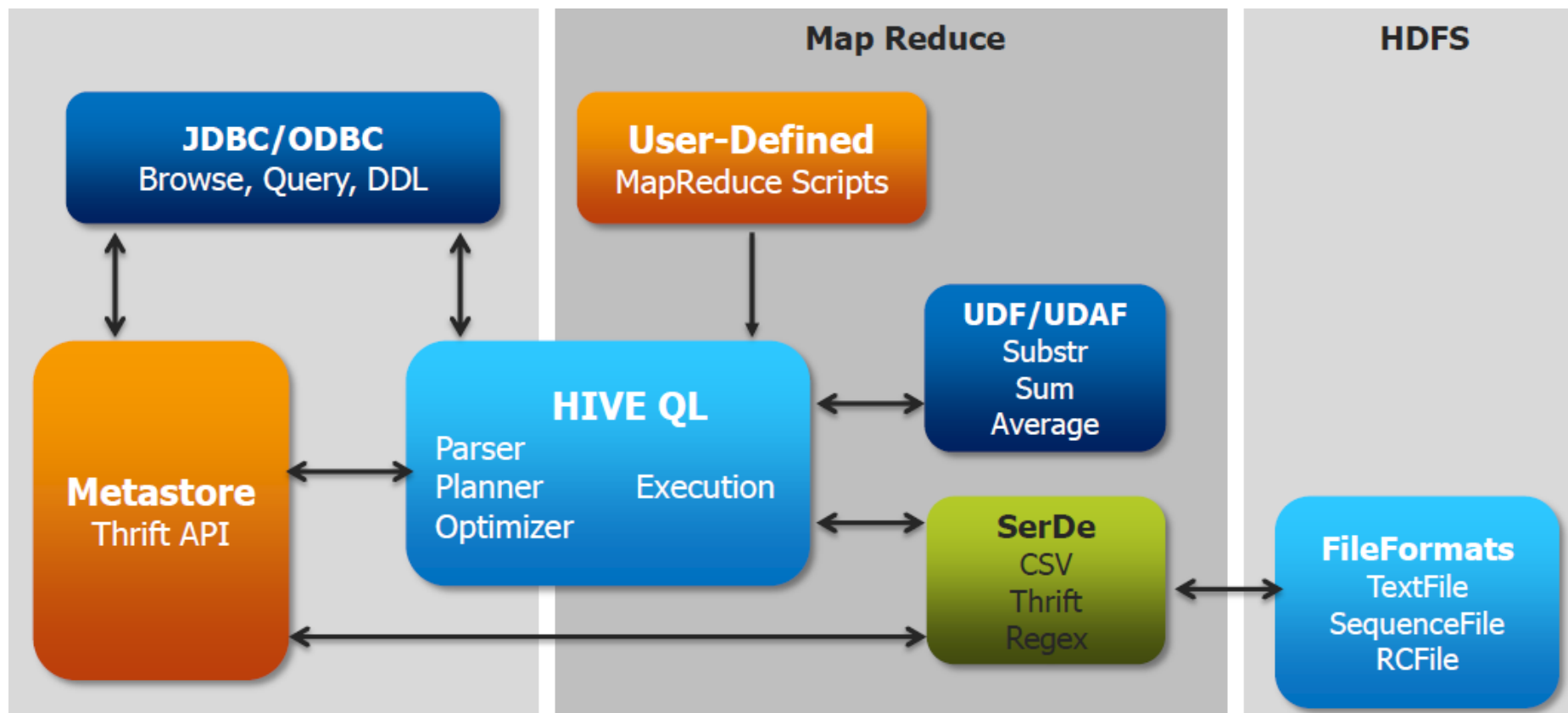


La table a été supprimée

Gestion de table avec Hive



Architecture de Hive



Architecture de Hive – composants (1/3)

- 2 sortes d'interface externe
 - Programmation (API) : JDBC / ODBC
 - Utilisateur : WEB, commande line (CLI)
- Metastore: catalogue qui contient les metadonnées des tables stockées dans Hive
- Thrift API pour exécuter les requetes HiveQL

Architecture de Hive – composants (2/3)

- **User defined: Scripts MapReduce en Java**
- **Compiler (Parser, Planner, Optimizer)**
 - **Transforme les requêtes en un plan qui constitue une suite de DAG de job MapReduce**
- **Compiler (Execution)**
 - **Moteur d'exécution des étapes de DAG**
- **UDF: User Defined Functions**

Architecture de Hive – composants (3/3)

« Ser » pour Sérialiseur

- Utilisé pour écrire des données
- Prend un objet Java utilisé par Hive et le transforme dans un format pour HDFS

« De » pour Désérialiseur

- Utilisé pour des requêtes
- Prend une représentation binaire d'un enregistrement de HDFS et le traduit en un objet Java utilisable par Hive

Techniques performantes avec Hive (1/ 3)

Record Columnar File format (RCFile)

- **Format row-columnar hybride qui permet une analyse efficace lorsque seul un sous-ensemble des données est nécessaire**

Techniques performantes avec Hive (2/3)

Partition: définir ses propres colonnes, son stockage et sa sérialisation

- Les partitions permettent d'accélérer les requêtes
- Les partitions sont physiquement stockées dans des répertoires séparés dans HDFS

```
CREATE EXTERNAL TABLE clicks (  
    hms          STRING,  
    hostname     STRING,  
    process      STRING,  
    pid          INT,  
    uid          INT,  
    message      STRING)  
PARTITIONED BY (  
    year         INT,  
    month        INT,  
    day          INT);
```

Techniques performantes avec Hive (3/3)

Buckets: découpage des partitions

- Optimisation pour les jointures

La jointure

- La commande pour la jointure est: JOIN
- Syntaxe:

```
SELECT alias1.* FROM alias1  
      JOIN alias2 ON (alias1.id = alias2.id)
```

- alias1, alias2: représentent les tables
 - alias1.id, alias2.id : representent les champs qui doivent correspondre
-
- Sélection des enregistrements avec les mêmes clés

Opérations avec Hive



Mahout



Sommaire pour Mahout

- **Besoins d'analyse statistiques**
- **Présentation de Mahout**
- **Introduction des méthodes de clustering**
- **Introduction des méthodes de classification**
- **Introduction des méthodes de Collaborative Filtering**

Cycle de vie des Big Data avec Hadoop

Collecte

**Sqoop
Flume**

Stockage

**HDFS
Hbase**

Traitement

**MapReduce
Hive, Pig**

Analyse

**RHadoop
Mahout
Hive, Pig
Spark
Storm
Solr**

Exploration

**RHadoop
Mahout
Hive, Pig
Spark
Storm
Solr**

Rationaliser les décisions métier

- **En les fondant sur les réponses apportées par l'analyse de données**
- **En prenant en compte des événements pertinents dans le contexte métier**
- **En traitant les données en temps réel**

Apport d'un système de « machine learning »

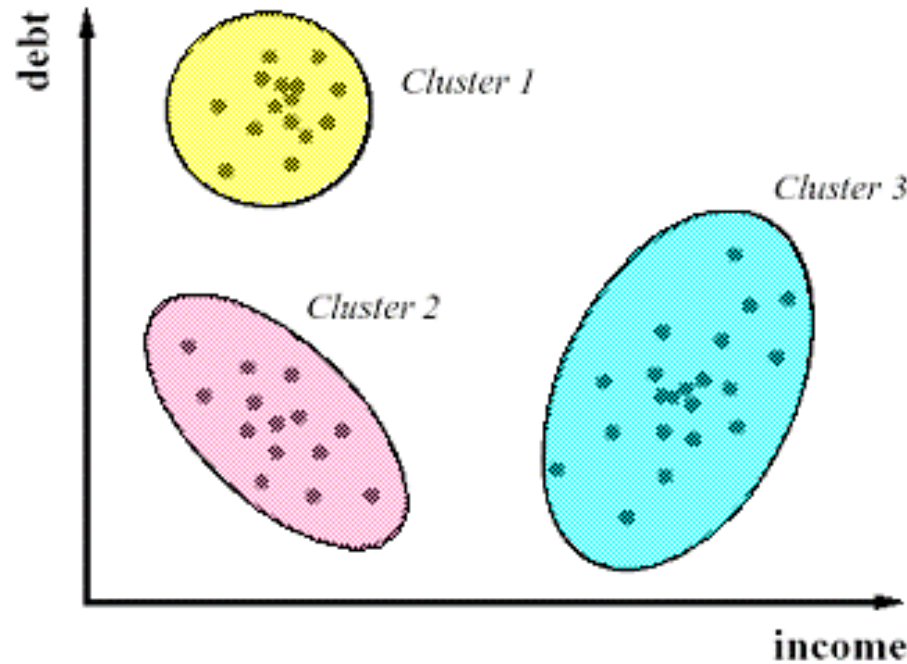
- **Extraire des tendances à partir des données**
 - **Marketing**
 - **Surveillance**
 - **Détection de fraude**
 - **Découverte scientifique**
 - **Découvrir les articles qui sont achetés ensemble**
- **Extraire des modèles de prédiction à partir des données**

Qu'est ce que Mahout ?

Librairie Java pour mettre en œuvre des techniques de:

- **Clustering**
- **Classification**
- **Collaborative Filtering**

Méthodes de Clustering avec Mahout



- Canopy clustering
- K-means clustering
- Fuzzy k-means
- Streaming k-means
- Special clusterin

Partitionnement avec Mahout

- **Sert à diviser un jeu de données en partitions connexes**
- **Utilisation de la méthode des k-moyennes**
 - **Diviser un jeu de données en k partitions en minimisant la distance qui sépare des points et le centre de la partition**

4 étapes du Partitionnement avec Mahout (1/4)

- ❑ Seqdirectory: Convertir un dossier HDFS en fichiers
SequenceFile

4 étapes du Partitionnement avec Mahout (2/4)

- ❑ Seq2sparse: Convertir le contenu des fichiers en vecteurs avec les paramètres suivants:
 - ❑ --input: le répertoire HDFS contenant les fichiers convertis en SequenceFile
 - ❑ --output: le répertoire HDFS contenant le fichier des vecteurs
 - ❑ --namedVector: nom donné aux vecteurs
 - ❑ -ml: seuil logarithmique permettant de retenir les termes les plus importants
 - ❑ -ng: taille d'un N-gramme
 - ❑ -md: nombre minimal de documents dans lequel un terme doit apparaître pour être pris en compte
 - ❑ -s: nombre minimal d'occurrence d'un terme dans un document pour être pris en compte
 - ❑ -wt: algorithme de pondération (ex: tfidf)
 - ❑ -a: type d'analyseur à employer (ex: WhitespaceAnalyzer)

4 étapes du Partitionnement avec Mahout (3/4)

- ❑ Kmeans: lancement de l'algorithme des k-moyennes sur les vecteurs avec les paramètres suivants:
 - ❑ --input: le répertoire HDFS contenant le fichier des vecteurs
 - ❑ --output: le répertoire HDFS de sortie pour le job des k-moyennes
 - ❑ --maxIter: nombre maximal de jobs MapReduce à démarrer
 - ❑ --numClusters: nombre maximal de partitions à identifier
 - ❑ --clusters: les points de partition de la configuration initiale
 - ❑ --clustering: indicateur qui demande à Mahout d'itérer sur les données avant de procéder au partitionnement
 - ❑ --overwrite: indicateur qui permet de remplacer le contenu du répertoire de sortie

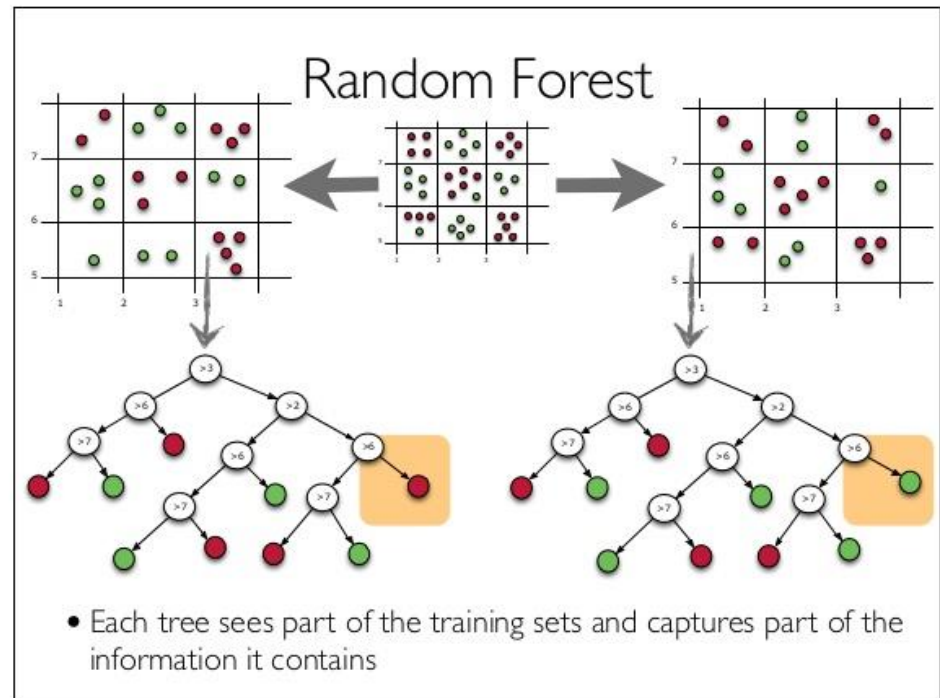
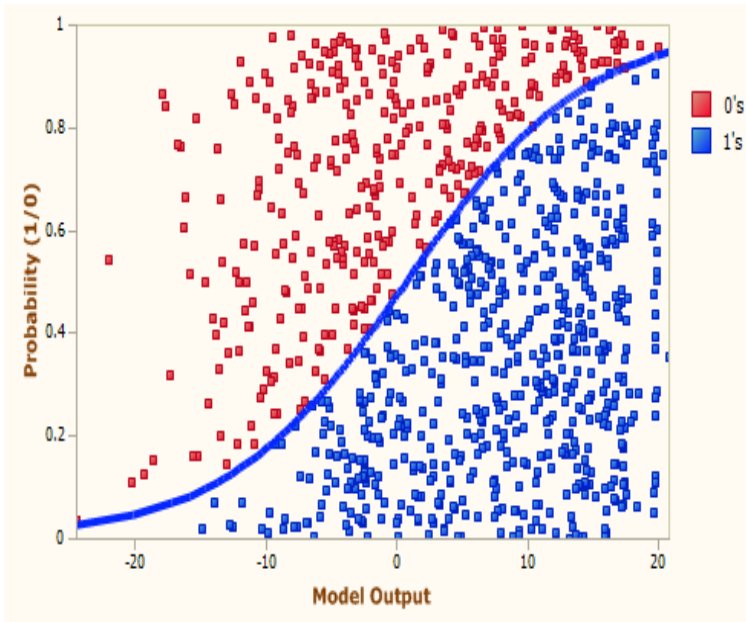
4 étapes du Partitionnement avec Mahout (4/4)

- ❑ Clusterdump: affichage des partitions créées par mahout

Partitionnement



Méthodes de Classification avec Mahout



- **Logistic regression**
- **Naive Bayes**
- **Random Forest**
- **Hidden Markov Models**
- **Multilayer Perceptron**

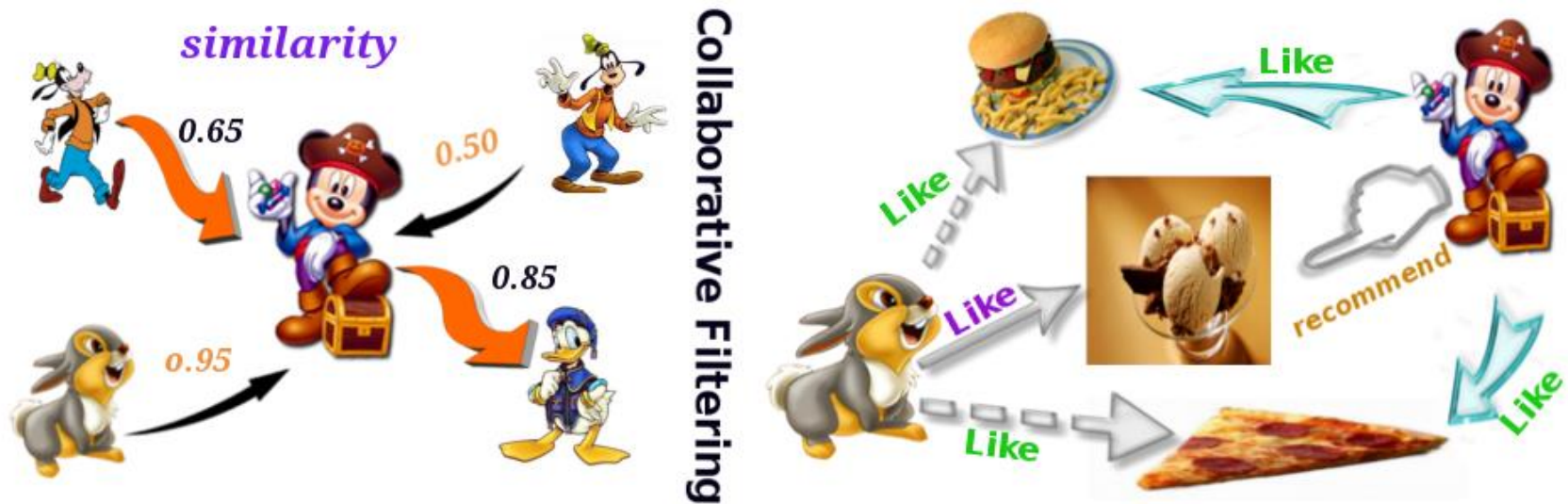
Classification avec Mahout

- **Déterminer la propension d'une personne à aimer ou non certains produits**
- **Utilisation du classificateur bayésien natif d'Apache Mahout**

Etales de Classification avec Mahout

- **Répartition de 80% des données en jeu d'apprentissage et 20% en jeu de test**
- **Apprentissage du classificateur bayésien naif à l'aide du jeu de données d'apprentissage**
- **Test du classificateur à l'aide du jeu de test en affichant un sommaire et une matrice de confusion**

Méthodes de Collaborative Filtering avec Mahout



- User-Based Collaborative Filtering
- Item-Based Collaborative Filtering
- Matrix Factorization with ALS
- Matrix Factorization with ALS on Implicit Feedback
- Weighted Matrix Factorization, SVD++

Recommandation avec Mahout

- Classification des spams
- Découverte des informations les plus partagées
- Bâtir des systèmes de recommandation

Exemple:

Customers Who Bought This Item Also Bought



Oliver Twist (Dover Thrift Editions)
› Charles Dickens
★★★★☆ (213)
Paperback
\$3.50



David Copperfield (Dover Thrift Editions)
› Charles Dickens
★★★★☆ (196)
Paperback
\$5.00

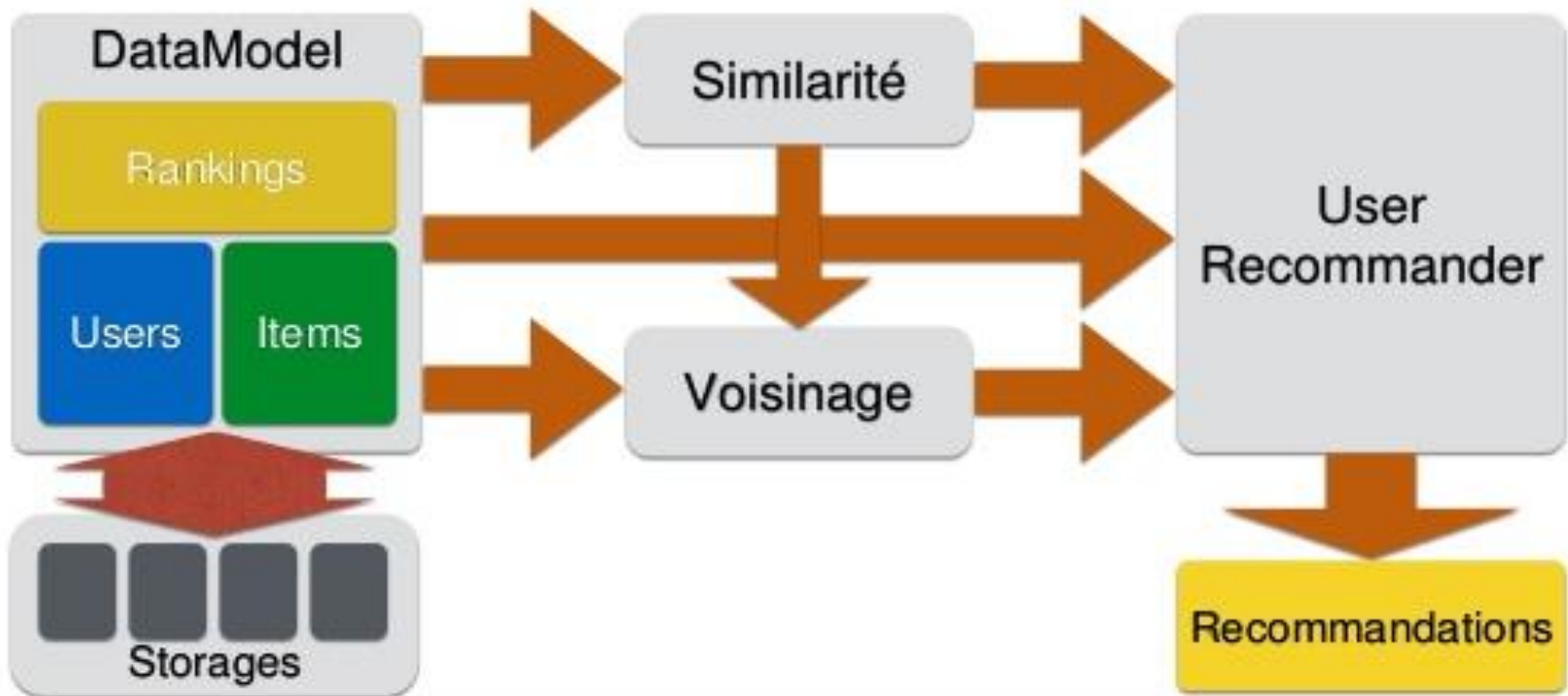


JANE EYRE
› Charlotte Bronte
★★★★☆ (1,045)
Paperback
\$2.99

2 types de recommandation

- **Basé sur les utilisateurs (user-based) :**
 - **2 personnes proches auront probablement des goûts similaires.**
 - **Quels clients ressemblent à ce client et qu'aiment-ils ?**
- **Basé sur les objets (item-based) :**
 - **les recommandations s'appuient sur les similarités entre objets (produits, pages, ...)**
 - **Quels sont les produits similaires à ceux qu'aime ce client ?**

Recommandation basée sur les utilisateurs

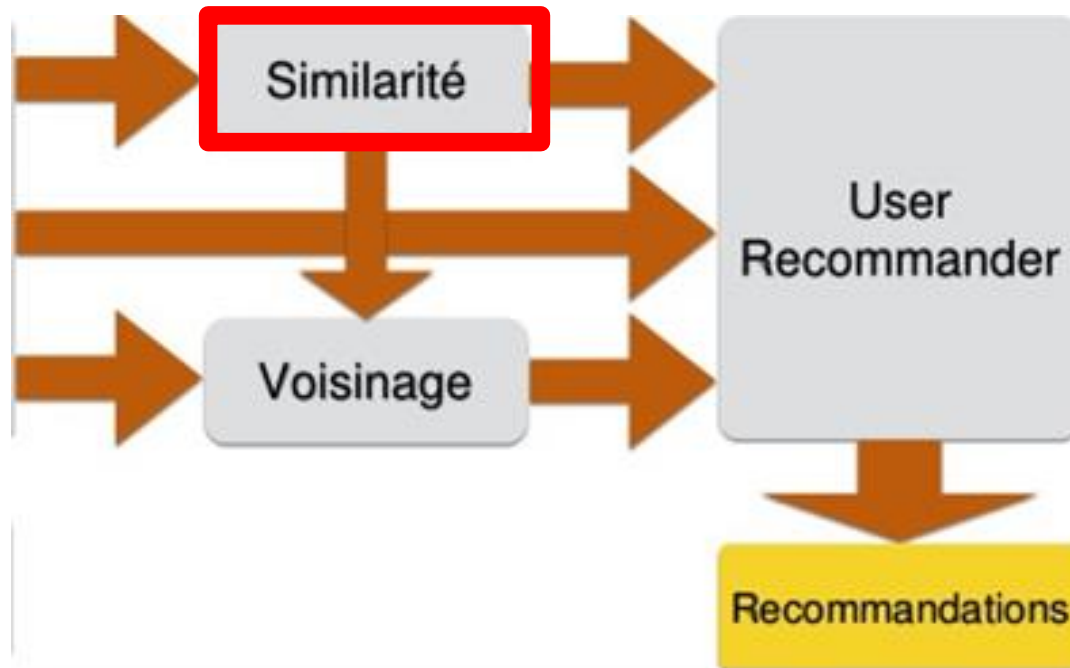


Data Model



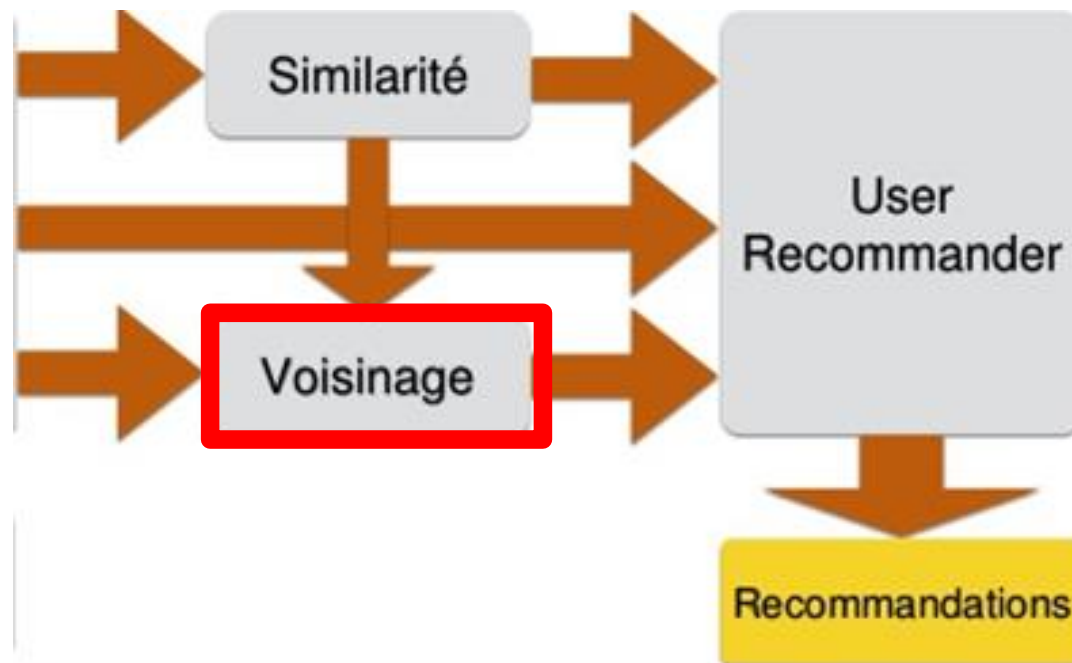
- Les « Users » modélisant les utilisateurs ayant consulté ou acheté via le site
- Les « Items » qui correspondent aux différents produits du catalogue
- Les « Rankings » qui peuvent être des notations, des visualisations de page produit, des achats concrets, ...

Algorithmes de similarité



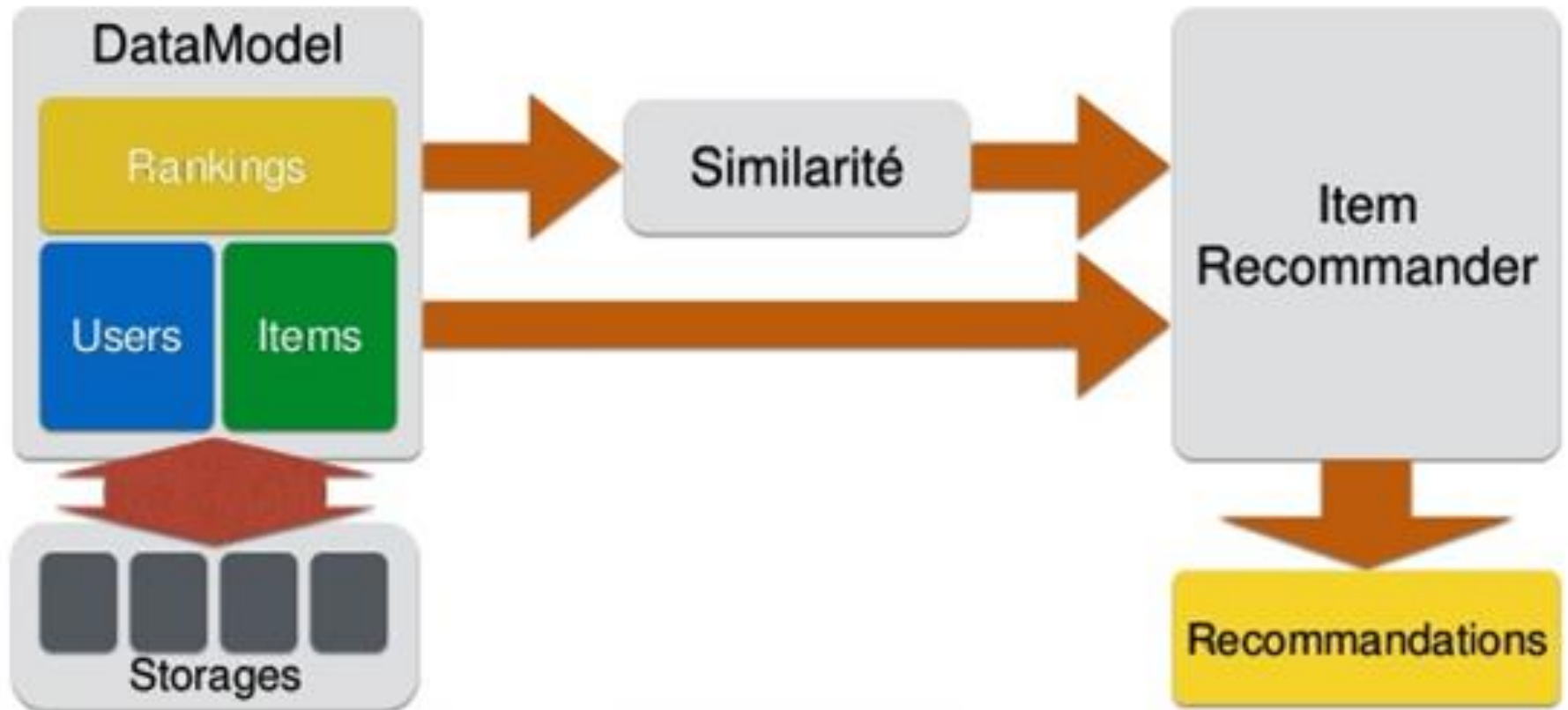
- Les algorithmes de similarité déterminent les utilisateurs les plus proches
- Utilisation de la distance euclidienne, la corrélation de Pearson, la similarité cosinus, ...

Algorithmes de similarité et de voisinage



- Les algorithmes de voisinage déterminent un ensemble d'utilisateurs selon la règle de similarité
- Type Nearest (les X utilisateurs les plus similaires)
- Threshold (tous les utilisateurs dépassant un certain seuil de similarité)

Recommandation basée sur les items



Evaluation

- **Evaluer la qualité d'un système de recommandation en entraînant l'algorithme sur une partie des données et en comparant les résultats obtenus avec le reste des données.**
- **Techniques d'évaluation d'un système "Recommender"**
 - **Prediction-based measures**
 - **IR-based measures**

Ce qu'il faut retenir...

- **Pig est un langage procédural de haut niveau**
- **Hive est un entrepot de données pour Hadoop et est un langage similaire à SQL**
- **Mahout est l'outil d'analyse statistique de Hadoop pour clustering, classification, collaborative filtering**

Merci

