SYSTÈME DE RECOMMANDATION DE FILMS

Résumé. —

Dans ce rapport, nous allons expliqué le déroulement d'un processus Data pour développer un système de recommandation de films en utilisant `ALS` dans l'environnement `Spark`. Ce dernier possède une bibliothèque d'apprentissage automatique intitulé `MLlib` qui dispose d'une implémentation très efficace de `ALS`.

Nous allons détaillé passer par étape par étape dans le développement d'un système de recommandation de films, Nous allons utiliser le jeu de données de MovieLens.

Table des matières

Mots clefs. — ALS,Spark,pySpark,MLlib.

1. Introduction

Les systèmes de recommandation sont apparu avec l'explosion des données qui se résume en filtrage d'information visant des éléments comme les films, les musiques, les pages web .. etc qui vont intéresser l'utilisateur.

Deux systèmes de recommandation existe qui sont :

- Filtrage collaboratif :
  - Similarités entre utilisateurs à partir des articles choisis
  - Le principe est simple proposer du contenu similaires à ceux qu'ils ont déjà choisi, ou alors du contenu choisis par des utilisateurs similaires.
  - Aucune similarité entre les produits n'est considéré
- Basé sur le contenu
  - Comparer les contenus actuels du contenu déjà aimés auparavant

Dans notre exemple nous allons se pencher sur un système de recommandation basé sur le filtrage collaboratif.

2. Collecte de données

Nous allons utiliser le jeu de données MovieLens, qui comporte 100K avis sur 9125 films pour des contraintes matérielles nous n'allons pas sur le grand DataSet qui serait intéressant vu qu'il contient 26 millions d'avis.
Notre dataset se décompose comme ci-dessous :

- links : *movieId,imdbId,tmdbId*
- *Movies : movieId,title,genres*

— *Ratings : userId,movieId,rating,timestamp*
— *Tags : userId,movieId,tag,timestamp*

Nous allons travailler avec rating.csv et movies.csv pour construire notre système de recommandation ensuite les autres pour améliorer notre système.

3. Traitement et exploration des données

Dans cette partie nous allons explorer les données aprés la création d'une session spark nous allons lire les fichiers avec leur header et explorer ces données avec count,schemaPrint,show. Nous remarquons a travers ces informations nous allons convertir rating en float et userId et movieId en integer. Dans les problèmes du monde réel, la matrice d'utilité devrait être très claire, car chaque utilisateur ne rencontre qu'une petite fraction d'éléments parmi le vaste éventail d'options disponibles.
Un problème de démarrage à froid peut survenir lors de l'ajout d'un nouvel utilisateur ou d'un nouvel élément où les deux n'ont pas d'historique en termes de notes.

4. Alternating Least Squares matrix factorization

ALS estime la matrice de notation comme le produit de deux matrices de rang inférieur. En général, ces approximations sont appelées matrices de «facteurs». L'approche générale est itérative. Au cours de chaque itération, l'une des matrices de facteurs est maintenue constante, tandis que l'autre est résolue pour l'utilisation des moindres carrés. La matrice factorielle nouvellement résolue est ensuite maintenue constante lors de la résolution de l'autre matrice factorielle. Les paramètres de ALS sont comme ci-dessous :

— ratings - matrice userID, productID, et rating
— rank - numéro de features
— maxIter - nombre d'itération (10 par défaut)
— lambda - paramètre de régularisation (0.01 par défaut)
— blocks - level of parallelism to split computation into

5. Modélisation algorithmique

Après avoir fini l'étape de traitement et exploration de données nous allons diviser notre dataset en deux partie train et test et créer notre modèle.

```
# ALS model
als = ALS(
          userCol="userId",
          itemCol="movieId",
          ratingCol="rating",
          nonnegative = True,
          implicitPrefs = False,
          coldStartStrategy="drop"
)
model = cv.fit(train)
model.save('als-model')

best_model = model.bestModel
test_predictions = best_model.transform(test)
RMSE = evaluator.evaluate(test_predictions)
print(RMSE)
```

```
0.9070873952326595
```

Nous allons utiliser `CrossValidator` afin de trouver les meilleurs paramètres à notre algorithme de 16 modèles tester nous avons trouver les meilleurs paramètres comme ci dessous :

— Rank 100
— MaxIter 10
— RegParam 0.15

6. Analyse et validation

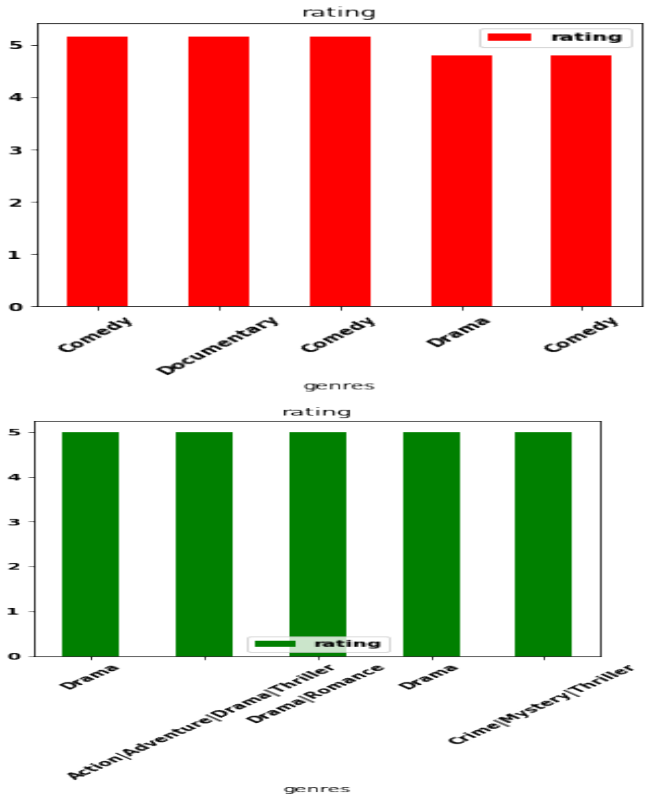
```
# ALS model
als = ALS(
         userCol="userId",
         itemCol="movieId",
         ratingCol="rating",
         nonnegative = True,
         implicitPrefs = False,
         coldStartStrategy="drop"
)

param_grid = ParamGridBuilder() \
            .addGrid(als.rank, [10, 25, 50, 100]) \
            .addGrid(als.regParam, [.01, .05, .1, .15]) \
            .build()

evaluator = RegressionEvaluator(
           metricName="rmse",
           labelCol="rating",
           predictionCol="prediction")

print ("Num models to be tested: ", len(param_grid))
cv = CrossValidator(estimator=als, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=5)

model = cv.fit(train)
best_model = model.bestModel

print("**Best Model**")
print("  Rank:", best_model._java_obj.parent().getRank())
print("  MaxIter:", best_model._java_obj.parent().getMaxIter())
print("  RegParam:", best_model._java_obj.parent().getRegParam())
```

```
Num models to be tested:  16
**Best Model**
  Rank: 50
  MaxIter: 10
  RegParam: 0.15
```

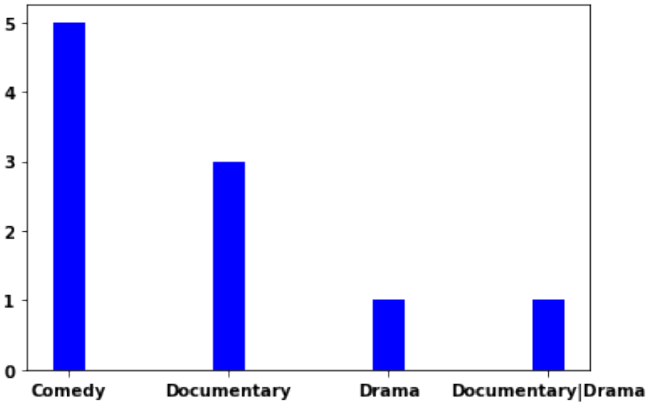
Nous allons effectuer des recommandation pour tous les utilisateurs et nous avons remarquer qu'on arrive à proposer des recommandations très proches du genres par rapport à l'historique utilisateur.

7. Visualisation et reporting

Recommandation à 3 utilisateurs grâce à leur historiques :



En rouge notre recommandation et en vert les films déjà vu et noté, maintenant nous allons regarder nos meilleurs genre dans les recommandation les plus recommandés :



Voici la liste des meilleurs 10 films en recommandation :

— 'Cops (1922)'
— 'Land of Silence and Darkness (Land des Schweigens und der Dunkelheit) (1971)'
— 'Goat, The (1921)'
— 'Ben X (2007)'
— 'My Best Friend (Mon meilleur ami) (2006)'
— 'Gates of Heaven (1978)'
— 'Cameraman, The (1928)'
— 'One Piece Film : Strong World (2009)'
— 'Navigator, The (1924)'
— 'Dylan Moran : Monster (2004)'

Documentation et sources

[1] learn-co-students
LEARN-CO-STUDENTS
https://github.com/learn-co-students/dsc-4-39-06-building-recommendation-system-als-pyspark-demo-online-ds-000

[2] Youtube Video
EDUREKA
https://www.youtube.com/watch?v=GIXAsrco2Rc&list=LL&index=2&ab_channel=edureka

[3] Youtube Video
JAMENLONG1
https://www.youtube.com/watch?v=FgGjc5oabrA&list=LL&index=1&t=100s&ab_channel=jamenlong1

[4] Spark Documentation
DOCUMENTATION
https://spark.apache.org/docs/2.2.0/ml-collaborative-filtering.html

[5] Code Flow Site
UNKOWN
https://www.codeflow.site/fr/article/build-recommendation-engine-collaborative-filtering

[6] Collaborative Filtering with ALS
SNEHAL NAIR
https://towardsdatascience.com/build-recommendation-system-with-pyspark-using-alternating-least-squares-als-matrix-factorisat