


서울대학교
융합과학기술대학원
Seoul National University
Graduate School of Convergence
Science and Technology

Recommendation Systems 실습

Test 1

- Code
  - https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html
- Data
  - https://github.com/apache/spark/blob/master/data/mllib/als/test.data

```python
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating

# Load and parse the data
data = sc.textFile("data/mllib/als/test.data")
ratings = data.map(lambda l: l.split(','))\
    .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))

# Build the recommendation model using Alternating Least Squares
rank = 10
numIterations = 10
model = ALS.train(ratings, rank, numIterations)

# Evaluate the model on training data
testdata = ratings.map(lambda p: (p[0], p[1]))
predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]), r[2]))
ratesAndPreds = ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
print("Mean Squared Error = " + str(MSE))

# Save and load model
model.save(sc, "target/tmp/myCollaborativeFilter")
sameModel = MatrixFactorizationModel.load(sc, "target/tmp/myCollaborativeFilter")
```

Loading the data

```
from pyspark.mllib.recommendation import ALS,
MatrixFactorizationModel, Rating

# Load and parse the data
data = sc.textFile("data/mllib/als/test.data")
ratings = data.map(lambda l: l.split(','))\
    .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))
```

Build the model

```
# Build the model using Alternating Least Squares

rank = 10
numIterations = 10

model = ALS.train(ratings, rank, numIterations)

# Build the model using ALS based on implicit ratings
# model = ALS.trainImplicit(ratings, rank, numIterations,
alpha=0.01)

# Alpha는 regularization을 위한 파라메터
# 크면 regularization이 잘됨 그러나 factorization이 잘 안될 수도
```

Read the programming manual

https://spark.apache.org/docs/2.3.1/api/python/pyspark.mllib.html#module-pyspark.mllib.recommendation

class `pyspark.mllib.recommendation.`**`ALS`** [source]

Alternating Least Squares matrix factorization

New in version 0.9.0.

classmethod **`train`**(*ratings*, *rank*, *iterations=5*, *lambda_=0.01*, *blocks=-1*, *nonnegative=False*, *seed=None*) [source]

Train a matrix factorization model given an RDD of ratings by users for a subset of products. The ratings matrix is approximated as the product of two lower-rank matrices of a given rank (number of features). To solve for these features, ALS is run iteratively with a configurable level of parallelism.

Parameters:

- **ratings** – RDD of *Rating* or (userID, productID, rating) tuple.
- **rank** – Number of features to use (also referred to as the number of latent factors).
- **iterations** – Number of iterations of ALS. (default: 5)
- **lambda** – Regularization parameter. (default: 0.01)
- **blocks** – Number of blocks used to parallelize the computation. A value of -1 will use an auto-configured number of blocks. (default: -1)
- **nonnegative** – A value of True will solve least-squares with nonnegativity constraints. (default: False)
- **seed** – Random seed for initial matrix factorization model. A value of None will use system time as the seed. (default: None)

New in version 0.9.0.

Evaluating the model

```
# 기존 자료에서 세번째 column 데이터를 삭제
# rating은 사용자 아이템 선호점수 순으로 들어 있었음
testdata = ratings.map(lambda p: (p[0], p[1]))

# 학습된 모델을 바탕으로 추정치를 모든 테스트데이터에 대하여 구한다
# 이를 map함수를 이용하여 추정치의 처음 세가지의 column을 추출
# 세가지 말고 다른 정보도 주는가?
# https://spark.apache.org/docs/2.3.1/api/python/pyspark.mllib.html#module-
pyspark.mllib.recommendation
# 잘 모르면?
# model.predictAll(testset).take(10)
# [Rating(user=1, product=1, rating=1.0...), Rating(user=1, product=2,
rating=1.9...)]
predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]), r[2]))
```

```
predictions.take(5)
```

```
[((4, 4), 4.996116497184431),
 ((4, 1), 1.0012320185093573),
 ((4, 2), 4.996116497184431),
 ((4, 3), 1.0012320185093573),
 ((1, 4), 1.0012122958229726)]
```

```
ratesAndPreds = ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
```

```
ratesAndPreds.take(5)
```

```
[((1, 3), (5.0, 4.996216033749226)),
 ((1, 4), (1.0, 1.0012122958229726)),
 ((3, 2), (5.0, 4.996116497184431)),
 ((2, 2), (1.0, 1.0012122958229726)),
 ((2, 4), (1.0, 1.0012122958229726))]
```

```
# 실제값과 추정값을 조인
# key를 (사용자, 프로덕트)로 만든 것에 주의
ratesAndPreds = ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)

# (실제값 - 측정값)^2
MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
print("Mean Squared Error = " + str(MSE))

0.0000xxxx…
```

Test 2

- Code
  - https://spark.apache.org/docs/latest/ml-collaborative-filtering.html

- Data
  - https://github.com/apache/spark/blob/master/data/mllib/als/sample_movielens_ratings.txt

```python
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row

lines = spark.read.text("data/mllib/als/sample_movielens_ratings.txt").rdd
parts = lines.map(lambda row: row.value.split("::"))
ratingsRDD = parts.map(lambda p: Row(userId=int(p[0]), movieId=int(p[1]),
                                     rating=float(p[2]), timestamp=long(p[3])))
ratings = spark.createDataFrame(ratingsRDD)
(training, test) = ratings.randomSplit([0.8, 0.2])

# Build the recommendation model using ALS on the training data
# Note we set cold start strategy to 'drop' to ensure we don't get NaN evaluation metrics
als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating",
          coldStartStrategy="drop")
model = als.fit(training)

# Evaluate the model by computing the RMSE on the test data
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))

# Generate top 10 movie recommendations for each user
userRecs = model.recommendForAllUsers(10)
# Generate top 10 user recommendations for each movie
movieRecs = model.recommendForAllItems(10)

# Generate top 10 movie recommendations for a specified set of users
users = ratings.select(als.getUserCol()).distinct().limit(3)
userSubsetRecs = model.recommendForUserSubset(users, 10)
# Generate top 10 user recommendations for a specified set of movies
movies = ratings.select(als.getItemCol()).distinct().limit(3)
movieSubSetRecs = model.recommendForItemSubset(movies, 10)
```

Reading data

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row

lines = spark.read.text("data/mllib/als/sample_movielens_ratings.txt").rdd

parts = lines.map(lambda row: row.value.split("::"))
#ratingsRDD = parts.map(lambda p: Row(userId=int(p[0]), movieId=int(p[1]),
#                                     rating=float(p[2]), timestamp=long(p[3])))
# Python3에서는 long대신에 int
ratingsRDD = parts.map(lambda p: Row(userId=int(p[0]), movieId=int(p[1]),
                                     rating=float(p[2]), timestamp=int(p[3])))

ratings = spark.createDataFrame(ratingsRDD)
(training, test) = ratings.randomSplit([0.8, 0.2])
```

Training

```
# Build the recommendation model using ALS on the training data
# Note we set cold start strategy to 'drop' to ensure we don't
get NaN evaluation metrics
# 각 parameter가 무엇을 하는지 매우 잘 알아야 합니다.
als = ALS(maxIter=5, regParam=0.01, userCol="userId",
                itemCol="movieId", ratingCol="rating",
                coldStartStrategy="drop")

model = als.fit(training)
```

평가

```
# Evaluate the model by computing the RMSE on the test data

# Transform과 prediction의 차이는? RDD vs. Data Frame
predictions = model.transform(test)
# predictions.take(5)

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

rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))
```

RegressionEvaluator

RegressionEvaluator — Evaluator of Regression Models

`RegressionEvaluator` is an Evaluator of regression models (e.g. ALS, `DecisionTreeRegressor`, DecisionTreeClassifier, `GBTRegressor`, `GBTClassifier`, RandomForestRegressor, RandomForestClassifier, LinearRegression, `RFormula`, `NaiveBayes`, LogisticRegression, `MultilayerPerceptronClassifier`, `LinearSVC`, GeneralizedLinearRegression).

Table 1. RegressionEvaluator's Metrics and isLargerBetter Flag

Metric	Description	isLargerBetter
`rmse`	Root mean squared error	`false`
`mse`	Mean squared error	`false`
`r2`	• (default) Unadjusted coefficient of determination • Regression through the origin	`true`
`mae`	Mean absolute error	`false`

결과의 사용

```
# Generate top 10 movie recommendations for each user
userRecs = model.recommendForAllUsers(10)
# Generate top 10 user recommendations for each movie
movieRecs = model.recommendForAllItems(10)
movieRecs.take(1)
```

```
# Generate top 10 movie recommendations for each user
userRecs = model.recommendForAllUsers(10)
# Generate top 10 user recommendations for each movie
movieRecs = model.recommendForAllItems(10)
```

```
userRecs.take(1)
```

```
[Row(userId=28, recommendations=[Row(movieId=22, rating=5.917771339416504), Row(movieId=92, r
ating=5.2026591300964355), Row(movieId=81, rating=4.823528289794922), Row(movieId=80, rating=
4.709908962249756), Row(movieId=12, rating=4.491816997528076), Row(movieId=64, rating=4.28580
1887512207), Row(movieId=2, rating=4.262893199920654), Row(movieId=74, rating=4.2363982200622
56), Row(movieId=89, rating=4.162567615509033), Row(movieId=51, rating=3.908174991607666)])]
```

```
movieRecs.take(1)
```

```
[Row(movieId=31, recommendations=[Row(userId=7, rating=2.9601876735687256), Row(userId=14, ra
ting=2.722259044647217), Row(userId=8, rating=2.4851861000061035), Row(userId=6, rating=2.386
873960494995), Row(userId=23, rating=2.2613019943237305), Row(userId=21, rating=2.03930854797
3633), Row(userId=16, rating=1.7136212587356567), Row(userId=17, rating=1.6259549856185913),
Row(userId=12, rating=1.568527102470398), Row(userId=13, rating=1.542325496673584)])]
```

```
# Generate top 10 movie recommendations for a specified set of users
users = ratings.select(als.getUserCol()).distinct().limit(3)
userSubsetRecs = model.recommendForUserSubset(users, 10)
# Generate top 10 user recommendations for a specified set of movies
movies = ratings.select(als.getItemCol()).distinct().limit(3)
movieSubSetRecs = model.recommendForItemSubset(movies, 10)
```

```
users = ratings.select(als.getUserCol()).distinct().limit(3)
users.collect()
```

```
[Row(userId=26), Row(userId=29), Row(userId=19)]
```

```
userSubsetRecs = model.recommendForUserSubset(users, 10)
```

```
userSubsetRecs.take(1)
```

```
[Row(userId=26, recommendations=[Row(movieId=64, rating=5.988473415374756), Row(movieId=17, r
ating=5.609796047210693), Row(movieId=68, rating=5.400238990783691), Row(movieId=94, rating=
5.183830261230469), Row(movieId=90, rating=5.170174598693848), Row(movieId=23, rating=5.02645
9217071533), Row(movieId=88, rating=4.999044418334961), Row(movieId=24, rating=4.951570987701
416), Row(movieId=22, rating=4.78335428237915), Row(movieId=46, rating=4.768810272216797)])]
```

Mllib에 무슨 알고리즘이 포함되어 있는가?

https://spark.apache.org/docs/2.3.1/api/python/pyspark.mllib.html

- Classification
- Clustering
- Association Mining
- Recommendation
- Regression
- Statistics
- Decision Tree

- Utility
  - Feature: Vector, Word, Scaler
  - Evaluation: Metrics
  - Linear algebra