스파크 MLib

순천향대학교 컴퓨터공학과

이 상 정

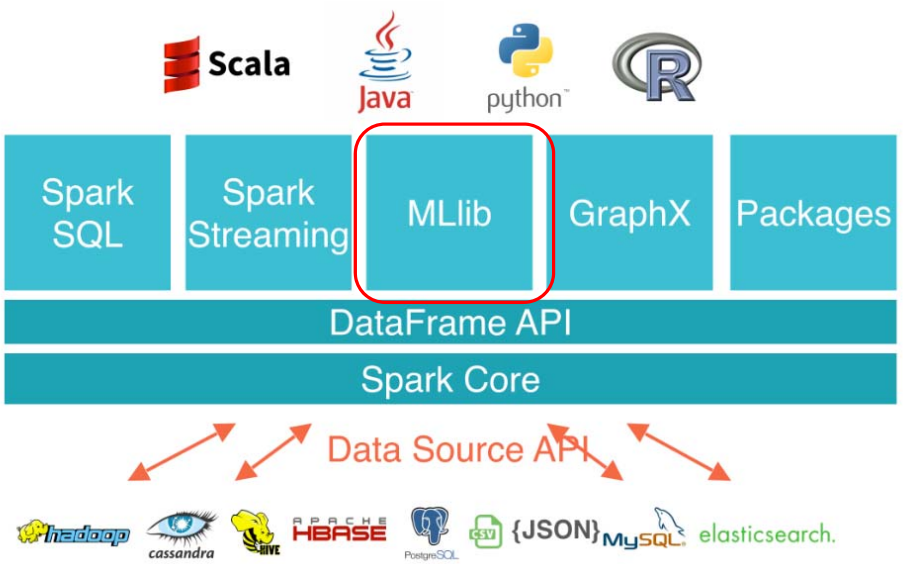
학습 내용

1. MLib 개요
2. 머신 러닝 소개
   - 분류 (Classification)
   - 클러스터링 (Clustering)
   - 협업 필터링 (Collaborative Filtering)
3. 사용자 선택 예측을 위한 협업 필터링

1. MLib 개요

MLiB 소개

- MLib는 스파크 머신 러닝(Machine Learning) 라이브러리
  - 클러스터 상에서 병렬로 실행되는 머신 러닝 알고리즘



MLib 알고리즘과 유틸리티

Algorithms and Utilities	Description
Basic statistics	Includes summary statistics, correlations, hypothesis testing, random data generation
Classification and regression	Includes methods for linear models, decision trees and Naïve Bayes
Collaborative filtering	Supports model-based collaborative filtering using alternating least squares (ALS) algorithm
Clustering	Supports K-means clustering
Dimensionality reduction	Supports dimensionality reduction on the RowMatrix class; singular value decomposition (SVD) and principal component analysis (PCA)
Feature extraction and transformation	Contains several classes for common feature transformations

MLib 알고리즘과 유틸리티 – 기본 통계

Basic statistics	Includes summary statistics, correlations, hypothesis testing, random data generation

- MLib는 기본 통계를 수행하는 기능 제공
  - 요약 통계값 (Summary Statistics)
    - 평균값, 최대/최소값, 분산(variance) 등
  - 상관 분석 (Correlation)
    - 두 데이터 집합들 간의 연관된 정도를 분석
      - 피어스 상관분석 (Pearson's correlation)
      - 스피어만 상관분석 (Spearmann's correlation)
  - 가설 검정 (Hypothesis Testing)
    - 통계적 가설의 유의성(significance) 여부 판단
      - 피어슨 카이제곱 검정 (Pearson's Chi-squared test)
  - 난수 데이터 생성 (Random Data Generation)

MLib 알고리즘과 유틸리티 – 분류 및 회귀분석

Classification and regression	Includes methods for linear models, decision trees and Naïve Bayes

- MLib는 다양한 분류 및 회귀분석 알고리즘 제공
  - 분류 (Classification)
    - 입력 데이터의 카테고리를 결정
      - 로지스틱 회귀 (Logistic Regression)
      - 결정 트리 분류기 (Decision Tree Classifier)
      - 랜덤 포레스트 분류기 (Random Forest Classifier)
      - 선형 SVM (Linear Support Vector Machine)
      - 나이브 베이즈 분류 (Naive Bayes Classification)
  - 회귀 분석 (Regression Analysis)
    - 연속형 데이터들의 모형을 계산하고 적합도를 측정하는 분석으로 통계적 예측에 적용
  - 결정 트리 (Decision Tree)
    - 어떤 항목에 대한 관측값과 목표값을 연결시켜주는 예측 모델

MLib 알고리즘과 유틸리티 – 협업 필터링

Collaborative filtering	Supports model-based collaborative filtering using alternating least squares (ALS) algorithm

- MLib는 추천 시스템(recommender system)에 주로 사용되는 협업 필터링(Collaborative Filtering) 제공
  - 많은 사용자들로부터 얻은 과거의 정보(경향)로 부터 사용자의 미래의 관심사를 예측
    - ALS (Alternative Least Squares) 알고리즘

MLib 알고리즘과 유틸리티 – 클러스터링

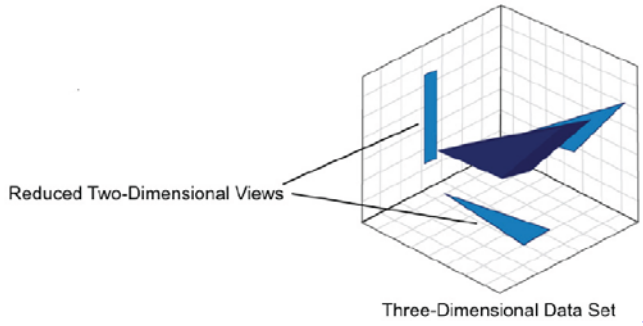
Clustering	Supports K-means clustering

- MLib 는 클러스터링 (Clustering) 알고리즘을 제공
  - 유사한 데이터들을 같은 그룹으로 군집화
    - K-평균 알고리즘 (K-means algorithm)
    - 잠재 디리클레 할당 (Latent Dirichlet Allocation, LDA)

MLib 알고리즘과 유틸리티 – 차원 축소

Dimensionality reduction	Supports dimensionality reduction on the RowMatrix class; singular value decomposition (SVD) and principal component analysis (PCA)

- MLib 는 차원 축소 (Dimension Reduction) 알고리즘을 제공
  - 고차원의 데이터를 저차원의 데이터(변수의 수를 줄임)로 변환하는 알고리즘으로 원본 데이터에서 특징(feature)을 추출하는데 적용
    - 주성분 분석 (Principal Component Analysis, PCA)
    - 특이값 분해 (Singular Value Decomposition)



MLib 알고리즘과 유틸리티 – 특징 추출 및 변환

Feature extraction and transformation	Contains several classes for common feature transformations

- MLib는 원본 데이터로부터 특징(Feature)을 추출(extraction), 변환(transformation), 선택(selsction)하는 기능 제공
  - 기계 학습에서 원본 데이터로부터 학습 및 분석에 유용한 유도된 값들인 특징을 추출
    - TF-IDF (Term Frequency-Inverse Document Frequency)
    - Word2Vec
    - Tokenizer
    - VectorSlicer

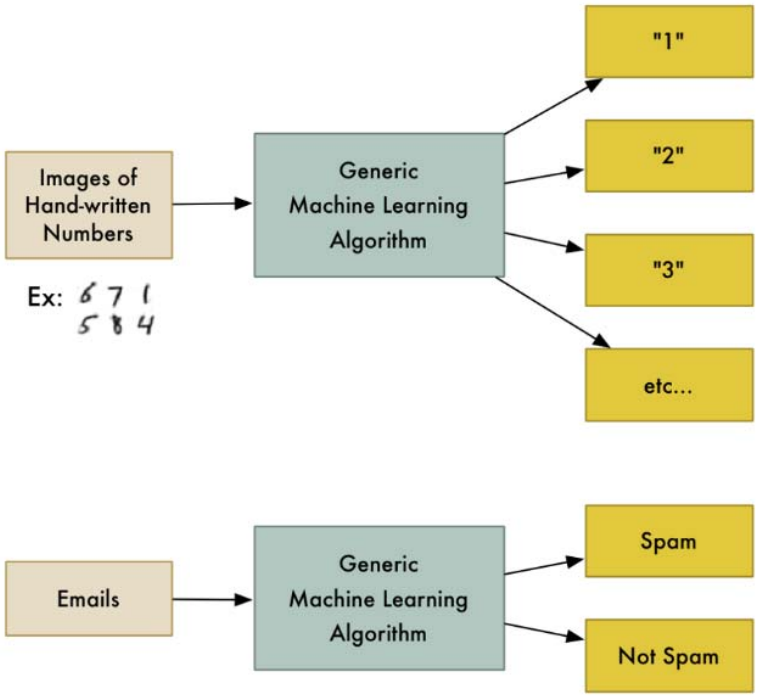
2. 머신 러닝 소개

머신 러닝이란?

- 머신 러닝은 문제를 해결하기 위한 맞춤 코드(custom code)가 아닌 일련의 데이터에 대해 무언가 흥미로운 것을 알려줄 수 있는 일반 알고리즘(generic algorithms)
  - 코드를 작성하는 대신 데이터를 일반 알고리즘에 공급하면, 데이터를 기반으로 한 자체 로직이 생성

- 분류 알고리즘(classification algorithm) 예
  - 데이터를 서로 다른 그룹으로 분류
  - 필기체 숫자를 인식에 사용되는 동일한 분류 알고리즘을 그대로 이메일의 스팸 분류에 적용
  - 동일한 알고리즘이지만 다른 학습 데이터를 제공하면 다른 분류 로직을 자동으로 생성
  - 머신 러닝은 이런 종류의 일반 알고리즘(generic algorithms)을 의미하는 포괄적인 용어

분류 알고리즘 예

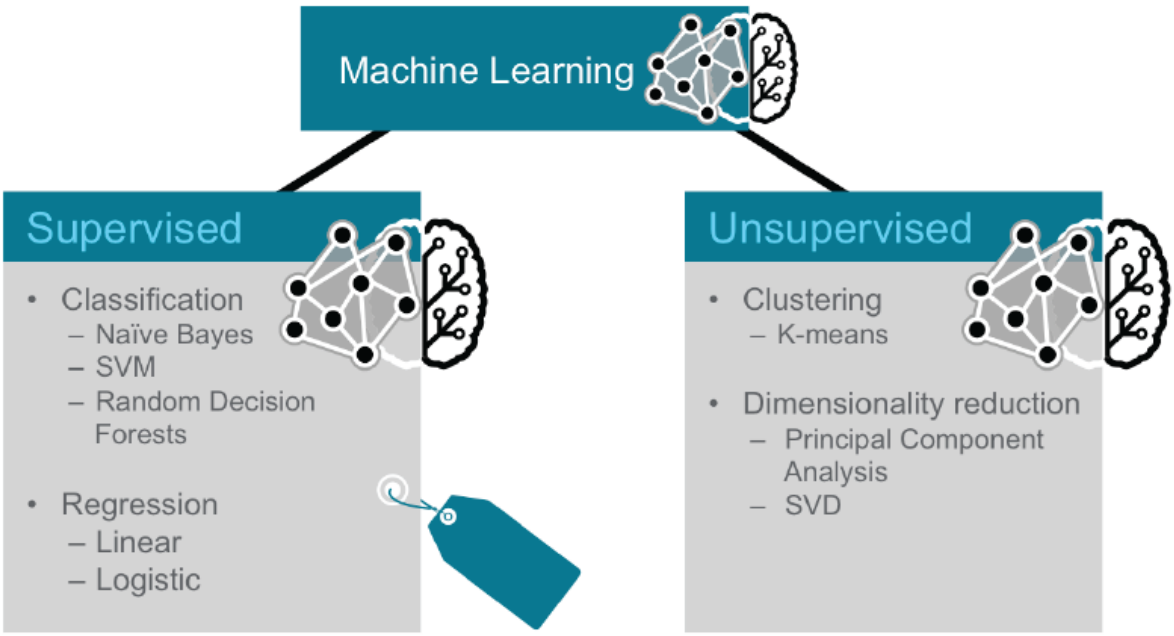


지도 학습과 비지도 학습 (1) (Supervised vs. Unsupervised Learning)

- 기계 학습 알고리즘은 지도 학습 (supervised learning)과 비지도 학습 (unsupervised learning)으로 구분

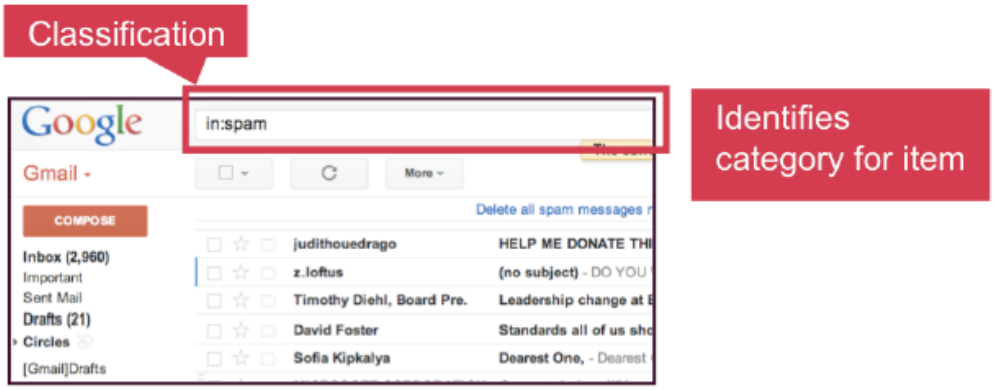
- 지도 학습 (supervised learning)
  - 결과에 대한 사전 지식이 필요
  - 훈련 데이터에 라벨링이 되어 있어야 함.
    - 즉, 각 질문(input)에 대해 무엇이 정답(output)인지 훈련 데이터가 알고 있어야 함
  - 회귀분석(regression), 분류(classification), 협업 필터링 (collaborative filtering), 딥러닝(deep learning) 등
- 비지도 학습 (unsupervised learning)
  - 구체적 결과에 대한 사전 지식이 없지만 데이터를 통해 유의미한 지식을 얻고자 하는 경우에 사용
  - 데이터만 존재, 즉, 문제는 있는데 답은 없는 경우
  - 클러스터링(clustering), 차원 축소(dimension reduction) 등

지도 학습과 비지도 학습 (2)



분류 알고리즘 (1)

- 구글의 Gmail은 이메일의 스팸 여부를 분류(classification) 알고리즘을 사용하여 감지
  - 이메일의 데이터(송신자, 수신자, 제목, 메시지)에 근거하여 분류
  - 훈련된 데이터(라벨된 데이터)가 알고리즘에 제공되는 지도 학습



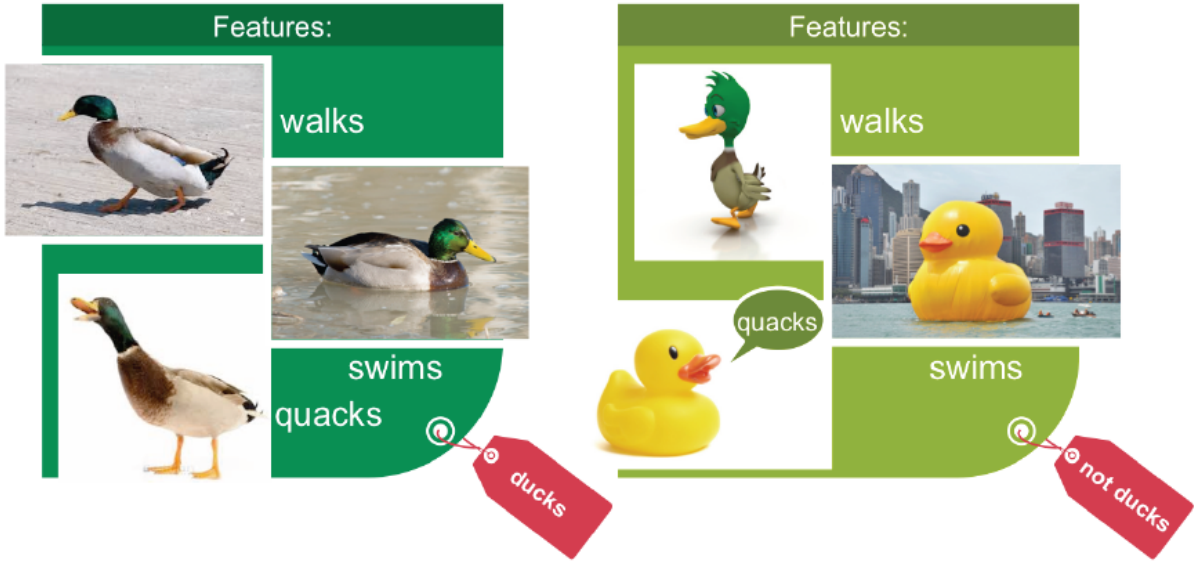
분류 알고리즘 (2)

- 훈련 데이터로 학습이 끝나면, 새로이 입력된 데이터에 대해 미리 정의된 카테고리 중 하나로 분류

- 분류 알고리즘 적용 예
  - 스팸 메일 감지, 카드 사기 감지, 감성 분석(sentiment analysis)

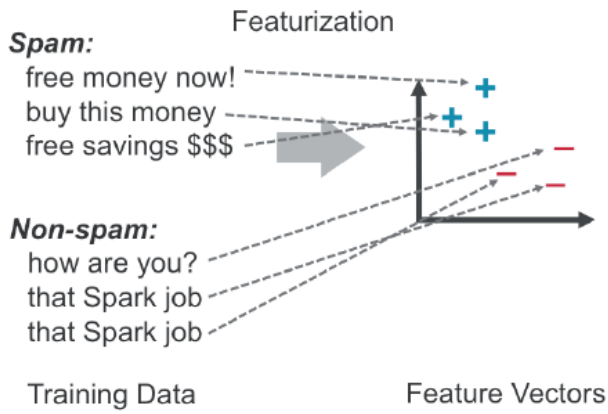
분류 알고리즘 (3)

- 미리 정의된 특징(feature)에 기반하여 분류



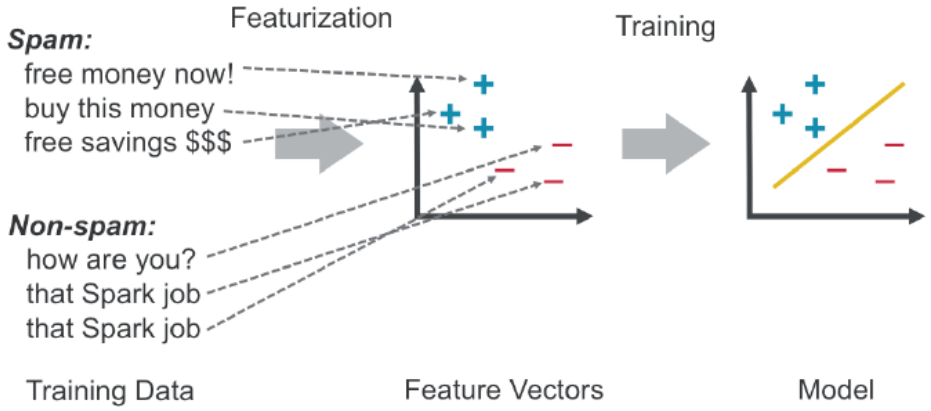
분류기 모델 구축 – 특징 추출

- 분류기 모델 (classifier model) 구축 첫 단계로는 특징을 추출하고, 특징 벡터(Feature Vector)로 변환
  - 특징 벡터는 각 특징들을 나타내는 수치들의 벡터



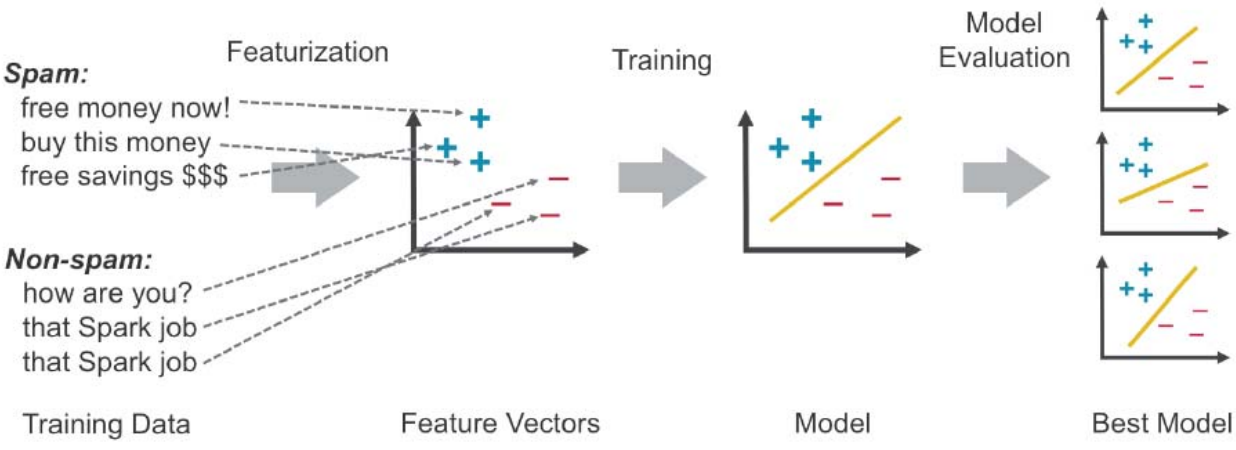
분류기 모델 구축 – 모델 훈련

- 훈련 데이터의 입력 특징들과 라벨된 출력 값들 사이의 연관성으로 분류기 모델을 훈련



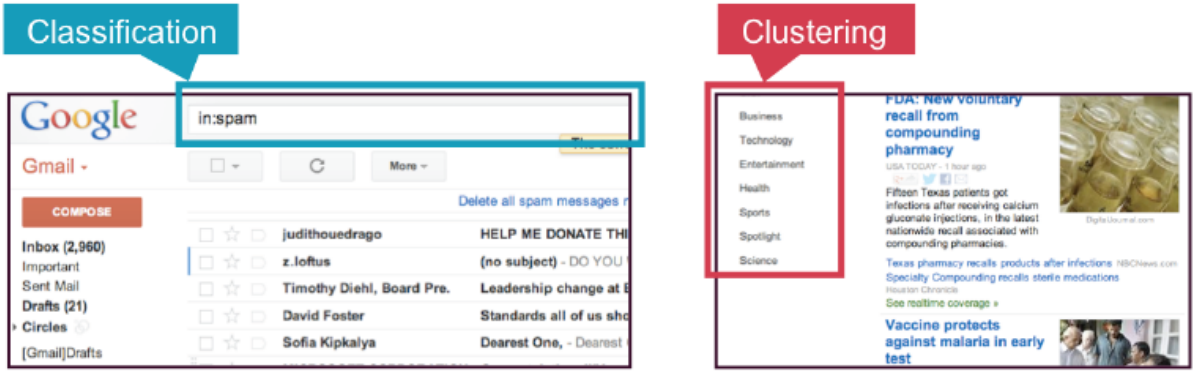
분류기 모델 구축 – 분류

- 새로운 입력 데이터에 대해 특징들을 추출하고 모델에 적용 평가하여 미리 정의된 카테고리 중 하나로 분류



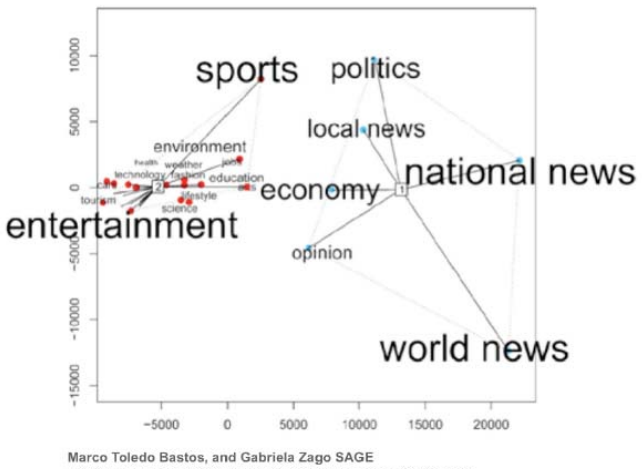
클러스터링 알고리즘 (1)

- 구글 News는 제목과 내용에 기반하여 뉴스 기사들을 카테고리로 그룹화하는데 클러스터링 (Clustering) 알고리즘 을 사용



클러스터링 알고리즘 (2)

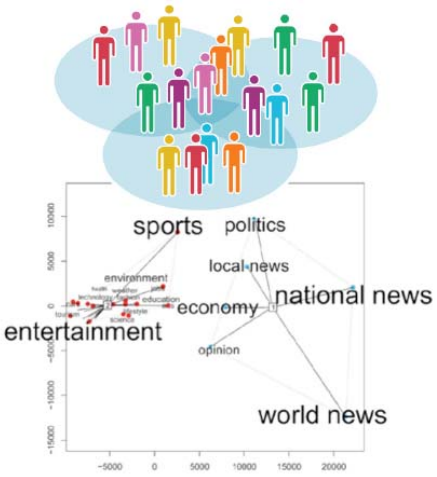
- 클러스터링 알고리즘은 입력된 데이터(뉴스 기사 등)들의 유사성(similarity)을 분석하여 그룹화된 카테고리로 분류
  - 라벨된 훈련 데이터가 필요 없는 비지도 학습



Marco Toledo Bastos, and Gabriela Zago SAGE

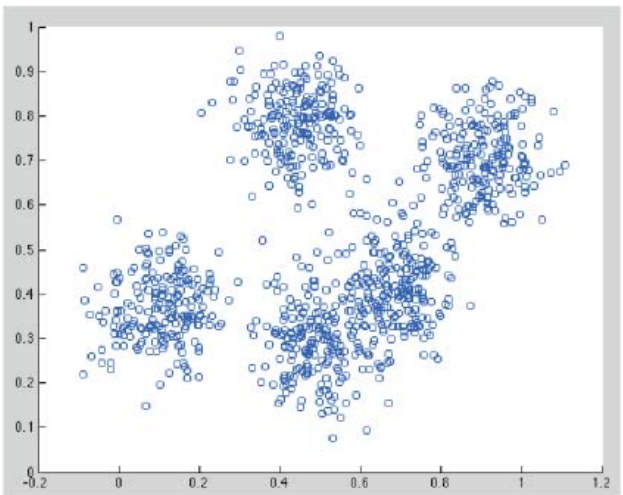
클러스터링 알고리즘 (3)

- 높은 유사성을 갖는 개체들은 클러스터로 그룹화
- 클러스터 적용 예
  - 검색 엔진의 검색 결과들의 그룹화
  - 고객 관리를 위한 고객들의 그룹화
  - 사기 감지(fraud detection)를 위한 비정상성 감지
  - 도서를 장르 별로 분류



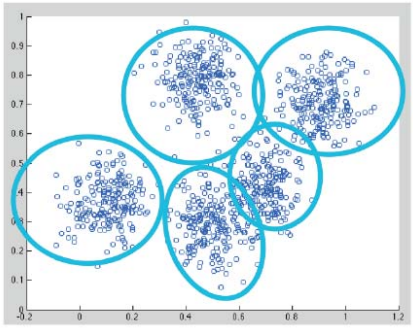
클러스터링 예 (1)

- 가장 유사성이 큰 데이터 포인트들을 그룹화 하는 예
- K-평균 알고리즘을 사용하여 클러스터링



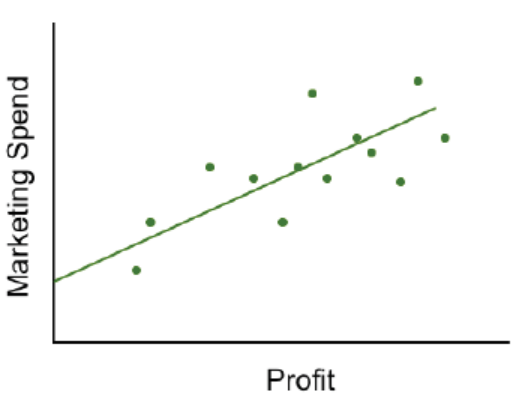
클러스터링 예 (2)

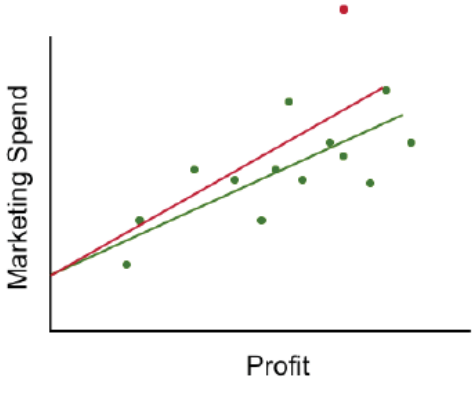
- K-평균 알고리즘 적용 절차
  1. K개의 클러스터(그룹)의 중심(centroid) 초기화
     - 무작위 분할 등 초기화 알고리즘 사용하여 초기화
  2. 모든 데이터 포인트에 대해 유클리드 거리(Euclidean distance)가 가장 가까운 중심에 할당
  3. 할당된 데이터 포인트들의 중앙이 되도록 중심 변경
  4. 종료 조건을 만족할 때까지 반복
     - 이전 반복과 차이가 없거나 클러스터 내의 거리가 충분히 작거나, 클러스터 간의 거리가 충분 큰 경우 종료



선형 회귀분석 (Linear Regression Analysis)

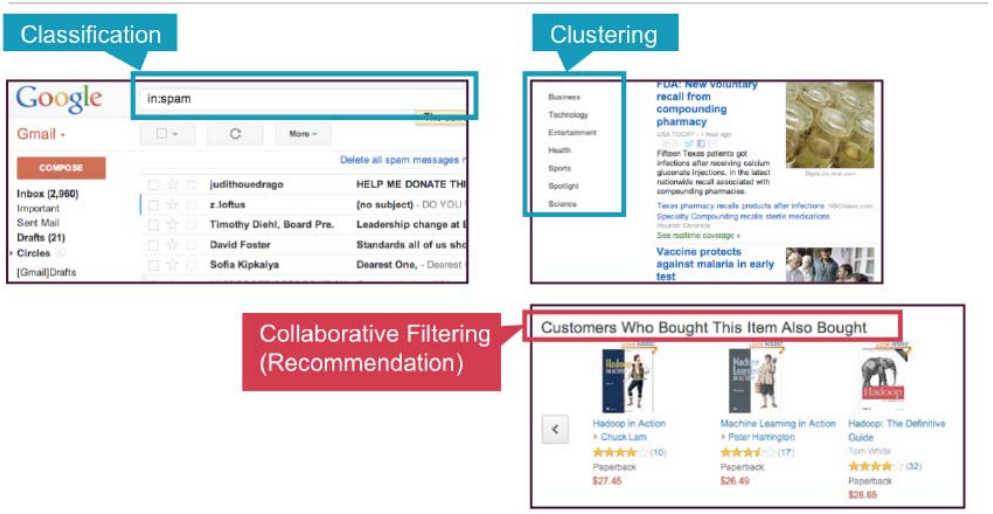
- 선형 회귀분석은 직선을 데이터 포인트들 간의 관계가 기울기를 결정하는 직선을 정의
  - 이상치(outlier)에 민감하여 기울기에 영향





협업 필터링 (1)

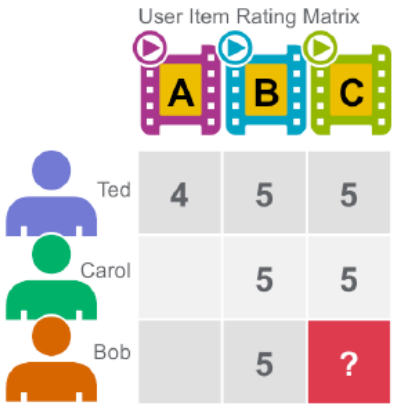
- 아마존은 사용자의 이력과 다른 사용자와의 유사성에 근거하여 사용자의 구매 의사가 있는 상품을 추천하는 알고리즘으로 협업 필터링(Collabaortive Filetring)을 적용



협업 필터링 (2)

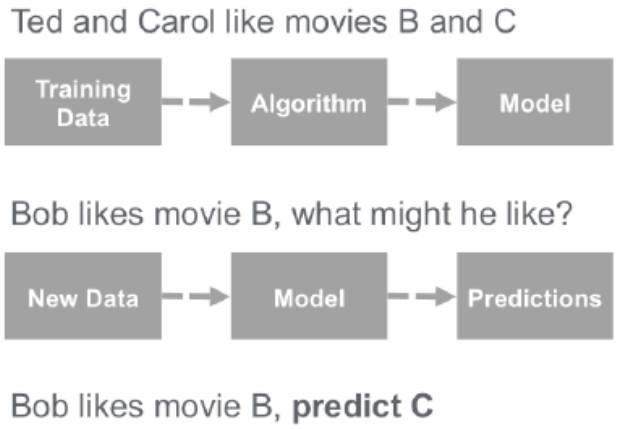
- 협업 필터링 알고리즘은은 많은 사용자들의 선호도 정보(preference information)에 기반으로 항목을 추천
  - 유사성(similarity)에 기반
  - 과거에 같은 항목들을 선호하는 사람들은 미래에도 비슷한 항목들을 선호

- 영화 예
  - Ted는 영화 A,B,C 선호
  - Carol은 B,C 선호
  - Bob은 B를 선호
  - Bob에 추천할 영화는?
    => C
    - B를 선호한 사용자들은 C도 선호



협업 필터링 (3)

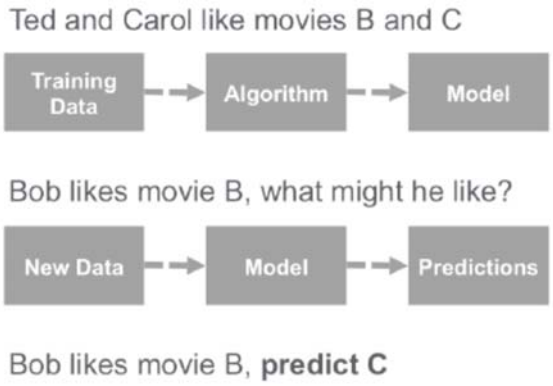
- 협업 필터링 알고리즘은 사용자들의 선호도 정보를 입력받아 추천 또는 예측에 사용되는 모델을 생성



3. 사용자 선택 예측을 위한 협업 필터링

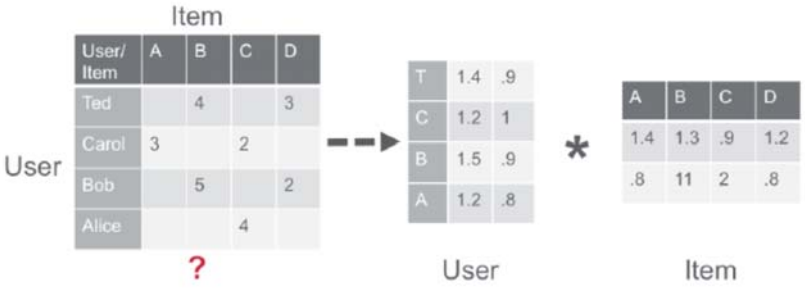
영화 추천 예

- 협업 필터링 알고리즘은 사용자의 영화 선호도를 입력받아 훈련(train)시켜 모델을 생성하고, 이 모델을 사용하여 사용자의 영화 선호도를 예측



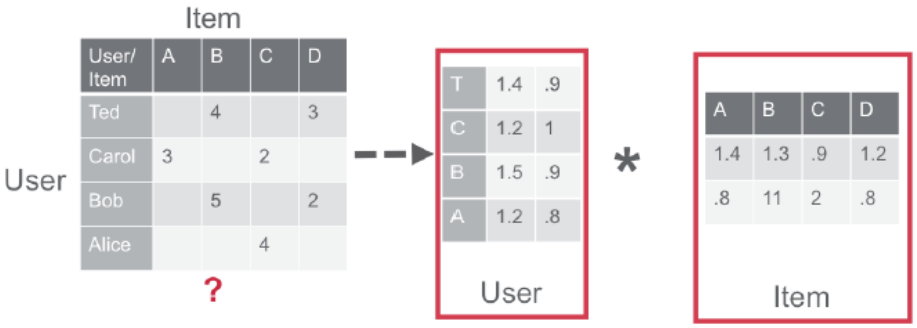
교차 최소 제곱 (1) (Alternating Least Squares, ALS)

- 교차 최소 제곱(ALS) 알고리즘
  - 다수의 사용자와 항목(영화) 사이에서 관측된 상호 작용(observed interaction, 영화 추천)을 상대적으로 적은 수의 관측되지 않은 숨은 원인(unobserved underlying reason)으로 설명하고 할 때 사용
  - 관측된 상호 작용은 희소 행렬(sparse matrix)로 표현
    - 행은 사용자, 열은 영화 (사용자는 많은 영화 중 일부만 선호)
  - 희소행렬은 작고 밀도가 높은 행렬 곱으로 분해



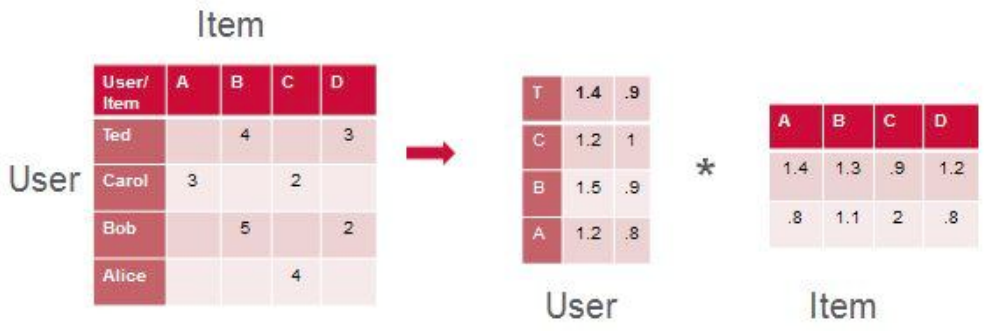
교차 최소 제곱 (2) (Alternating Least Squares, ALS)

- 각 사용자와 영화 항목의 숨겨진 특징을 학습
  - 사용자-잠재 요인 행렬 (user-latent factor matrix)
    - 각 사용자의 잠재적 또는 숨겨진 특징을 표현
  - 잠재 요인-영화 항목 행렬 (latent factor-movie item matrix)
    - 각 영화 항목의 잠재적 특징을 표현



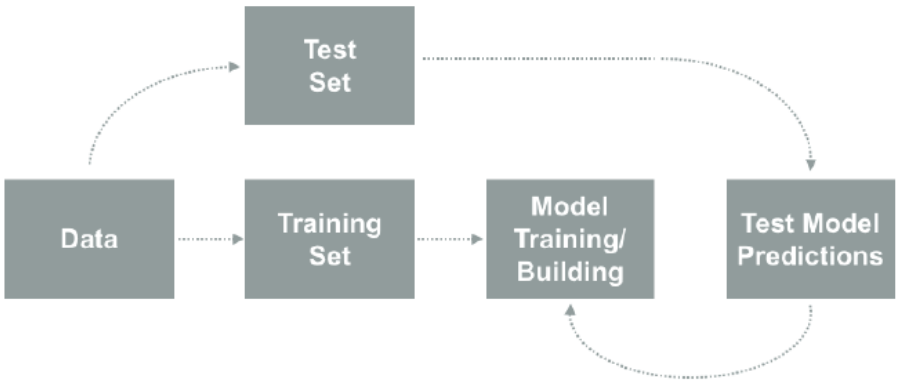
교차 최소 제곱 (3)

- ALS는 교대로(alternatively) 두 요인 행렬 중 하나를 고정시키고 다른 요인 행렬의 해(solution)을 구함.
- 수렴 시까지 반복적으로 수행되는 알고리즘(iterative algorithm)
  - 두 행렬 간의 대응되는 원소 간의 차의 제곱의 합을 최소화 (최소 제곱)



영화 추천 예 – 머신 러닝 처리 절차

- 영화 추천을 위한 머신 러닝 처리 절차
  - 샘플 데이터를 적재하고 ALS 알고리즘의 입력 형식을 변환
  - 모델을 훈련/구축 및 테스트 용으로 데이터를 분할
  - 모델을 훈련하고 구축
    - 훈련 데이터로 예측하고 결과 관측
  - 테스트 데이터로 모델 테스트



영화 추천 예 – 데이터 파일

- 영화 추천 사이트 데이터
  - Movielens: https://movielens.org/
  - Grouplens에서 데이터 제공: https://grouplens.org/datasets/
- 데이터 파일
  - ratings.dat: 사용자의 영화 평가
    - UserID::MovieID::Rating::Timestamp
  - movies.dat: 영화 정보
    - MovieID::Title::Genres
  - users.dat: 사용자 정보
    - UserID::Gender::Age::Occupation ::Zip-code

```
bigdata@slave1:~/spark/movie$ head -n3 ratings.dat
1::1193::5::978300760
1::661::3::978302109
1::914::3::978301968
bigdata@slave1:~/spark/movie$
bigdata@slave1:~/spark/movie$ head -n3 movies.dat
1::Toy Story (1995)::Animation|Children's|Comedy
2::Jumanji (1995)::Adventure|Children's|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
bigdata@slave1:~/spark/movie$
bigdata@slave1:~/spark/movie$ head -n3 users.dat
1::F::1::10::48067
2::M::56::16::70072
3::M::25::15::55117
```

데이터 다운로드

- 데이터 다운로드
  - $ mkdir ~/spark/movie
  - $ cd ~/spark/movie
  - $ wget http://cs.sch.ac.kr/lecture/BigData/download/ratings.dat
  - $ wget http://cs.sch.ac.kr/lecture/BigData/download/movies.dat
  - $ wget http://cs.sch.ac.kr/lecture/BigData/download/users.dat
- HDFS 적재
  - $ hadoop fs -mkdir /sparkdata/movie
  - $ hadoop fs -put ratings.dat /sparkdata/movie
  - $ hadoop fs -put movies.dat /sparkdata/movie
  - $ hadoop fs -put users.dat /sparkdata/movie


```
bigdata@slave1:~/spark/movie$ wget http://cs.sch.ac.kr/lecture/BigData/download/ratings.dat
--2019-08-14 07:45:43--  http://cs.sch.ac.kr/lecture/BigData/download/ratings.dat
Resolving cs.sch.ac.kr (cs.sch.ac.kr)... 220.69.209.31

bigdata@slave1:~/spark/movie$ wget http://cs.sch.ac.kr/lecture/BigData/download/ratings.dat
--2019-08-14 07:45:54--  http://cs.sch.ac.kr/lecture/BigData/download/ratings.dat
Resolving cs.sch.ac.kr (cs.sch.ac.kr)... 220.69.209.31

bigdata@slave1:~/spark/movie$ wget http://cs.sch.ac.kr/lecture/BigData/download/movies.dat
--2019-08-14 07:46:02--  http://cs.sch.ac.kr/lecture/BigData/download/movies.dat
Resolving cs.sch.ac.kr (cs.sch.ac.kr)... 220.69.209.31

bigdata@slave1:~/spark/movie$ head -n3 ratings.dat
1::1193::5::978300760
1::661::3::978302109
1::914::3::978301968
bigdata@slave1:~/spark/movie$
bigdata@slave1:~/spark/movie$ head -n3 movies.dat
1::Toy Story (1995)::Animation|Children's|Comedy
2::Jumanji (1995)::Adventure|Children's|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
bigdata@slave1:~/spark/movie$
bigdata@slave1:~/spark/movie$ head -n3 users.dat
1::F::1::10::48067
2::M::56::16::70072
3::M::25::15::55117
bigdata@slave1:~/spark/movie$ hadoop fs -mkdir /sparkdata/movie
bigdata@slave1:~/spark/movie$ hadoop fs -put ratings.dat /sparkdata/movie
bigdata@slave1:~/spark/movie$ hadoop fs -put movies.dat /sparkdata/movie
bigdata@slave1:~/spark/movie$ hadoop fs -put users.dat /sparkdata/movie
bigdata@slave1:~/spark/movie$
bigdata@slave1:~/spark/movie$ hadoop fs -ls /sparkdata/movie
Found 3 items
-rw-r--r--   3 bigdata supergroup     171308 2019-08-14 07:52 /sparkdata/movie/movies.dat
-rw-r--r--   3 bigdata supergroup   24594131 2019-08-14 07:52 /sparkdata/movie/ratings.dat
-rw-r--r--   3 bigdata supergroup     134368 2019-08-14 07:52 /sparkdata/movie/users.dat
bigdata@slave1:~/spark/movie$
```

데이터 스키마 정의

- 케이스 클래스 사용하여 평가, 영화, 사용자 데이터 스키마 정의 및 클래스 변환 함수 정의

```
// 패키지 임포트
import org.apache.spark.ml.evaluation.RegressionEvaluator
import org.apache.spark.ml.recommendation.ALS

//// 케이스 클래스 정의
// 평가 케이스클래스, UserID::MovieID:Rating::Timestamp
case class Rating(userId: Int, movieId: Int, rating: Float, timestamp: Long)

// 영화 케이스클래스, MovieID::Title::Genres
case class Movie(movieId: Int, title: String, genres: String)

// 사용자 케이스클래스, UserID::Gender::Age::Occupation::Zip-code
case class User(userId: Int, gender: String, age: Int, occupation: Int, zip: String)
```


데이터 스키마 정의 – 실행

```
// 패키지 임포트
import org.apache.spark.ml.evaluation.RegressionEvaluator
import org.apache.spark.ml.recommendation.ALS

//// 케이스 클래스 정의
// 평가 케이스클래스, UserID::MovieID:Rating::Timestamp
case class Rating(userId: Int, movieId: Int, rating: Float, timestamp: Long)

// 영화 케이스클래스, MovieID::Title::Genres
case class Movie(movieId: Int, title: String, genres: String)

// 사용자 케이스클래스, UserID::Gender::Age::Occupation::Zip-code
case class User(userId: Int, gender: String, age: Int, occupation: Int, zip: String)
```

```
import org.apache.spark.ml.evaluation.RegressionEvaluator
import org.apache.spark.ml.recommendation.ALS
defined class Rating
defined class Movie
defined class User
```

데이터 변환 함수 정의

- 입력 데이터의 라인을 케이스클래스로 변환하는 함수 정의

```
//// 입력 라인을 케이스클래스로 변환하는 함수
// 평가, UserID::MovieID::Rating::Timestamp
def parseRating(str: String): Rating = {
    val fields = str.split("::")
    Rating(fields(0).toInt, fields(1).toInt, fields(2).toFloat, fields(3).toLong)
}
// 영화, MovieID::Title::Genres
def parseMovie(str: String): Movie = {
    val fields = str.split("::")
    Movie(fields(0).toInt, fields(1), fields(2))
}
// 사용자, UserID::Gender::Age::Occupation::Zip-code
def parseUser(str: String):User = {
    val fields = str.split("::")
    assert(fields.size == 5)
    User(fields(0).toInt, fields(1).toString, fields(2).toInt, fields(3).toInt, fields(4).toString)
}
```


데이터 변환 함수 정의 – 실행

```
//// 입력 라인을 케이스클래스로 변환하는 함수
// 평가, UserID::MovieID::Rating::Timestamp
def parseRating(str: String): Rating = {
    val fields = str.split("::")
    Rating(fields(0).toInt, fields(1).toInt, fields(2).toFloat, fields(3).toLong)
}

// 영화, MovieID::Title::Genres
def parseMovie(str: String): Movie = {
    val fields = str.split("::")
    Movie(fields(0).toInt, fields(1), fields(2))
}

// 사용자, UserID::Gender::Age::Occupation::Zip-code
def parseUser(str: String):User = {
    val fields = str.split("::")
    assert(fields.size == 5)
    User(fields(0).toInt, fields(1).toString, fields(2).toInt, fields(3).toInt, fields(4).toString)
}
```

```
parseRating: (str: String)Rating
parseMovie: (str: String)Movie
parseUser: (str: String)User
```

평가 데이터프레임 생성

- 평가 데이터를 적재하고 데이터프레임 생성
  - 평가, 영화, 사용자 총 갯수 카운트

```
// ratings.dat 적재 후 데이터프레임 생성
val ratingsDF =
    spark.read.textFile("/sparkdata/movie/ratings.dat").map(parseRating).toDF().
    cache()
ratingsDF.first()

// 평가, 영화, 사용자 총 갯수 카운트
val numRatings = ratingsDF.count()
val numMovies = ratingsDF.select($"movieId").distinct().count()
val numUsers = ratingsDF.select($"userId").distinct().count()
```


평가 데이터프레임 생성 – 실행

```
// ratings.dat 적재 후 데이터프레임 생성
val ratingsDF = spark.read.textFile("/sparkdata/movie/ratings.dat").map(parseRating).toDF().cache()
ratingsDF.first()
```

```
ratingsDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [userId: int, movieId: int ... 2
res139: org.apache.spark.sql.Row = [1,1193,5.0,978300760]
```

Took 7 sec. Last updated by admin at August 14 2019, 5:23:10 PM.

```
// 평가, 영화, 사용자 총 갯수 카운트
val numRatings = ratingsDF.count()
val numMovies = ratingsDF.select($"movieId").distinct().count()
val numUsers = ratingsDF.select($"userId").distinct().count()

// 결과 출력
println(s"영화 = $numMovies 편, 사용자 = $numUsers 명, 평가 = $numRatings 개")
```

```
numRatings: Long = 1000209
numMovies: Long = 3706
numUsers: Long = 6040
영화 = 3706 편, 사용자 = 6040 명, 평가 = 1000209 개
```

영화/사용자 데이터프레임 생성

- 영화, 사용자 데이터프레임 생성하고 뷰로 등록

```
// 영화, 사용자 데이터프레임 생성
val moviesDF =
    spark.read.textFile("/sparkdata/movie/movies.dat").map(parseMovie).toDF()
val usersDF = spark.read.textFile("/sparkdata/movie/users.dat").map(parseUser).toDF()

// 데이터프레임을 뷰로 등록
ratingsDF.createOrReplaceTempView("ratings")
moviesDF.createOrReplaceTempView("movies")
usersDF.createOrReplaceTempView("users")

// 스키마 프린트
ratingsDF.printSchema()
moviesDF.printSchema()
usersDF.printSchema()
```


영화/사용자 데이터프레임 생성 – 실행

```
// 영화, 사용자 데이터프레임 생성
val moviesDF = spark.read.textFile("/sparkdata/movie/movies.dat").map(parseMovie).toDF()
val usersDF = spark.read.textFile("/sparkdata/movie/users.dat").map(parseUser).toDF()

// 데이터프레임을 뷰로 등록
ratingsDF.createOrReplaceTempView("ratings")
moviesDF.createOrReplaceTempView("movies")
usersDF.createOrReplaceTempView("users")

// 스키마 프린트
ratingsDF.printSchema()
moviesDF.printSchema()
usersDF.printSchema()

moviesDF: org.apache.spark.sql.DataFrame = [movieId: int, title: string ... 1 more field]
usersDF: org.apache.spark.sql.DataFrame = [userId: int, gender: string ... 3 more fields]
root
 |-- userId: integer (nullable = false)
 |-- movieId: integer (nullable = false)
 |-- rating: float (nullable = false)
 |-- timestamp: long (nullable = false)

root
 |-- movieId: integer (nullable = false)
 |-- title: string (nullable = true)
 |-- genres: string (nullable = true)

root
 |-- userId: integer (nullable = false)
 |-- gender: string (nullable = true)
 |-- age: integer (nullable = false)
 |-- occupation: integer (nullable = false)
```

데이터프레임 질의 1

- 데이터프레임 질의로 데이터 조사
  - 평가된 영화의 제목, 최대 점수, 최소 점수, 평가자 수 조사
  - 평가자 수의 내림차순으로 정렬

```
// 평가된 영화의 제목, 최대 점수, 최소 점수, 평가자 수 조사
val results = spark.sql("select movies.title, movierates.maxr, movierates.minr,
    movierates.cntu from (SELECT ratings.movieId, max(ratings.rating) as maxr,
    min(ratings.rating) as minr,count(distinct userId) as cntu FROM ratings group
    by ratings.movieId ) movierates join movies on movierates.movieId =
    movies.movieId order by movierates.cntu desc")
results.show(10)
```


데이터프레임 질의 1 – 실행

SPARK JOBS FINISHED

```
// 평가된 영화의 제목, 최대 점수, 최소 점수, 평가자 수 조사
val results = spark.sql("select movies.title, movierates.maxr, movierates.minr, movierates.cntu from(SELECT
    ratings.movieId, max(ratings.rating) as maxr, min(ratings.rating) as minr,count(distinct userId) as cntu
    FROM ratings group by ratings.movieId ) movierates join movies on movierates.movieId = movies.movieId order
    by movierates.cntu desc")
results.show(10)
```

```
results: org.apache.spark.sql.DataFrame = [title: string, maxr: float ... 2 more fields]
+--------------------+----+----+----+
|               title|maxr|minr|cntu|
+--------------------+----+----+----+
|American Beauty (...| 5.0| 1.0|3428|
|Star Wars: Episod...| 5.0| 1.0|2991|
|Star Wars: Episod...| 5.0| 1.0|2990|
|Star Wars: Episod...| 5.0| 1.0|2883|
|Jurassic Park (1993)| 5.0| 1.0|2672|
|Saving Private Ry...| 5.0| 1.0|2653|
|Terminator 2: Jud...| 5.0| 1.0|2649|
|  Matrix, The (1999)| 5.0| 1.0|2590|
|Back to the Futur...| 5.0| 1.0|2583|
|Silence of the La...| 5.0| 1.0|2578|
+--------------------+----+----+----+
only showing top 10 rows
```

데이터프레임 질의 2

- 가장 평가를 많이한 10명의 사용자가 평가한 영화 수
  - 평가한 영화 수의 내림차순으로 정렬

```
// 가장 평가를 많이한 10명의 사용자가 평가한 영화 수
val mostActiveUsersDF = spark.sql("SELECT ratings.userId, count(*) as ct from
    ratings group by ratings.userId order by ct desc limit 10")
mostActiveUsersDF.show(10)
```


데이터프레임 질의 2 – 실행

SPARK JOB FINISHED

```
// 가장 평가를 많이한 10명의 사용자가 평가한 영화 수
val mostActiveUsersDF = spark.sql("SELECT ratings.userId, count(*) as ct from ratings group by ratings.userId
    order by ct desc limit 10")
mostActiveUsersDF.show(10)

mostActiveUsersDF: org.apache.spark.sql.DataFrame = [userId: int, ct: bigint]
+------+----+
|userId|  ct|
+------+----+
|  4169|2314|
|  1680|1850|
|  4277|1743|
|  1941|1595|
|  1181|1521|
|   889|1518|
|  3618|1344|
|  2063|1323|
|  1150|1302|
|  1015|1286|
+------+----+
```

데이터프레임 질의 3

- 사용자 ID 4169가 평가한 영화 중 4점 이상이 되는 영화
  - 평가 점수의 내림차순으로 정렬

```
// 사용자 ID 4169가 평가한 영화 중 4점 이상이 되는 영화 조사
val results = spark.sql("SELECT ratings.userId, ratings.movieId, ratings.rating,
    movies.title FROM ratings JOIN movies ON movies.movieId = ratings.movieId
    where ratings.userId = 4169 and ratings.rating > 4")
results.show
```


데이터프레임 질의 3 – 실행

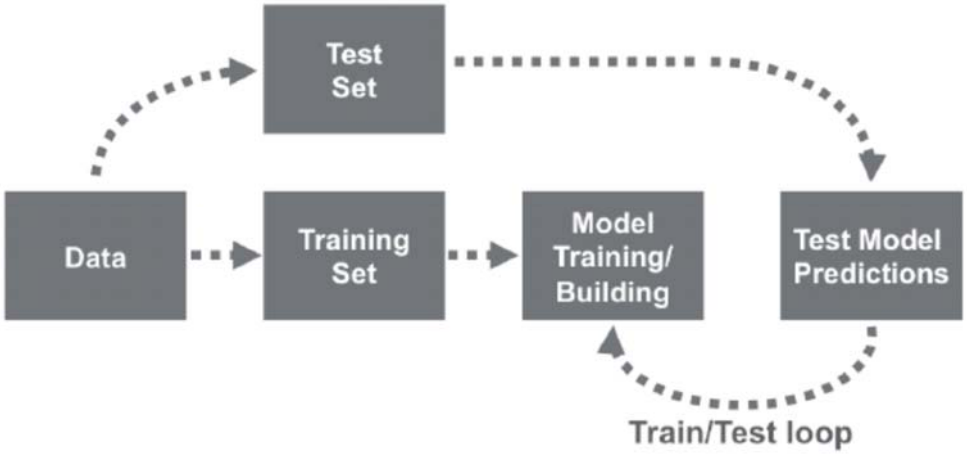
SPARK JOBS FINISHED

```
// 사용자 ID 4169가 평가한 영화 중 4점 이상이 되는 영화 조사
val results = spark.sql("SELECT ratings.userId, ratings.movieId, ratings.rating, movies.title FROM ratings JOIN
    movies ON movies.movieId = ratings.movieId where ratings.userId = 4169 and ratings.rating > 4")
results.show
```

```
results: org.apache.spark.sql.DataFrame = [userId: int, movieId: int ... 2 more fields]
+------+-------+------+--------------------+
|userId|movieId|rating|               title|
+------+-------+------+--------------------+
|  4169|   3789|   5.0|Pawnbroker, The (...|
|  4169|   3006|   5.0| Insider, The (1999)|
|  4169|   1408|   5.0|Last of the Mohic...|
|  4169|   2065|   5.0|Purple Rose of Ca...|
|  4169|   2066|   5.0|Out of the Past (...|
|  4169|   2068|   5.0|Fanny and Alexand...|
|  4169|    590|   5.0|Dances with Wolve...|
|  4169|    593|   5.0|Silence of the La...|
|  4169|    594|   5.0|Snow White and th...|
|  4169|    595|   5.0|Beauty and the Be...|
|  4169|    596|   5.0|    Pinocchio (1940)|
|  4169|   3011|   5.0|They Shoot Horses...|
|  4169|   1411|   5.0|       Hamlet (1996)|
|  4169|   1416|   5.0|        Evita (1996)|
```

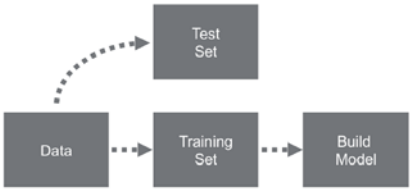
영화 추천 예 – ALS 처리 절차

- 머신 러닝 처리 절차에 따라 영화 추천 예 적용



모델 적용 데이터 분할

- 모델을 훈련(모델 구축) 및 테스트 용으로 데이터프레임을 분할
  - 훈련용 데이터 (80%)
  - 테스트 데이터 (20%)

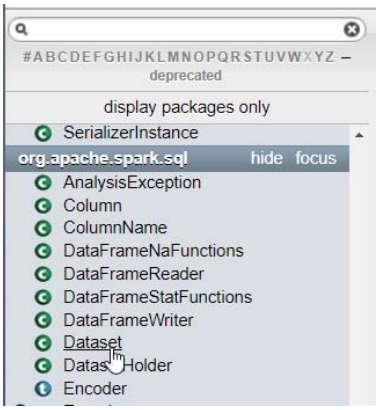



```
//// 협업 필터링을 사용한 영화 추천
// 모델을 훈련(80%) 및 테스트(20%) 용으로 데이터프레임을 분할
val splits = ratingsDF.randomSplit(Array(0.8, 0.2))
val trainingRatingsDF = splits(0).cache()                    // 훈련용 데이터프레임
val testRatingsDF = splits(1).cache()                        // 테스트용 데이터프레임

// 훈련 및 테스트용 데이터 카운트
val numTraining = trainingRatingsDF.count()
val numTest = testRatingsDF.count()
println(s"Training: $numTraining, test: $numTest.")
```

참조 – 스파크 API 문서

- 스파크 스칼라 API 문서는 아래 사이트 참조
  - https://spark.apache.org/docs/latest/api/scala/index.html
- 데이터프레임의 randomSplit() 함수 참조 예
  - org.apache.spark.sql 패키지에서 Dataset 클래스 참조
    - 데이터프레임은 Dataset[Row]와 동일
  - Dataset 클래스의 Typed transformations에서 randomSplit() 함수 참조



org.apache.spark.sql
Dataset

`class Dataset[T] extends Serializable`

Typed transformations

`def alias(alias: Symbol): Dataset[T]`
(Scala-specific) Returns a new Dataset with an alias set.

`def randomSplit(weights: Array[Double]): Array[Dataset[T]]`
Randomly splits this Dataset with the provided weights.

weights	weights for splits, will be normalized if they don't sum to 1.
Since	2.0.0

`def randomSplit(weights: Array[Double], seed: Long): Array[Dataset[T]]`
Randomly splits this Dataset with the provided weights.

모델 적용 데이터 분할 – 실행

```
//// 협업 필터링을 사용한 영화 추천
// 모델을 훈련(80%) 및 테스트(20%) 용으로 데이터프레임을 분할
val splits = ratingsDF.randomSplit(Array(0.8, 0.2))
val trainingRatingsDF = splits(0).cache()cache          // 훈련용 데이터프레임
val testRatingsDF = splits(1).cache()                   // 테스트용 데이터프레임

// 훈련 및 테스트용 데이터 카운트
val numTraining = trainingRatingsDF.count()
val numTest = testRatingsDF.count()
println(s"Training: $numTraining, test: $numTest.")
```

```
splits: Array[org.apache.spark.sql.Dataset[org.apache.spark.sql.Row]] = Array([userId:
fields], [userId: int, movieId: int ... 2 more fields])
warning: there was one feature warning; re-run with -feature for details
trainingRatingsDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [userId: ir
ields]
testRatingsDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [userId: int, m
s]
numTraining: Long = 800314
numTest: Long = 199895
Training: 800314, test: 199895.
```

모델 구축

- 사용자의 영화 선호도(훈련 데이터)를 입력받아 훈련(train) 시켜 모델을 구축
  - 모델 생성 시 파라미터
    - 사용자와 항목에 해당하는 데이터의 필드 지정
    - Iteration은 모델 구축 시 최대 반복 횟수 (디폴트 10)
    - Rank는 모델의 잠재 요인(특징)의 수 (디폴트 10)
    - 유효하지 않은 예측 NaN(Not a Number)을 제거하기 위해 setColdStartStrategy("drop") 적용

```
// 훈련 데이터 지정하여 모델 구축
val model = new
    ALS().setMaxIter(10).setRank(20).setUserCol("userId").setItemCol("movieId")
    .setRatingCol("rating").setColdStartStrategy("drop").fit(trainingRatingsDF)
```

```
// 훈련 데이터 지정하여 모델 구축
val model = new ALS().setMaxIter(10).setRank(20).setUserCol("userId").setItemCol("movieId").setRatingCol("rating"
    ).setColdStartStrategy("drop").fit(trainingRatingsDF)
```

SPARK JOBS FINISHED

```
model: org.apache.spark.ml.recommendation.ALSModel = als_5b5f5d3e36ca
```


테스트 데이터 추천

- 테스트 데이터를 모델에 적용하여 예측된 추천 결과 생성


```
// 테스트 데이터를 모델에 적용하여 추천
val predictionsDF = model.transform(testRatingsDF)
predictionsDF.createOrReplaceTempView("predictions")    // 뷰 등록

predictionsDF.count
predictionsDF.show(5)
```

테스트 데이터 추천 – 실행

```
// 테스트 데이터를 모델에 적용하여 추천
val predictionsDF = model.transform(testRatingsDF)
predictionsDF.createOrReplaceTempView("predictions")    // 뷰 등록

predictionsDF.count
predictionsDF.show(5)
```

```
predictionsDF: org.apache.spark.sql.DataFrame = [userId: int, movieId:
res58: Long = 199895
+------+-------+------+---------+----------+
|userId|movieId|rating|timestamp|prediction|
+------+-------+------+---------+----------+
|  3184|    148|   4.0|968708953|  3.165064|
|  4387|    148|   1.0|977034181| 2.7107003|
|  4784|    148|   3.0|970000570| 2.9670832|
|  3829|    148|   2.0|965940170| 2.5618804|
|  3650|    463|   2.0|966459084| 2.6768384|
+------+-------+------+---------+----------+
only showing top 5 rows
```


사용자 추천 결과 조회 (1)

- 사용자(4169)를 위한 추천 영화 Top 5 조사

```
// 사용자(4169)를 위한 추천 영화 Top 5 조사
val movieTitlesForUser = spark.sql("select userId, movieId, prediction from
    predictions where userID = 4169 order by prediction desc limit 5")
movieTitlesForUser.show(5)
```

SPARK JOB FINISHED

```
// 사용자(4169)를 위한 추천 영화 Top 5 조사
val movieTitlesForUser = spark.sql("select userId, movieId, prediction from predictions where userID = 4169 order
    by prediction desc limit 5")
movieTitlesForUser.show(5)
```

```
movieTitlesForUser: org.apache.spark.sql.DataFrame = [userId: int, movieId: int ... 1 more field]
+------+-------+----------+
|userId|movieId|prediction|
+------+-------+----------+
|  4169|   2905|  4.607808|
|  4169|   1207| 4.5629587|
|  4169|    912| 4.5543933|
|  4169|   1131|  4.498174|
|  4169|   1212|  4.489414|
+------+-------+----------+
```

사용자 추천 결과 조회 (2)

- 사용자(4169)를 위한 추천 영화 Top 5 조사
  - 영화 ID가 아닌 영화 제목 출력
  - 영화 테이블과 조인하는 질의

```
// 사용자(4169)를 위한 추천 영화 Top 5 조사
val movieTitlesForUser = spark.sql("select userId, movies.title, prediction from
    predictions join movies on predictions.movieId = movies.movieId where
    userID = 4169 order by prediction desc limit 5")
movieTitlesForUser.show(5)
```

SPARK JOBS FINISHED

```
// 사용자(4169)를 위한 추천 영화 Top 5 조사
val movieTitlesForUser = spark.sql("select userId, movies.title, prediction from predictions join movies on
    predictions.movieId = movies.movieId where userID = 4169 order by prediction desc limit 5")
movieTitlesForUser.show(5)

movieTitlesForUser: org.apache.spark.sql.DataFrame = [userId: int, title: string ... 1 more field]
+------+--------------------+----------+
|userId|               title|prediction|
+------+--------------------+----------+
|  4169|      Sanjuro (1962)|  4.607808|
|  4169|To Kill a Mocking...| 4.5629587|
|  4169|   Casablanca (1942)| 4.5543933|
|  4169|Jean de Florette ...|  4.498174|
|  4169|Third Man, The (1...|  4.489414|
+------+--------------------+----------+
```


거짓 양성 (False Positive) 조사

- 영화 추천에서 거짓 양성 조사
  - 테스트 데이터에서 예측 추천은 4 이상(양성)으로 추천했지만 실제 평가는 1 이하로 틀린 추천(거짓 양성) 조사

```
// 테스트 데이터에서 예측 추천은 4 이상(양성)으로 추천했지만 실제 평가는 1 이하로 틀
    린 추천(거짓 양성) 조사
val results = spark.sql("select userId, movieId, rating, prediction from predictions
    where prediction >= 4 and rating <= 1")
results.count
results.show(5)
```

```
results: org.apache.spark.sql.DataFrame = [userId: int, movieId: int ... 2 more fields]
res81: Long = 111
+------+-------+------+----------+
|userId|movieId|rating|prediction|
+------+-------+------+----------+
|  1645|   1088|   1.0| 4.4125257|
|  2793|   3175|   1.0|  4.005207|
|   188|    858|   1.0| 4.2515364|
|  1790|   1303|   1.0|  4.035326|
|   372|   3000|   1.0| 4.4219136|
+------+-------+------+----------+
only showing top 5 rows
```

모델 평가

- 실제 영화 평가와 예측 평가 사이의 오류율 계산
  - 평균 절대값 오차(Mean Absolute Error, MAE) 계산
  - 사용자 정의 함수(UDF) 사용

```
// 평균 절대값 오차(Mean Absolute Error, MAE) 계산 사용자 정의 함수
def AbsoluteError(testR:Float, predR:Float): Double = {
   val err = (testR - predR)
   Math.abs(err)
}

// UDF 등록
spark.udf.register("AbsErr", AbsoluteError(_:Float, _:Float))

// 평균 절대값 오차를 사용하여 모델 평가
spark.sql("select avg(AbsErr(rating,prediction)) as MAE from predictions").show
```


모델 평가 – 실행

SPARK

```
// 평균 절대값 오차(Mean Absolute Error, MAE) 계산 사용자 정의 함수
def AbsoluteError(testR:Float, predR:Float): Double = {
    val err = (testR - predR)
    Math.abs(err)
}

// UDF 등록
spark.udf.register("AbsErr", AbsoluteError(_:Float, _:Float))

// 평균 절대값 오차를 사용하여 모델 평가
spark.sql("select avg(AbsErr(rating,prediction)) as MAE from predictions").show
```

```
AbsoluteError: (testR: Float, predR: Float)Double
res86: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(<functi
loatType, FloatType)))
+------------------+
|               MAE|
+------------------+
|0.6974450569541301|
+------------------+
```

모델 훈련 데이터 추가

- 기존의 평가 데이터프레임에 새로운 데이터를 추가하여 모델을 구축하는 예

```
// 사용자 0의 평가 추가,  평가 케이스클래스 UserID::MovieID:Rating::Timestamp
val data = Seq(Rating(0,260,4, 988300760), Rating(0,1,3, 988300765), Rating(0,16,3,
    988300770), Rating(0,25,4, 988300776), Rating(0,32,4, 988300788),
    Rating(0,335,1, 988300870), Rating(0,379,1, 988301770), Rating(0,296,3,
    989300770), Rating(0,858,5, 989300775), Rating(0,50,4, 989300780))

// 새로운 데이터프레임을 기존 평가 데이터프레임에 추가
val newRatingsDF = data.toDF()
val unionRatingsDF = ratingsDF.union(newRatingsDF)

// 추가된 데이터프레임 사용하여 모델 구축
val model = new
    ALS().setMaxIter(10).setRank(20).setRegParam(0.01).setUserCol("userId").setItemCol(
    "movieId").setRatingCol("rating").fit(unionRatingsDF)
// 이 후 모델의 추천 적용은 동일
// .....
```


모델 훈련 데이터 추가 – 실행

SPARK JOBS FINISHED

```
// 사용자 0의 평가 추가
// 평가 케이스클래스, UserID::MovieID:Rating::Timestamp
val data = Seq(Rating(0,260,4, 988300760),Rating(0,1,3, 988300765),Rating(0,16,3, 988300770),
Rating(0,25,4, 988300776),Rating(0,32,4, 988300788),Rating(0,335,1, 988300870),Rating(0,379,1, 988301770),
Rating(0,296,3, 989300770),Rating(0,858,5, 989300775),Rating(0,50,4, 989300780))

// 새로운 데이터프레임을 기존 평가 데이터프레임에 추가
val newRatingsDF = data.toDF()
val unionRatingsDF = ratingsDF.union(newRatingsDF)

// 추가된 데이터프레임 사용하여 모델 구축
val model = new ALS().setMaxIter(10).setRank(20).setRegParam(0.01).setUserCol("userId").setItemCol("movieIc
    ).setRatingCol("rating").fit(unionRatingsDF)

// 이 후 모델의 추천 적용은 동일
// .....
```

```
data: Seq[Rating] = List(Rating(0,260,4.0,988300760), Rating(0,1,3.0,988300765), Rating(0,16,3.0,988300770),
(0,25,4.0,988300776), Rating(0,32,4.0,988300788), Rating(0,335,1.0,988300870), Rating(0,379,1.0,988301770),
(0,296,3.0,989300770), Rating(0,858,5.0,989300775), Rating(0,50,4.0,989300780))
newRatingsDF: org.apache.spark.sql.DataFrame = [userId: int, movieId: int ... 2 more fields]
unionRatingsDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [userId: int, movieId: int ... 2 mo
s]
model: org.apache.spark.ml.recommendation.ALSModel = als_8020c7fc0d07
```

과제

- 강의 시간의 실습 내용을 정리하여 제출
- 텀 프로젝트 과제
  - 텀 프로젝트 데이터를 사용하여 앞에서 배운 스파크 응용 모니터링을 적용하고 실행
  - 자신의 주제에 맞추어 ALS 뿐만 아니라 어떤 종류의 Mlib 알고리즘 적용도 가능

참고 자료

- Advanced Apache Spark
  - https://learn.mapr.com/series/sparkv2/dev-362-advanced-apache-spark-spark-v21
    - Lesson 8: Use Apache Spark MLib
- Machine Learning Library (MLib) Guide
  - https://spark.apache.org/docs/latest/ml-guide.html
- Spark Collaborative Filtering
  - https://spark.apache.org/docs/latest/ml-collaborative-filtering.html
- Apache Spark Machine Learning Tutorial
  - https://mapr.com/blog/apache-spark-machine-learning-tutorial/
- Matrix Factorization Techniques for Recommender Systems
  - https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf