MLlib로 해보는 머신러닝

아꿈사 박주희

Spark ML 개요

- MLlib의 디자인과 철학은 단순: '결국 RDD를 사용하는 ML 함수 집합.'

- ML을 이용한 text classification 작업의 예

  1. 메시지를 표현하는 문자열들이 담긴 RDD를 준비.

  2. MLlib 특성 추출 알고리즘들 중 하나를 써서 문자열을 수치화된 특성으로 변환을 통한 결과 벡터들의 RDD 리턴.

  3. 벡터의 RDD에 분류 알고리즘을 호출. 신규 데이터들을 분류하는데 쓰일 모델 객체 리턴.

  4. MLlib 평가 함수들 중 하나를 써서 테스트 데이터 세트에 모델을 적용, 평가

- MLlib은 병렬 클러스터에 적합한 알고리즘(RF, K-Mn, ALS) 을 주로 보유하므로, 작은 데이터셋을 가지고 여러 학습모델을 적용할 경우는 각 노드에 단일 노드용 알고리즘을 적용하고 스파크 Map()을 통해 병렬로 처리하는 것이 낫다.

- 더불어 다양한 설정에 따른 최적의 알고리즘 해를 찾을 경우는 Parallelize()를 적용하고, 각 노드에 단일 머신 알고리즘을 적용하여 테스트 할수 있다.

Spark ML 개발(cont'd)

- 현재 MLlib 구성은 아래와 같다.

  - spark.mllib contains the original API built on top of RDDs.
  - spark.ml provides higher-level API built on top of DataFrames for constructing ML pipelines.

- mllib는 다양한 ml개발자 및 학자들이 새로운 알고리즘등을 만들고 테스트를 지원하기 위한것으로, 사실상 spark.ml를 사용하는 것을 추천함.

시스템 요구사항

- MLlib은 시스템에 몇몇 선형대수 라이브러리 설치를 필요로 함.
  만약 해당 오류 발생시 우선 OS에 gfortran 실행 라이브러리를 설치.

- Dependencies.
  - 스칼라, 자바 : breeze
  - 파이썬 : numpy

- 파이썬 numpy 설치는 아래와 같이....
  - sudo yum install numpy scipy python-matplotlib ipython python-pandas sympy python-nose

머신러닝의 기초

- 머신러닝 알고리즘은 훈련 데이터를 통한 알고리즘의 수학적 결과들을 토대로 예측, 분류 등에 대한 결정을 내리려고 시도한다.

- 머신러닝의 일반적인 단계:
문제 정의 -> 변수 추출 -> 정제 및 변환 -> 분석 ->측정

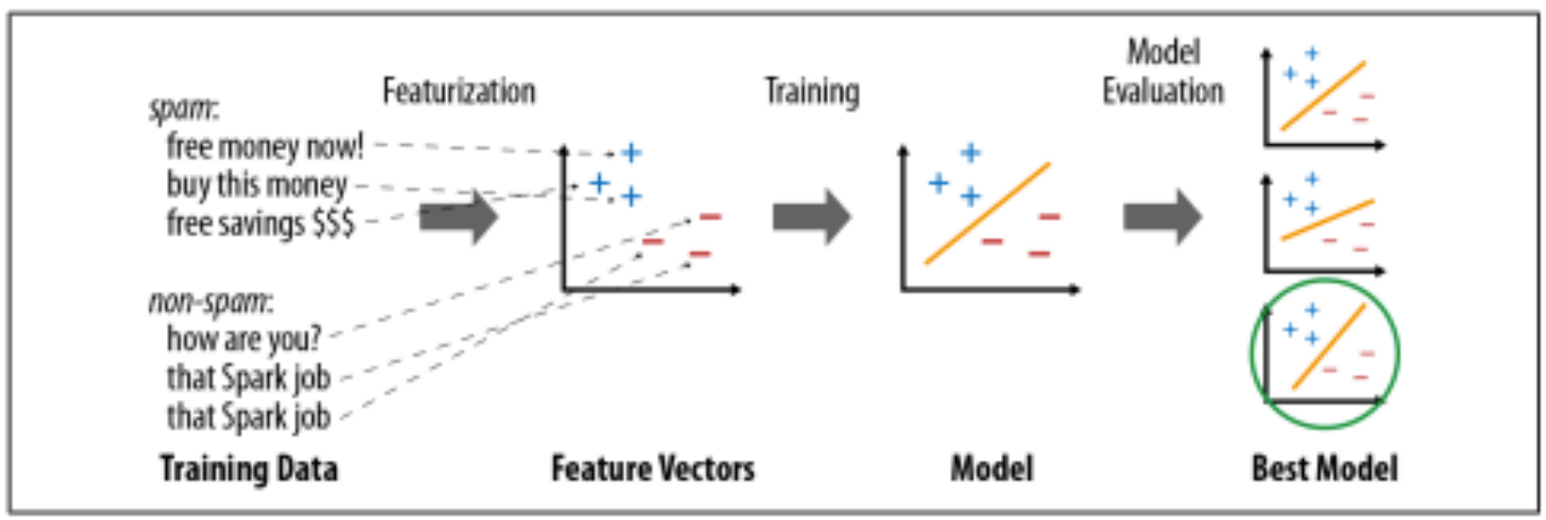

Figure 11-1. Typical steps in a machine learning pipeline

예제) 스팸 분류

```
>>> from pyspark.mllib.regression import LabeledPoint
>>> from pyspark.mllib.feature import HashingTF
>>> from pyspark.mllib.classification import LogisticRegressionWithSGD
>>> spam = sc.textFile("hdfs:///tmp/spam.txt")
>>> normal = sc.textFile("hdfs:///tmp/ham.txt")
>>> tf = HashingTF(numFeatures = 10000)
>>> spamFeatures = spam.map(lambda email: tf.transform(email.split(" ")))
>>> normalFeatures = normal.map(lambda email: tf.transform(email.split(" ")))
>>> positiveExamples = spamFeatures.map(lambda features: LabeledPoint(1, features))
>>> negativeExamples = normalFeatures.map(lambda features: LabeledPoint(0, features))
>>> trainingData = positiveExamples.union(negativeExamples)
>>> trainingData.cache()
>>> model = LogisticRegressionWithSGD.train(trainingData)
15/11/12 02:30:30 INFO TaskSetManager: Finished task 2.0 in stage 103.0 (TID 408) in 13 ms onserver02.hadoop.com (3/4)
15/11/12 02:30:30 INFO DAGScheduler: Stage 103 (treeAggregate at GradientDescent.scala:189) finished in 0.067 s
15/11/12 02:30:30 INFO DAGScheduler: Job 103 finished: treeAggregate at GradientDescent.scala:189, took 0.076100 s
15/11/12 02:30:30 INFO GradientDescent: GradientDescent.runMiniBatchSGD finished. Last 10 stochastic losses
0.0702147385476085, 0.07011160584006985, 0.07001068362068279, 0.06991190453587529, 0.06981520392578779,
0.0697205196905890, 0.0696277921647086, 0.06953696399844017, 0.06944798004641789, 0.06936078726250021
15/11/12 02:30:30 INFO TaskSetManager: Finished task 3.0 in stage 103.0 (TID 409) in 11 ms onserver02.hadoop.com (4/4)
15/11/12 02:30:30 INFO YarnScheduler: Removed TaskSet 103.0, whose tasks have all completed, from pool
15/11/12 02:30:30 INFO MapPartitionsRDD: Removing RDD 9 from persistence list
15/11/12 02:30:30 INFO BlockManager: Removing RDD 9
>>> posTest = tf.transform("O M G GET cheap stuff by  sending money to .... ".split(" "))
>>> negTest = tf.transform("Hi Dad, I started studying Spark the other ... ".split(" "))
>>> print "Prediction for positive test examples: %g" % model.predict(posTest)
        Prediction for positive test examples: 1
>>> print "Prediction for negative test examples: %g" % model.predict(negTest)
        Prediction for negative test examples: 0
```

데이터 타입

- MLlib에서의 특정 타입들의 패키지 위치 :
  - 자바, 스칼라: org.apache.spark.mllib
  - 파이썬: pyspark.mllib

- 주요 데이터 타입.
  - Local vector : 수학적인 의미의 벡터, 고밀도 벡터와 저밀도 벡터를 모두 지원함.
  - Labeled point : 분류나 회귀같은 supervised 학습 알고리즘을 위한 Labeled Data Point
    (쉽게 생각하면 변수이름 + 수치 데이터의 벡터)
  - Rating : 사용자 상품 순위, 상품 추천을 위한 mllib.recommendation 패키지에서 제공.
  - 다양한 Model 클래스 : 학습된 알고리즘의 결과들의 형태로, 신규 데이터(RDD, Data Point등)
    에 대한 예측을 위한 Predict() 함수를 제공.

working with vectors

- 벡터 사용시 몇 가지 주의할 점:

1. 벡터들은 두 가지 타입으로 구성됨
    - dense, sparse (0이 아닌 값이 10% 이하 여부에 따라 선택)

2. 벡터를 생성하는 방식이 언어별로 다름.
    - 파이썬: numpy 배열 or mllib.linalg.vectors 사용.
    - 자바, 스칼라: mllib.linalg.vectors

3. 벡터 연산 제공 유무
    - 파이썬 : numpy의 경우 수학연산 가능.
    - 자바, 스칼라 : 데이터 표현만 가능.

알고리즘

- Feature Extraction
  - mllib.feature 패키지는 일반적인 특성 트랜스포메이션(변수 변환)들을 위한 여러 클래스를 보유.
  - 문자열로부터 특성 벡터를 구축하는 알고리즘과 특성들을 정규화하고 정량화하는 방법을 제공.

- 단어 빈도-역문서 빈도 (TF-IDF): 텍스트 마이닝이나 검색기등에서 사용되는 문서에서 특성 벡터를 생성하는 간단한 방식
  - TF: 하나의 문서에서 단어가 등장하는 빈도
  - IDF: 전체 문서군에서 얼마나 자주 단어가 등장하는지에 대한 빈도.

- TF-IDF를 위한 MLlib 제공 알고리즘 (mllib.feature)
  - HashingTF: 문서로부터 주어진 길이의 TF를 계산 (각 단어의 해쉬값을 주어진 벡터크기로 나누고,나머지값에 해당되는 곳에 매핑)

- IDF: 전체 문서의 수를 해당단어가 포함된 문서들의 수로 나눈 값에 로그를 취한 값.

- TF-IDF : TF와 IDF의 곱한 값. (주어진 문서에서 TF가 크고, 전체문서의 TF가 작을수록 커짐, 따라서 공통적으로 등장하는 단어들은 수치가 낮아져 걸러지게 됨.)

알고리즘 (cont'd)

- 정량화
  대부분의 머신 러닝 알고리즘은 특징 벡터에서 각 요소의 척도를 고려함.따라서 특성들이 정량화(Scaling)되었을 때 잘 동작하므로, 이를 위해 요소들이 동일하게 정량화 할수 있도록 ML에서 기능을 제공.
  - mllib.feature의 StandardScaler (평균 0, 표준편차 1로 정량화)

- 정규화
  어떤 상황에서는 벡터를 길이 1로 정규화하는 것이 입력 데이터를 준비하는 데 도움이 된다.
  - Normalizer 클래스가 이를 담당하며 Normalizer().transform(rdd)를 호출하여 사용.

통계

- 기본적인 통계는 간단한 탐색이든 머신 러닝을 위한 데이터 이해든 데이터 분석에서 중요한 부분으로
  MLlib은 mllib.stat.Statistics 클래스의 메소드들을 통해 RDD에서 직접 사용 가능한 통계 함수들을 제공.

- Statistics.colStats(rdd) :
  - 벡터의 RDD의 통계적인 요약을 계산하며, 최소값, 최대값, 평균 각 칼럼의 분산등을 벡터들의 집합에 저장한다.
    한번에 다양한 통계량를 구하기 위해 쓰임.

- Statistics.corr(rdd, method) :
  - 벡터 RDD의 칼럼들 사이에 상관관계 행렬을 계산하며, 피어슨이나 스피어만 상관관계 중 하나를 사용.

- Statistics.chiSqTest(rdd):
  - 모든 변수(LabeledPoint 형태로 추출된 Feature) 에 대한 피어슨 독립성 검증 제공
  - 결과로 유의확률값, test 통계량, 각 변수의 자유도를 배열 형태로 제공.

- 기타: mean, stdev, sum등의 기초 통계함수 지(LabeledPoint 형태로 추출된 Feature) 원.

분류와 회귀

- 분류와 회귀는 훈련데이터를 사용한 특징 객체들로부터 변수를 예측하는 supervised learning의 일반적인 두가지 형태.
  MLlib의 LabeledPoint(label, feature로 구성) 클래스를 사용하며, 이는 mllib.regression 패키지에 위치함.
  - 분류 : 예측하는 변수의 형태가 이산적.
  - 회귀 : 예측하는 변수의 형태가 연속적.

- 선형 회귀
  - 선형 회귀는 회귀를 위한 가장 일반적인 알고리즘. 출력 변수를 특성들의 선형 조합 형태로 예측.
  - 선형 회귀는 L1(Lasso), L2(ridge regression)으로 알려 있다.
  - 선형 회귀 알고리즘으로 mllib.regression 내에서 LinearRegressionwithSGD, LassowithSGD, RidgeRegresionwithSGD 제공.
  - SGD(Stochastic Gradient Descent 확률적 경사 강하).

분류와 회귀(cont'd)

- 알고리즘 최적화 인자
  - numIterations : 실행할 반복 횟수(기본값 100)
  - stepSize: 경사 강하를 위한 단계의 크기(기본값 10)
  - intercept : 데이터에 인터셉트나 바이어스 특성을 추가할지, 다시 말해 항상 값이 1인 다른 특성이 있는지
  - regParam : 라쏘나 능형 회귀를 위한 정규화 인자값.

- 각언어별 호출 방법:

  ex 11-10)

```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.regression import LinearRegressionWithSGD
points = # (LabeledPoint RDD 생성)
model = LinearRegressionWithSGD.train(points, iterations =200, intercept = True)
print "weights: %s, intercept %s" % (model.weights, model.intercepts)
```

분류와 회귀(cont'd)

- 로지스틱 회귀
  - 로지스틱 회귀는 양성과 음성 예제들 중에서 선형으로 구분할 수 있는 기준을 찾아내는 binary classification method
  - MLlib에서 이는 라벨이 0이나 1인 LabeledPoint 형태의 데이터를 다루고, 신규 데이터들을 예측할 수 있는 LogisticeRegressionModel을 되돌려준다.

- 로지스틱 회귀 알고리즘은 앞에서 다룬 선형회귀와 매우 유사한 알고리즘을 갖고 있다.
  - 차이점 : SGD 외에도 LBFGS 사용 가능.
  - 위치 : mllib.classification.LogisticRegressionWithLBFGS
    mllib.classification.LogisticRegressionWithSGD

- 이 알고리즘들로부터 나오는 LogisticRegressionModel은 개별 데이터포인트에 대해 0과 1 사이의 점수를 계산해주는 로지스틱 함수를 제공한다.

분류와 회귀(cont'd)

- SVD (Support Vector Machine)
  - 0과 1사이의 점수(기대값)를 계산해주는 또 다른 binary classification method.
  - 선형 회귀나 로지스틱 회귀와 유사한 인자를 받아들이는 SVMWithSGD 클래스를 통해 사용 가능.
  - SVMModel은 LogisticRegressionModel과 유사한 SVMModel를 돌려줌.

분류와 회귀(cont'd)

- Naive Bayes
  - 나이브베이즈는 선형함수의 Feature에 기초해 각각의 점이 각 분류에 얼마나 잘 들어가 있는지 특정하는 다중 분류 알고리즘. (주로 TF-IDF 특성에 대한 문서분류에 쓰임).
  - MLlib는 입력 변수(feature)로 음수가 아닌 빈도들을 사용하는 다항 나이브 베이즈를 제공.
  - 위치: mllib.classification.NaiveBayes

알고리즘 결과물인 NaiveBayesModel은 각 점의 최적의 분류를 찾는 predict() 제공. 더불어 훈련이 끝난 모델의 결과값을 제공.: 각 feature와 분류확률에 대한 행렬인 세타와 분류들로 구성된 C차원 벡터pi

분류와 회귀(cont'd)

- Decision trees and random forests
  의사결정트리는 분류와 회귀 양쪽에 쓸 수 있는 유연한 모델로, 노드들을 갖고 있는 트리로 표현된다. 의사 트리는 모델을 검증하기 쉽고 분류 가능한 특징이나 연속적인 특징을 모두 지원하므로 매력적.

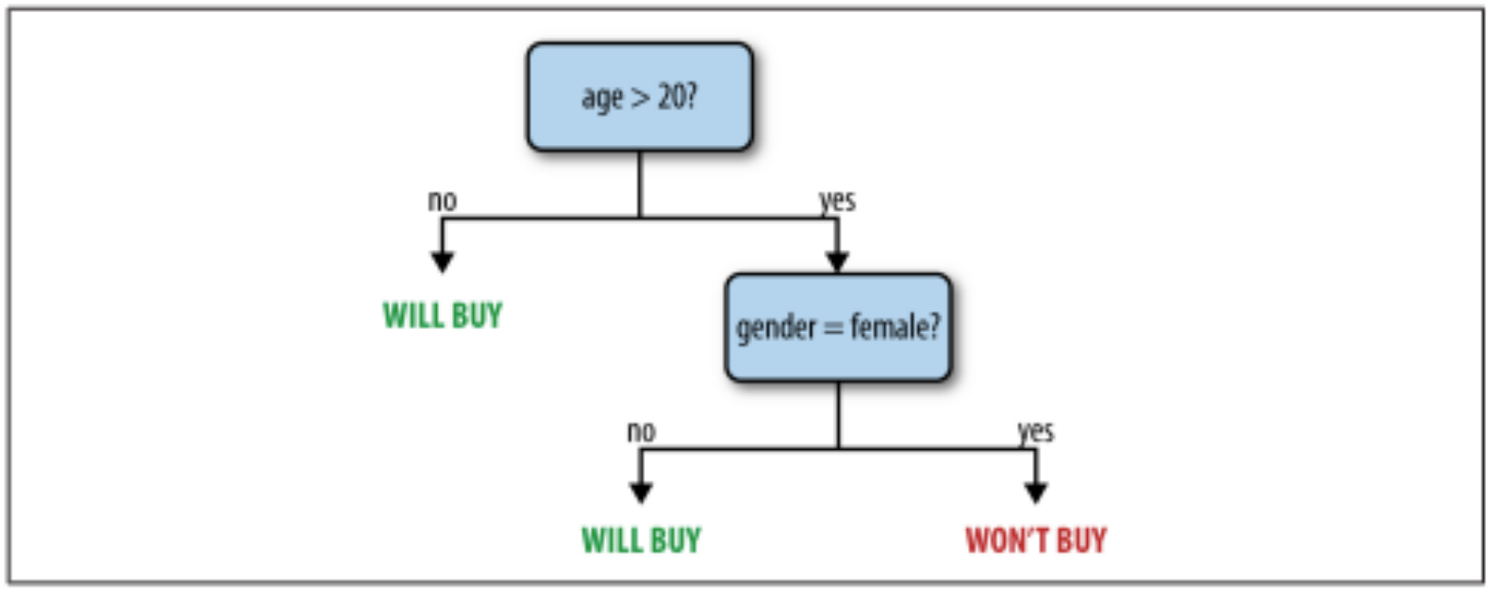

Figure 11-2. An example decision tree predicting whether a user might buy a product

분류와 회귀(cont'd)

- MLlib에서는 mllib.tree.DecisionTree 클래스에서 trainclassifier()나 trainRegresor()메소드를 써서 트리를 훈련시킬 수 있다. 다른 알고리즘과 달리 자바와 스칼라 API도 DecisionTree 객체의 세터를 쓰지 않고, static 메소드를 쓴다.

- 훈련 메소드들은 다음 인자들을 받아들인다.
  - data : LabeledPoint의 RDD
  - numclasses: 사용할 분류의 개수
  - impurity : 노드의 불순함의 척도, 분류를 위해서는 gini나 entropy가 되어야 하며, 회귀를 위해서는 variance가 된다.
  - maxDepth : 트리의 최대 깊이(기본값 9)
  - maxBins: 각 노드를 만들때 데이터를 분리해 담을 저장소의 개수(추천값 32)
  - categoricalFeatureInfo: 어떤 특징이 분류 가능하고 그것들이 얼마나 많은 개수의 카테고리를 갖고 있는지를 갖고 있는 형태의 정보.

- train메소드는 DecisionTreeModel을 되돌려준다. 이 모델은 새로운 특징 벡터의 값들을 예측하거나 predict()를 써서 벡터의 RDD를 예측하거나 toDebugString()을 써서 트리를 출력하는 데에 사용될 수 있다.

- 이 모델 객체는 직렬화 할 수 있으므로 자바 직렬화를 써서 저장하고, 다른 프로그램에서 불러서 사용할 수 있다

분류와 회귀(cont'd)

- 다수의 트리를 구축할 수 있는 RandomForest 클래스를 실험적으로 추가했다. 이는 RandomForest.trainclassfier와 trainRegressor를 쓸 수 있다.

- 리스트로 나열하는 트리별 인자들 말고도 RandomForest는 다음 인자들을 받아들인다.
  - numTrees: 얼마나 많은 트리를 구축할지 결정.
    numTrees의 증가는 훈련 데이터의 과적합 가능성을 줄여줌..
  - featureSubsetStrategy: 각 노드에서 분리를 위해 고려할 특징의 개수를 결정한다.
  - seed: 난수 발생을 위한 seed

- 랜덤 포레스트는 여러 개의 트리를 갖고 있는 weakHypothesisWeights로 계량되어서 weakHyptheses 필드에 저장된다.
  WeightedEnsembleModel을 리턴하며, vector나 RDD를 predict() 할 수 있다.

클러스터링

- 클러스터링은 높은 유사도를 가지는 클러스터로 그룹화를 하는 자율 학습(unsupervised learning Task).

- 데이터에 라벨을 필요로 하는 이전의 supervised task와는 다르게 클러스터링은 라벨없는 데이터를 이해하기 위해 사용될 수 있다.

- 대개 데이터 검사나 예외적인 것들을 발견하기 위해 주로 사용됨.

클러스터링(cont'd)

- MLlib은 클러스터링을 위한 인기있는 알고리즘 K-means와 함께 병렬 환경에서 더 나은 초기화를 제공하는 변형 알고리즘 K-means||도 제공한다. K-means||은 단일 노드 세팅에 쓰이는 K-평균++의 초기화 절차와 유사하다.

- K-평균에서 가장 중요한 인자는 생성할 클러스터의 개수 K이다. 실제로는 정말로 클러스터가 몇 개나 필요 할지는 알기 힘드므로 실용적인 방법은 여러가지 값의 K로 평균적인 클러스터간의 거리가 더이상 줄어들이지 않을때까지 시도해보는것이다.

- 아래 인자를 받아들인다.
  - initializationMode : 클러스터의 중심을 초기화하는 방법 k-means|| or random 중 하나만 가능.
  - maxIterations : 실행할 반복 횟수의 최댓값(기본값 100)
  - runs :알고리즘 동시 실행 개수. MLlib의 K-평균은 여러 개의 시작 위치에서 동시에 실행하여 최적의 결과를 찾는 기능을 지원하는데, 이는 전반적으로 더 나은 모델을 찾기에 좋은 방법이다.

- 다른 알고리즘들처럼 K-평균은 mllib.clustering.KMeans 객체를 만들거나 KMeans.train을 호출해서 쓸 수 있다.

- predict()는 언제나 점에 가장 가까운 클러스터 중심을 리턴해 준다는 점을 기억해야 한다.

collaborative Filtering and Recommendation

- 협업 필터링은 다양한 상품에 대한 사용자의 평점과 상호 작용이 혼합하여 새로운 상품을 추천하는 추천시스템을 위한 기술이다. 이는 사용자와 상품의 상호작용(반응) 리스트만 있으면 되므로 매력적이다.

- 이 상호작용(반응)이란 "표면적인" 것 일수도 있고 "추상적인" 것일 수도 있다. 오직 이런 반응들에만 기초하여 협업 필터링은 어느 상품들이 서로 유사성이 있는지, 어떤 사용자들끼리 유사성이 있는지, 새로운 추천이 가능한지등을 학습한다.

- MLlib API는 "사용자"와 "상품"에 대해서만 이야기하지만, 협업 필터링을 이용하여 어떤 SNS를 사용자에게 추천할지,올린 글에 어울리는 태그는 어떤 것인지, 라디오 방송에 추가할만한 음악은 어떤 것인지 등 다른 애플리케이션에 적용할 수 도 있다.

collaborative Filtering and Recommendation

- 교대 최소 제곱법 : MLlib은 클러스터들에서 정량화가 우수한 인기 있는 협업 필터링 알고리즘인 교대 최소 제곱법(ALS, Alternating Least Squares)의 구현을 포함하고 있다.
  이는 mllib.recommendation.ALS 클래스에 있다.

- ALS는 사용자 벡터와 상품벡터의 곱이 점수에 근접하도록 각 사용자와 상품에 대한 특징벡터를 정의하는 것으로 동작한다.

- ALS는 다음의 인자들을 받아들인다.
  - rank: 사용할 특징 벡터의 크기, 더 큰 벡터는 더 나은 모델을 만들어 낼 수 있지만 계산 비용이 더 커진다.
  - iterations : 실행할 반복 횟수
  - lambda : 정규화 인자
  - alpha : 암묵적인 ALS에서 신뢰도를 계산하는데 사용되는 상수
  - numUserBlocks, numProductBlocks : 사용자와 상품 데이터를 구분할 블록의 개수,
    병렬화를 제어한다. -1을 넘겨주면 MLlib이 자동적으로 결정하게 되며 이것이 기본동작 방식이다.

차원 축소

- 주성분 분석(PCA) : 차원 축소의 주된 기법은 주성분 분석으로, 저차원 표현에서 데이터의 분산이 가장 커지도록 저차원 공간에 대한 매핑을 시도하고 유용성이 떨어지는 차원은 무시한다. 매핑 계산을 위해서는 정규화된 상관관계 행렬이 만들어지고 특이 벡터들과 행렬의 값을 사용된다.
  가장 큰 특이값들과 연관된 특이 벡터들은 원본 데이터의 분산의 큰 부분을 재구축하는 데에 사용된다.

- Example 11-13 스칼라에서 PCA 호출

```
import org.apache.spark.mllib.linalg.Matrix

import org.apache.spark.mllib.distributed.RowMatrix

val points: RDD[Vector] = //...

val mat: RowMatrix = new RowMatrix(points)

val pc: Matrix = mat.computerPrincipalcomponents(2)

//점들을 저차원 공간에 곱한다.

val projected = mat.multiply(pc).rows

//k-평균 모델을 곱한 2차원 데이터에서 학습시킨다.

val model = kMeans.train(projected, 10)
```

차원 축소(cont'd)

- 특이값 분해(SVD, Singular value Decomposition)

- MLlib은 또한 저수준의 특이값 분해 기능을 제공한다.
  SVD는 m X n 행렬 A를 세 개의 행렬 A = U sum(V) T
  - U는 직교 행렬이며 열들이 좌측 특이 벡터로 불린다.
  - sum은 내림차순으로 대각 행렬값을 가지며 각 값은 특이값으로 불린다.
  - V는 직교행렬이며 열들이 우측 특이 벡터로 불린다.

차원 축소(cont'd)

- 특이값 분해(SVD, Singular value Decomposition)

- MLlib은 또한 singular value decomposition (SVD) 기능을 제공.
  The SVD factorizes an m x n matrix A into three matrices $A \approx U \sum V$
  - U is an orthonormal matrix, whose columns are called left singular vectors.
  - $\sum$ is a diagonal matrix with nonnegative diagonals in descending order, whose. diagonals are called singular values.
  - V is an orthonormal matrix, whose columns are called right singular vectors

- 큰 단위의 행렬에서는 complete factorization이 필요 없지만 최상위 singular values 들과 그와 관련된 singular vectors는 필요하다.
  이를 통해 저장 공간을 절약하고, 노이즈를 줄여 행렬의 low-rank 구조를 복원하게 해준다.
  만약 상위 k개의 singular value들을 가진다면, 결과 행렬의 차원은 U : m x k, $\sum$: k x k, V : n x k 가 된다.

모델 평가

- 머신러닝 작업에 쓰이는 알고리즘이 어떤 것이든 모델 평가는 머신 러닝 파이프라인의 시작에서 끝까지 중요한 역할을 차지한다.

- 많은 학습 작업들은 사용하는 모델들이 다르거나, 심지어 동일 알고리즘내의 인자 설정에 따라 다른 결과를 도출하는 도전을 받게 된다.

- 덧붙혀 데이터에 대한 모델의 과적합의 위험도 항상 존재하므로 데이터 학습보다는 먼저 모델을 테스트함으로서 최적의 평가를 해볼 수 있다.

- 스파크의 파이프라인 API를 써서 ML알고리즘들과 평가지표를 결정하고, 자동적으로 시스템이 인자들을 찾아 상호검증을 통해 최적의 모델을 찾을 수 있다.

팁과 성능 고려 사항:

Preparing Features

- 머신 러닝에 대한 소개들은 사용하는 알고리즘에 보통 무게를 두지만, 실제로 각 알고리즘은 사용자가 입력하는 특성들이 잘 선별되었느냐에 따라 우수함이 결정된다는 것이 중요하다.

- 대규모 머신 러닝 작업에 종사하는 많은 이들이 특성을 준비하는 것이 가장 중요한 단계
  정보성이 큰 특성들을 추가하고 가능한 특성들을 적절하게 벡터로 변환하는 것등은 모두 결과물의 향상에 큰 보탬이 된다.

- 입력 특성들을 계량화하라. 특성들에 동일한 수준으로 가중치를 부여하기 위해 정량화에서 쓴 것처럼 StandardSclaer등을 활용하자
- 문자열을 올바르게 특성화한다. NLTK 같은 외부 라이브러리를 써서 단어를 분리하고 TF-IDF를 위해 대표성 있는 문서군에 대해 IDF를 활용하자
- 각 분류에 라벨을 바르게 붙인다. MLlib은 분류 라벨을 0에서 C-1까지 쓰며 여기서 C는 분류 개수이다.

알고리즘 설정

- MLlib의 많은 알고리즘은 정규화를 쓰는 것이 예측 확률면에서 더 잘 동작한다. 또한 대부분의 SGD 기반 알고리즘은 좋은 결과를 얻기 위해 100번 가량의 반복을 수행해야 한다.

- MLlib은 일반적으로 유용한 기본값을 제공하도록 설정되어 있지만 정확도가 향상되는지 확인하기 위해 기본값보다 더 반복 횟수를 올리는 것 또한 시도해 보아야 한다.

- 학습동안에 쓰일 테스트 데이터에 대해 이런 설정값의 변경을 확인해 보아야 한다.

재사용을 위한 RDD 캐싱

- MLlib의 대부분 알고리즘은 데이터를 여러 번 처리하는 반복적인 알고리즘이다. 그러므로 MLlib에 데이터를 넘기기 전에 입력이 되는 데이터셋트에 대해 cache()를 사용하는 것은 중요하다. 심지어 메모리에 다 들어가지 못할 만큼 크더라도 persit를 쓰도록 한다.

- 파이썬에서는 MLlib은 파이썬에서 데이터가 넘어갈 때 자바쪽에 자동적으로 캐시하므로 프로그램 내에서 재사용하는 것이 아니라면 따로 파이썬 코드안에서 캐시할 필요는 없다. 하지만 스칼라와 자바에서는 개발자에게 캐시여부가 달려있다.

Recognizing Sparsity

- Feature vector의 값이 대부분 0이고 저밀도 포맷으로 저장된다면 큰 데이터 셋에 대해서 처리 시간과 공간을 매우 아낄수 있다.

- 공간 측면에서 저밀도 포맷은 고밀도 포맷에 비해 최대 2/3이 0이 아닌 값이라면 훨씬 적은 공간을 차지한다.

- 처리비용측면에서 저밀도 벡터는 최대 10%의 값이 0이 아닌 경우 일반적으로 적은 비용이 든다.

- 하지만 저밀도 벡터를 쓰느냐 아니냐에 따라 메모리에 캐시할 수 있느냐가의 여부가 결정된다면, 심지어 데이터의 밀도가 높을지라도 저밀도 벡터의 사용을 고려해 보아야 한다.

병렬화 수준

- 대부분의 알고리즘에 대해서 입력 RDD의 파티션 개수는 최소 전체 코어 개수만큼은 되어야 병렬화를 최대로 활용할 수 있다.

- 스파크는 기본적으로 파일의 블록마다 하나의 파티션을 만들며 블록의 크기는 대개 64MB이다.

- 파티션 개수를 변경하려면 최소 파티션 개수를 Sparkcontext.textFile()에 넘겨줄 수 있다.

- 혹은 RDD에서 repartition(numPartitions)를 호출하여 numPartitions 개수만큼 파티셔닝하도록 할 수 있다.

파이프라인 API

- MLlib은 머신 러닝을 위해 파이프라인 개념에 기반한 새로운 고수준 API를 추가하였다.

- 이들 파이프라인은 데이터셋을 변환하는 알고리즘들의 모음이다.

- 파이프라인의 각 단계는 인자를 가지며, 평가 지표들을 달리 해 가며 평가하는 격자 검색을 통해 자동적으로 최적의 인자구성을 찾아낼 수 있다.

- 파이프라인은 전체 데이터 셋에 일관된 표현 방식을 사용하는데 이는 SchemaRDD 이다.

- SchemaRDD는 여러 개의 이름 있는 컬럼을 가지고 있어서 데이터의 다른 필드들을 참조하기 쉽게 해준다. 이들 파이프라인은 R의 데이터프레임과 유사하다..