

다음 세대 스파크를 이용한 머신 러닝

XGBoost, LightGBM, Spark NLP,
Keras를 사용한 분산 딥 러닝 등

—

부치 피프스

Apress®

차세대 기계 스파크로 학습

XGBoost, LightGBM, Spark 포함
NLP, Keras를 사용한 분산 딥 러닝
등

부치 피프스

Apress®

Spark를 사용한 차세대 기계 학습: XGBoost, LightGBM,
Spark NLP, Keras를 사용한 분산 딥 러닝 등

부치 피프스
미국 캘리포니아 카슨

ISBN-13(pbk): 978-1-4842-5668-8
<https://doi.org/10.1007/978-1-4842-5669-5>

ISBN-13(전자): 978-1-4842-5669-5

Copyright © 2020 by Butch Quinto

이 작업은 저작권의 보호를 받습니다. 자료의 전체 또는 일부와 관련된 모든 권리, 특히 번역, 재인쇄, 삽화의 재사용, 암송, 방송, 마이크로필름 또는 기타 물리적 방식의 복제, 전송 또는 정보 저장에 대한 권리는 게시자에게 있습니다. 및 검색, 전자적 적응, 컴퓨터 소프트웨어, 또는 현재 알려져 있거나 향후 개발될 유사하거나 유사하지 않은 방법론에 의한 것입니다.

상표명, 로고 및 이미지가 이 책에 나타날 수 있습니다. 상표명, 로고 또는 이미지가 나타날 때마다 상표 기호를 사용하는 대신 상표를 침해할 의도 없이 상표 소유자의 이익을 위해 편집 방식으로만 이름, 로고 및 이미지를 사용합니다.

이 출판물에서 상호, 상표, 서비스 마크 및 이와 유사한 용어를 사용하는 것은 그러한 것으로 식별되지 않더라도 소유권의 적용을 받는지 여부에 대한 의견의 표현으로 간주되어서는 안 됩니다.

이 책의 조언과 정보는 출판일 현재 사실이고 정확한 것으로 여겨지지만 저자, 편집자 또는 출판사는 발생할 수 있는 오류나 누락에 대해 법적 책임을 지지 않습니다. 발행인은 여기에 포함된 자료와 관련하여 명시적이든 묵시적이든 어떠한 보증도 하지 않습니다.

Apress Media LLC 전무이사: Welmoed Spahr

인수 편집자: Susan McDermott

개발 편집자: Laura Berendson

코디네이터: 리타 페르난도

eStudioCalamar가 디자인한 표지

Freepik에서 디자인한 표지 이미지(www.freepik.com)

Springer Science+Business Media New York, 1 New York Plaza, New York, NY 10004에서 전 세계 서적 거래에 배포.
전화 1-800-SPRINGER, 팩스 (201) 348-4505, 이메일 주문-ny@springer-sbm.com을 방문하거나
www.springeronline.com을 방문하십시오. Apress Media, LLC는 캘리포니아 LLC이며 유일한 구성원(소유자)은 Springer Science + Business Media Finance Inc(SSBM Finance Inc)입니다. SSBM Finance Inc는 델라웨어주입니다.
법인.

번역에 대한 정보는 right@apress.com으로 이메일을 보내거나 <http://www.apress.com/>을 방문하십시오.
권리 허가.

Apress 타이틀은 학술, 기업 또는 판촉용으로 대량 구매할 수 있습니다. eBook 버전 및 라이선스도 대부분의 책에서 사용할 수 있습니다. 자세한 내용은 인쇄 및 전자책 대량 판매 웹 페이지(<http://www.apress.com/bulk-sales>)를 참조하십시오.

이 책에서 저자가 참조한 소스 코드 또는 기타 보충 자료는 www.apress.com/9781484256688에 있는 책의 제품 페이지를 통해 GitHub 독자에게 제공됩니다. 자세한 내용은 <http://www.apress.com/source-code>를 참조하십시오.

무산 종이에 인쇄

이 책은 제 아내 Aileen에게 바칩니다. 나의 자녀
들, 매튜, 티모시, 올리비아; 내 자매, Kat와 Kristel; 그
리고 나의 부모님, Edgar와 Cynthia.

목차

저자 소개	xiii
기술 검토자 소개	xv
감사의 말
서론	xix
1 장 : 머신 러닝 소개.....	1
AI 및 머신 러닝 사용 사례	
소개	3
교통	3
금융서비스	3
의료 및 생명공학	4
제조업	4
정부	4
머신 러닝 및 데이터	5
관찰	6
머신 러닝 방법	6
지도학습	6
비지도 학습	12
준지도 학습	15
강화학습	15
딥 러닝	15
신경망	16
컨볼루션 신경망	16

목차

피처 엔지니어링	16
기능 선택	17
기능 중요도	18
특징 추출	19
특징 구성	19
모형 평가	20
정확도	20
정밀도	21
리콜	21
F1 측정	21
AUROC(수신기 작동 특성 아래 영역)	21
과대적합과 과소적합	22
모델 선택	22
요약	23
참고문헌	23
2 장 : Spark 및 Spark MLlib 소개	29
개요	29
클러스터 관리자	30
아키텍처	30
Spark 응용 프로그램 실행	32
클러스터 모드	32
고객 패션	32
스파크쉘 소개	32
스파크세션	33
복원력 있는 분산 데이터 세트(RDD)	34
Spark SQL, Dataset 및 DataFrames API	43
Spark 데이터 소스	44
CSV	45
XML	45
JSON	46
관계형 및 MPP 데이터베이스	47

목차

쪽모이 세공	50
HBase	50
아마존 S3	56
솔러	56
マイクロ소프트 엑셀	57
보안 FTP	58
Spark MLlib 소개	59
Spark MLlib 알고리즘	59
ML 파이프라인	60
파이프라인	61
변압기	61
에스티메이터	61
ParamGridBuilder 페도라 페도라	61
교차검증기	62
평가자	62
특징 추출, 변환 및 선택	62
StringIndexer	62
토큰나이저	64
벡터어셈블러	65
스탠다드스케일러	66
StopWordsRemover	67 n-
gram	68
OneHotEncoderEstimator	68
SQLTransformer	70
Term Frequency–Inverse Document Frequency(TF-IDF)	71
주성분분석(PCA)	7
ChiSqSelector	74
상관관계	75
평가지표	77
AUROC(수신기 작동 특성 아래 영역)	77
F1 측정	78
RMSE(Root Mean Squared Error)	78

목차

모델 지속성	
Spark MLlib 예제	79
그래프 처리	83
Spark MLlib를 넘어서: 타사 기계 학습 통합.....	84
Alluxio로 Spark 및 Spark MLlib 최적화	84
아키텍처	84
왜 Alluxio를 사용합니까?	86
상 86 빅데이터 처리 성능 및 확장성 대폭 향 유할 수 있음 87	
애플리케이션 종료 또는 실패 시 고가용성 및 지속성 제공.....	89
전체 메모리 사용을 최적화하고 가비지 수집을 최소화합니다.	
하드웨어 요구 사항 감소	92
Apache Spark 및 Alluxio	93
요약	93
참고문헌	94
3 장 : 지도 학습	97
구분	97
2진 분류	97
Spark MLlib 분류 알고리즘	98
제3자 분류 및 회귀 알고리즘	104
로지스틱 회귀를 사용한 다중 클래스 분류	104
랜덤 포레스트를 이용한 이탈 예측	116
XGBoost4J-Spark를 사용한 극도의 그래디언트 부스팅	133
LightGBM: Microsoft의 빠른 그래디언트 부스팅	144
Naive Bayes를 사용한 감정 분석	156
회귀	164
단순 선형 회귀	164
XGBoost4J-Spark를 사용한 다중 회귀	168
LightGBM을 사용한 다중 회귀	176
요약	182
참고문헌	183

목차

4 장 : 비지도 학습	189
K-평균을 사용한 군집화	189
예	191
LDA(Latent Dirichlet Allocation)를 사용한 주제 모델링	199
Spark용 Stanford CoreNLP	200
John Snow Labs의 Spark NLP	202
예	208
Isolation Forest를 통한 이상 탐지	224
매개변수	227
예	228
주성분 분석을 통한 차원 축소 ...	232
예	
요약	
참고문헌	242
5 장 : 권장 사항	245
추천 엔진의 종류 ...	246
교대 최소 제곱을 사용한 협업 필터링 246	
매개변수	248
예	248
FP-Growth로 시장 바구니 분석	255
예	
콘텐츠 기반 필터링	265
요약	266
참고문헌	266
6 장 : 그래프 분석	269
그래프 소개	269
무방향 그래프	269
그래프 분석 활용 사례	271
사기 탐지 및 자금세탁방지(AML)	272

목차

GraphX 소개	273
그래프	273
VertexRDD	274
모서리	274
에지RDD	274
에지트리플렛	274
EdgeContext	274
GraphX 예	274
그래프 알고리즘	278
그래프 프레임	281
요약	
참고문헌	286
7 장 : 딥 러닝	289
신경망	291
신경망의 간략한 역사	292
컨볼루션 신경망	294
컨볼루션 신경망 아키텍처	296
특징 탐지 계층	296
컨벌루션 레이어	296
ReLU(Rectified Linear Unit) 활성화 함수	298
풀링 계층	298
분류 계층	299
딥 러닝 프레임워크	299
텐서플로우	299
Theano	300
파이토치	300
딥러닝4J	300
CNTK	300
하드	300
Keras를 사용한 딥 러닝	301

목차

Spark를 사용한 분산 딥 러닝	313
모델 병렬도 vs. 데이터 병렬도	314
Spark용 분산 딥 러닝 프레임워크 ...	315
Elephas: Keras 및 Spark를 사용한 분산 딥 러닝.....	316
Keras 및 Spark와 함께 Elephas를 사용하는 MNIST로 필기 숫자 인식	317
분산 하드(Dist-Hard)	328
Keras 및 Spark와 함께 Dist-Keras를 사용하는 MNIST로 필기 숫자 인식	328
개와 고양이 이미지 분류	335
요약	344
참고문헌	345
지수	349

저자 소개

Butch Quinto는 방위, 산업 및 운송 산업을 위한 첨단 솔루션을 개발하는 인공 지능 회사인 Intelvi AI의 설립자이자 최고 AI 책임자입니다. Butch는 최고 AI 책임자로서 전략, 혁신, 연구 및 개발을 이끌고 있습니다. 이전에는 선도적인 기술 회사의 인공 지능 이사이자 AI 스타트업의 최고 데이터 책임자였습니다. Deloitte의 분석 이사로서 그는 여러 엔터프라이즈급 AI 및 IoT 솔루션의 개발과 전략, 비즈니스 개발 및 벤처 캐피탈 실사를 주도했습니다. Butch는 은행 및 금융, 통신, 정부, 유틸리티, 운송, 전자 상거래, 소매, 제조 및 생물 정보학을 포함한 여러 산업 분야에서 다양한 기술 및 리더십 역할에서 20년 이상의 경험을 가지고 있습니다. 그는 또한 차세대 빅 데이터 (Apress, 2018)의 저자이자 인공 지능 진흥 협회 및 미국 과학 진흥 협회의 회원입니다.

기술 검토자 정보

Irfan Elahi는 데이터 과학 및 기계 학습 분야에서 다년간의 다분야 경험을 보유하고 있습니다. 그는 컨설팅 회사, 자신의 신생 기업, 학계 연구실과 같은 다양한 수직 분야에서 일했습니다. 수년 동안 그는 기업이

데이터 자산에서 엄청난 가치를 얻을 수 있습니다.

감사의 말

Apress의 모든 사람, 특히 Rita Fernando Kim, Laura C. Berendson, Susan McDermott에게 이 책을 출판하는 데 도움과 지원을 해주셔서 감사합니다. Apress 팀과 함께 일할 수 있어서 즐거웠습니다. 여러 사람들이 이 책에 직간접적으로 기여했습니다. Matei Zaharia, Joeri Hermans, Max Pumperla, Fangzhou Yang, Alejandro Correa Bahnsen, Zygmunt Zawadzki 및 Irfan Elahi에게 감사드립니다. Databricks와 전체 Apache Spark, ML 및 AI 커뮤니티 덕분입니다.

격려와 지원에 대해 Kat, Kristel, Edgar 및 Cynthia에게 특별한 감사를 전합니다. 마지막으로 제 아내 Aileen과 아이들인 Matthew, Timothy에게 감사드립니다.
그리고 올리비아.

소개

이 책은 Spark 및 Spark MLlib에 대한 접근 가능한 소개를 제공합니다. 그러나 이것은 아직 표준 Spark MLlib 알고리즘에 대한 또 다른 책이 아닙니다. 이 책은 표준 Spark MLlib 라이브러리에서 사용할 수 있는 것 이상의 강력한 타사 기계 학습 알고리즘 및 라이브러리에 중점을 둡니다. 제가 다루는 고급 주제에는 XGBoost4J-Spark, LightGBM on Spark, Isolation Forest, Spark NLP, Stanford CoreNLP, Alluxio, Keras를 사용한 분산 딥 러닝 및 Elephas 및 분산 Keras를 사용한 Spark 등이 있습니다.

Spark 및 Spark MLlib에 대한 사전 경험이 없다고 가정합니다. 그러나 이 책의 예제를 따르려면 기계 학습, Scala 및 Python에 대한 약간의 지식이 있으면 도움이 됩니다. 이 책을 최대한 활용하려면 예제를 통해 작업하고 코드 샘플을 실행하는 것이 좋습니다. 1 장은 기계 학습에 대한 간략한 소개로 시작합니다. 2 장에서는 Spark 및 Spark MLlib를 소개합니다. 더 고급 주제로 바로 이동하려면 관심 있는 장으로 바로 이동하십시오. 이 책은 실무자를 위한 책이다. 나는 이론에 집중하기보다는 실습 접근에 초점을 맞추면서 가능한 한 간단하고 실용적인 책을 유지하려고 노력했습니다(이 책에도 많은 내용이 있음에도 불구하고). 기계 학습에 대한 더 철저한 소개가 필요하면 Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani의 통계 학습 소개(Springer, 2017) 및 The Elements of Statistical Learning by Trevor Hastie, Robert Tibshirani 및 Jerome Friedman(Springer, 2016). Spark MLlib에 대한 자세한 내용은 온라인에서 Apache Spark의 MLlib(기계 학습 라이브러리) 가이드를 참조 하세요. 딥러닝에 대한 철저한 치료를 위해서는 Ian Goodfellow, Yoshua Bengio, Aaron Courville의 Deep Learning을 추천합니다(MIT Press, 2016).

1장

기계 소개

학습

나는 당신에게 일반적인 주장을 할 수 있습니다. 그러나 진실은 발견의 전망이 너무 달콤하다는 것입니다.

—제프리 힌토니

머신 러닝(ML)은 인공 지능의 하위 분야이며 지능적인 기계를 만드는 과학 및 공학입니다.ⁱⁱ 인공 지능의 선구자 중 한 명인 Arthur Samuel은 머신 러닝을 "컴퓨터에 학습 능력을 부여하는 연구 분야"라고 정의했습니다. 명시적으로 프로그래밍되어 있습니다."ⁱⁱⁱ 그림 1-1은 인공 지능, 머신 러닝 및 딥 러닝 간의 관계를 보여줍니다. 인공 지능(AI)은 다른 분야를 포괄합니다. 즉, 모든 머신 러닝이 AI이지만 모든 AI가 머신 러닝인 것은 아닙니다. 인공 지능의 또 다른 분야인 상징적 인공 지능은 20세기 대부분의 기간 동안 지배적인 인공 지능 연구 패러다임이었습니다.^{iv} 상징적 인공 지능 구현은 본질적으로 if-then 문을 사용하는 규칙 엔진인 지식 그래프 또는 전문가 시스템이라고 합니다. 연역적 추론을 사용하여 논리적인 결론을 도출합니다. 상상할 수 있듯이 상징적 AI에는 몇 가지 주요 제한 사항이 있습니다. 그 중 가장 중요한 것은 규칙 엔진에서 정의된 규칙을 수정하는 복잡성입니다. 더 많은 규칙을 추가하면 규칙 엔진의 지식이 증가하지만 기존 지식은 변경할 수 없습니다.^v 반면에 기계 학습 모델은 더 유연합니다. 그들은 새로운 것을 배우거나 기존 지식을 수정하기 위해 새로운 데이터에 대해 재교육을 받을 수 있습니다. Symbolic AI에는 상당한 인간 개입도 포함됩니다. 그것은 인간의 지식에 의존하고 인간이 규칙 엔진에서 규칙을 하드 코딩하도록 요구합니다. 반면에 머신 러닝은 입력 데이터에서 패턴을 학습하고 인식하여 원하는 출력을 생성하는 보다 역동적입니다.

1장 기계 학습 소개

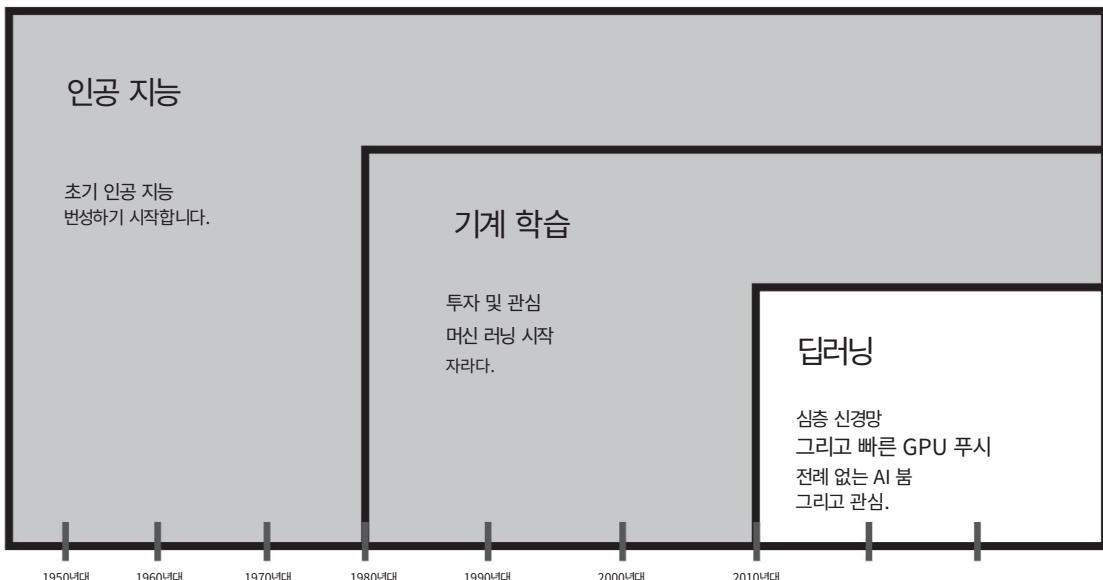


그림 1-1. AI, 머신러닝, 딥러닝의 관계 vi

2000년대 중반 딥 러닝의 부활은 일반적으로 인공 지능과 머신 러닝에 대한 연결주의적 접근 방식으로 다시 주목을 받았습니다. 딥 러닝의 부활, 고속 GPU(그래픽 처리 장치)의 가용성, 빅 데이터의 등장, Google, Facebook, Amazon, Microsoft 및 IBM과 같은 회사의 투자는 인공 지능의 르네상스를 촉발한 퍼펙트 스톰을 일으켰습니다..

AI 및 기계 학습 사용 사례

지난 10년 동안 기계 학습의 놀라운 발전이 있었습니다. 이러한 혁신은 우리의 일상 생활을 방해하고 생각할 수 있는 거의 모든 업종에 영향을 미치고 있습니다. 이것은 기계 학습 사용 사례의 완전한 목록은 아니지만 모든 산업에서 일어나고 있는 많은 혁신을 나타냅니다.

1장 기계 학습 소개

소매

소매업은 기계 학습의 이점을 처음으로 활용한 산업 중 하나였습니다. 수년 동안 온라인 쇼핑 사이트는 쇼핑 경험을 개인화하기 위해 협업 및 콘텐츠 기반 필터링 알고리즘에 의존해 왔습니다. 온라인 추천 및 고도로 타겟팅된 마케팅 캠페인은 소매업체에 수백만, 때로는 수십억의 수익을 창출합니다.

아마도 ML 기반 온라인 추천 및 개인화의 대표주자인 Amazon은 기계 학습의 이점을 실현한 가장 인기 있는(그리고 성공적인) 온라인 소매업체입니다. McKinsey에서 실시한 연구에 따르면 Amazon의 35%입니다.

com의 수익은 추천 엔진에서 나옵니다.^{vii} 소매를 위한 추가 기계 학습 응용 프로그램에는 선반 공간 계획, 플래노그램 최적화, 표적 마케팅, 고객 세분화 및 수요 예측이 포함됩니다.

운송

거의 모든 주요 자동차 제조업체는 심층 신경망으로 구동되는 AI 지원 자율 차량을 개발하고 있습니다. 이 차량에는 실시간 AI 인식, 탐색 및 경로 계획을 위해 초당 100조 이상의 작업(TOPS)을 처리할 수 있는 GPU 지원 컴퓨터가 장착되어 있습니다. UPS 및 FedEx와 같은 운송 및 물류 회사는 경로 및 연료 최적화, 차량 모니터링, 예방 유지보수, 이동 시간 추정, 지능형 지오펜싱을 위해 기계 학습을 사용합니다.

금융 서비스

고객평생가치(CLV) 예측, 신용 위험 예측 및 사기 탐지는 금융 서비스의 주요 기계 학습 사용 사례 중 일부입니다. 해지 펀드와 투자 은행은 기계 학습을 사용하여 Twitter Firehose의 데이터를 분석하여 시장을 움직일 가능성이 있는 트윗을 감지합니다. 금융 서비스에 대한 다른 일반적인 기계 학습 사용 사례에는 차선책 예측, 이탈 예측, 감정 분석 및 다중 채널 마케팅 속성이 포함됩니다.

1장 기계 학습 소개

의료 및 생명 공학

의료는 인공 지능과 기계 학습 연구 및 응용 분야에서 중요한 영역입니다. 병원과 의료 스타트업은 AI와 머신 러닝을 사용하여 심장병, 암, 결핵과 같은 생명을 위협하는 질병을 정확하게 진단하는 데 도움을 주고 있습니다. AI 기반 약물 발견과 이미징 및 진단은 AI가 견인력을 얻고 있는 가장 대표적인 분야입니다.^{viii} AI는 또 한 경로 분석, 마이크로어레이 분석, 유전자 예측 및 기능 주석의 새로운 혁신으로 이어지는 생명공학 및 유전체학 연구가 수행되는 방식에 혁명을 일으키고 있습니다.^{ix}

조작

미래 지향적인 제조업체는 품질 검사에 딥 러닝을 사용하여 하드웨어 제품의 균열, 고르지 않은 모서리 및 굵힘과 같은 결함을 감지합니다.

생존 분석은 중장비 장비의 고장 시간을 예측하기 위해 제조 및 산업 엔지니어에 의해 수년 동안 사용되었습니다. AI 기반 로봇은 제조 프로세스를 자동화하여 인간보다 더 빠르고 정밀하게 작동하여 생산성을 높이고 제품 결함을 줄입니다. 사물인터넷(IoT)의 도래와 풍부한 센서 데이터는 이 산업의 기계 학습 응용 프로그램의 수를 확대하고 있습니다.

정부

머신 러닝은 정부에서 광범위하게 적용됩니다. 예를 들어, 공공 유ти리티 회사는 기계 학습을 사용하여 유ти리티 파이프를 모니터링하고 있습니다.

이상 탐지 기술은 도시 전체의 서비스 중단과 수백만의 재산 피해를 유발할 수 있는 누출 및 파열 파이프를 탐지하는 데 도움이 됩니다. 머신 러닝은 실시간 수집 모니터링에도 사용되어 질병의 오염을 예방하고 잠재적으로 생명을 구할 수 있습니다. 에너지를 절약하기 위해 공기업 에너지 회사는 기계 학습을 사용하여 전기 사용량의 최고점과 최저점을 결정하여 그에 따라 에너지 출력을 조정합니다. AI 기반 사이버 보안은 특히 오늘날과 같이 빠르게 성장하는 또 다른 분야이자 중요한 정부 사용 사례입니다.

1장 기계 학습 소개

기계 학습 및 데이터

기계 학습 모델은 알고리즘과 데이터의 조합을 사용하여 구축됩니다. 강력한 알고리즘을 사용하는 것도 중요하지만 그에 빼지 않게 중요하고(일부는 더 중요하다고 말할 수도 있음) 충분한 양의 고품질 데이터로 훈련하는 것입니다. 일반적으로 머신 러닝은 데이터가 많을수록 더 잘 수행됩니다. 이 개념은 2001년 Microsoft 연구원 Michele Banko와 Eric Brill이 영향력 있는 논문 "Scaling to Very Large Corpora for Natural Language Disambiguation"에서 처음 발표했습니다.

Google의 연구 이사인 Peter Norvig은 "데이터의 불합리한 효율성"이라는 논문에서 이 개념을 더욱 대중화했습니다. 그러나 양보다 더 중요한 것은 품질입니다. 모든 고품질 모델은 고품질 기능으로 시작됩니다. 이것 이 기능 엔지니어링이 그림에 들어가는 곳입니다. 기능 엔지니어링은 원시 데이터를 고품질 기능으로 변환하는 프로세스입니다. 일반적으로 전체 기계 학습 프로세스에서 가장 힘든 부분이지만 가장 중요하기도 합니다. 기능 엔지니어링에 대해서는 이 장의 뒷부분에서 더 자세히 설명하겠습니다. 하지만 그 동안 일반적인 머신 러닝 데이터 세트를 살펴보겠습니다. 그림 1-2는 Iris 데이터 세트의 하위 집합을 보여줍니다. 이 책의 일부 예제에서 이 데이터 세트를 사용할 것입니다.

	꽃잎 길이	꽃잎 폭	꽃받침 길이	꽃받침 너비	클래스 레이블
1	3.2	4.8	7.6	6.4	버지니아
2	4.1	6.1	3.2	5.7	버지니아
4	5.4	5.5	4.5	3.9	부드러운
5	3.8	4.9	8.7	6.2	Versicolor
6	4.2	6.8	3.7	4.5	부드러운
7	5.7	7.3	4.6	4.1	Versicolor

그림 1-2. 머신러닝 데이터세트

1장 기계 학습 소개

관찰

단일 데이터 행을 관찰 또는 인스턴스라고 합니다.

특징

특징은 관찰의 속성입니다. 기능은 모델에 대한 입력으로 사용되는 독립 변수입니다. 그림 1-2에서 특징은 꽃잎 길이, 꽃잎 너비, 꽃받침 길이 및 꽃받침 너비입니다.

클래스 레이블

클래스 레이블은 데이터 세트의 종속 변수입니다. 우리가 예측하려고 하는 것입니다. 출력입니다. 이 예에서는 붓꽃의 종류(Virginica, Setosa, Versicolor)를 예측하려고 합니다.

모델

모델은 예측력이 있는 수학적 구성입니다. `dataset.Xi`의 독립 변수와 종속 변수 간의 관계를 추정합니다.

기계 학습 방법

다양한 유형의 머신 러닝 방법이 있습니다. 어떤 것을 사용할 것인지 결정하는 것은 달성하고자 하는 것과 가지고 있는 원시 데이터의 종류에 따라 크게 달라집니다.

지도 학습

지도 학습은 훈련 데이터 세트를 사용하여 예측을 수행하는 기계 학습 작업입니다. 지도 학습은 분류 또는 회귀로 분류할 수 있습니다.

회귀는 "가격", "온도" 또는 "거리"와 같은 연속 값을 예측하기 위한 것이고 분류는 "예" 또는 "아니요", "스팸" 또는 "스팸 아님" 또는 "악성"과 같은 범주를 예측하기 위한 것입니다. 또는 "양성".

1장 기계 학습 소개

분류

분류는 아마도 가장 일반적인지도 머신 러닝 작업일 것입니다. 인식하지 못한 채 분류를 활용한 애플리케이션을 이미 접했을 가능성이 큽니다. 인기 있는 사용 사례로는 의료 진단, 표적 마케팅, 스팸 감지, 신용 위험 예측, 감정 분석 등이 있습니다. 분류 작업에는 이진, 다중 클래스 및 다중 레이블의 세 가지 유형이 있습니다.

이진 분류

작업은 범주가 두 개뿐인 경우 이진 또는 이항 분류입니다. 예를 들어, 스팸 탐지를 위해 이진 분류 알고리즘을 사용할 때 출력 변수는 "스팸" 또는 "스팸 아님"의 두 가지 범주를 가질 수 있습니다. 암을 감지하기 위해 범주는 "악성" 또는 "양성"일 수 있습니다. 타겟 마케팅의 경우 누군가가 우유와 같은 품목을 구매할 가능성을 예측하는 경우 범주는 단순히 "예" 또는 "아니오"일 수 있습니다.

다중 클래스 분류

다중 클래스 또는 다항 분류 작업에는 3개 이상의 범주가 있습니다. 예를 들어, 기상 조건을 예측하기 위해 "비", "흐림", "맑음", "눈" 및 "바람"의 5가지 범주가 있을 수 있습니다. 타겟 마케팅 예를 확장하기 위해 다중 클래스 분류를 사용하여 고객이 전유, 저지방 우유, 저지방 우유 또는 탈지 우유를 구매할 가능성이 더 높은지 예측할 수 있습니다.

다중 레이블 분류

다중 레이블 분류에서는 각 관찰에 여러 범주를 할당할 수 있습니다. 대조적으로, 다중 클래스 분류에서는 하나의 범주만 관찰에 할당될 수 있습니다.

우리의 표적 마케팅 예를 사용하여 다중 라벨 분류는 고객이 우유뿐만 아니라 쿠키, 버터, 핫도그 또는 빵과 같은 다른 품목을 구매할 가능성이 더 높은지 예측하는 데 사용됩니다.

분류 및 회귀 알고리즘

다양한 분류 및 회귀 알고리즘이 수년에 걸쳐 개발되었습니다.

접근 방식, 기능, 복잡성, 사용 용이성, 교육 성능 및 예측 정확도가 다양합니다. 나는 다음 텍스트에서 가장 인기 있는 것에 대해 설명합니다. 3 장에서 더 자세히 다룹니다.

1장 기계 학습 소개

서포트 벡터 머신

SVM(Support Vector Machine)은 두 클래스 사이의 마진을 최대화하는 최적의 초평면을 찾고 데이터 포인트를 가능한 한 넓은 간격으로 별도의 클래스로 나누는 방식으로 작동하는 널리 사용되는 알고리즘입니다. 분류 경계에 가장 가까운 데이터 포인트를 지원 벡터라고 합니다.

로지스틱 회귀

로지스틱 회귀는 확률을 예측하는 선형 분류기입니다. 로지스틱(시그모이드) 함수를 사용하여 출력을 두 개의(이진) 클래스에 매핑할 수 있는 확률 값으로 변환합니다. 다중 클래스 분류는 다행 로지스틱(softmax) 회귀를 통해 지원됩니다.^{xii} 로지스틱 회귀와 같은 선형 분류기는 데이터에 명확한 결정 경계가 있을 때 적합합니다.

나이브 베이즈

Naive Bayes는 Bayes의 정리를 기반으로 하는 간단한 다중 클래스 선형 분류 알고리즘입니다. Naive Bayes는 데이터 세트의 기능이 독립적이라고 순진하게 가정하고 기능 간의 가능한 상관 관계를 무시하기 때문에 그 이름을 얻었습니다. 이것은 실제 시나리오의 경우가 아니지만 여전히 Naive Bayes는 특히 작은 데이터 세트 또는 높은 차원의 데이터 세트에서 잘 수행되는 경향이 있습니다. 선형 분류기와 마찬가지로 비선형 분류 문제에서는 성능이 좋지 않습니다. Naive Bayes는 데이터 세트에 대한 단일 패스만 필요로 하는 계산 효율적이고 확장성이 뛰어난 알고리즘입니다. 대규모 데이터 세트를 사용하는 분류 작업에 대한 좋은 기준 모델입니다. 특정 기능 집합이 주어진 클래스에 속할 확률을 찾는 방식으로 작동합니다.

다층 퍼셉트론

다층 퍼셉트론은 완전히 연결된 여러 노드 레이어로 구성된 피드포워드 인공 네트워크입니다. 입력 레이어의 노드는 입력 데이터 세트에 해당합니다. 중간 계층의 노드는 로지스틱(sigmoid) 함수를 사용하는 반면 최종 출력 계층의 노드는 softmax 함수를 사용하여 다중 클래스 분류를 지원합니다. 출력 레이어의 노드 수는 클래스 수와 일치해야 합니다.^{xiii}

1장 기계 학습 소개

의사결정나무

의사 결정 트리는 입력 변수에서 추론된 의사 결정 규칙을 학습하여 출력 변수의 값을 예측합니다. 시각적으로 결정 트리는 루트 노드가 맨 위에 있는 거꾸로 된 트리처럼 보입니다. 모든 내부 노드는 속성에 대한 테스트를 나타냅니다. 리프 노드는 클래스 레이블을 나타내고 개별 분기는 테스트 결과를 나타냅니다.

의사결정나무는 해석하기 쉽습니다. 로지스틱 회귀와 같은 선형 모델과 달리 의사 결정 트리에는 기능 확장이 필요하지 않습니다. 누락된 기능을 처리할 수 있으며 연속 및 범주형 기능 모두에서 작동합니다.^{xiv} 원-핫 인코딩 범주형 기능^{xv}는 필요하지 않으며 실제로 의사결정 트리 및 트리 기반 양상을 사용할 때 권장되지 않습니다. 원-핫 인코딩은 불균형 트리를 생성하고 우수한 예측 성능을 달성하기 위해 트리가 극도로 깊어야 합니다. 카디널리티가 높은 범주형 기능의 경우 특히 그렇습니다. 단점은 의사 결정 트리가 데이터의 노이즈에 민감하고 과적합되는 경향이 있다는 것입니다. 이러한 제한으로 인해 의사 결정 트리 자체는 실제 프로덕션 환경에서 거의 사용되지 않습니다. 오늘날 의사 결정 트리는 랜덤 포레스트 및 그라디언트 부스트 트리와 같은 보다 강력한 양상을 알고리즘의 기본 모델 역할을 합니다.

랜덤 포레스트

랜덤 포레스트는 분류 및 회귀를 위해 결정 트리 모음을 사용하는 양상을 알고리즘입니다. 낮은 편향을 유지하면서 분산을 줄이기 위해 배깅 (부트스트랩 집계)이라는 방법을 사용합니다. 배깅은 훈련 데이터의 하위 집합에서 개별 트리를 훈련합니다. 배깅 외에도 Random Forest는 기능 배깅이라는 또 다른 방법을 사용합니다. 배깅 (관측의 하위 집합 사용)과 달리 기능 배깅은 기능(열)의 하위 집합을 사용합니다. 기능 배깅은 의사 결정 트리 간의 상관 관계를 줄이는 것을 목표로 합니다. 기능 배깅이 없으면 개별 트리는 특히 몇 가지 주요 기능만 있는 상황에서 매우 유사합니다. 분류의 경우 개별 트리의 출력 또는 모드의 과반수 투표가 모델의 최종 예측이 됩니다. 회귀 분석의 경우 개별 트리 출력의 평균이 최종 출력이 됩니다(그림 3-3). 스파크는 각 트리가 랜덤 포레스트에서 독립적으로 훈련되기 때문에 여러 트리를 병렬로 훈련합니다.

1장 기계 학습 소개

그라디언트 부스트 트리

GBT(Gradient-Boosted Tree)는 랜덤 포레스트와 유사한 또 다른 트리 기반 양상을 알고리즘입니다. GBT는 부스팅이라는 기술을 사용하여 약한 학습자(얕은 나무)에서 강한 학습자를 만듭니다. GBT는 결정 트리의 양상을 순차적으로 훈련합니다.^{xvi} 각 후속 트리는 이전 트리의 오류를 줄입니다. 이것은 다음 모델에 맞추기 위해 이전 모델의 잔차를 사용하여 수행됩니다.^{xvii} 이 잔차 보정 프로세스^{xviii}는 잔차가 완전히 최소화될 때까지 교차 검증에 의해 결정된 반복 횟수로 설정된 반복 횟수로 수행됩니다.

XGBoost

XGBoost(eXtreme Gradient Boosting)는 현재 사용 가능한 최고의 그라디언트 부스트 트리 구현 중 하나입니다. 2014년 3월 27일에 Tianqi Chen이 연구 프로젝트로 출시한 XGBoost는 분류 및 회귀를 위한 지배적인 기계 학습 알고리즘이 되었습니다. XGBoost는 약한 학습자를 강한 학습자로 결합하는 그라디언트 부스팅의 일반 원리를 사용하여 설계되었습니다. 그러나 그라디언트 부스트 트리는 순차적으로 구축됩니다. 즉, 데이터로부터 천천히 학습하여 후속 반복에서 예측을 개선하는 반면 XGBoost는 트리를 병렬로 구축합니다.

XGBoost는 모델 복잡성을 제어하여 더 나은 예측 성능을 생성합니다. 내장된 정규화를 통해 과적합을 줄입니다. XGBoost는 연속 특징에 대한 최상의 분할 지점을 찾을 때 근사 알고리즘을 사용하여 분할 지점을 찾습니다.^{xix} 근사 분할 방법은 불연속 빈을 사용하여 연속 특징을 벅킹하여 모델 교육 속도를 크게 높입니다. XGBoost에는 연속 기능을 개별 빈으로 벅킹하는 훨씬 더 효율적인 방법을 제공하는 히스토그램 기반 알고리즘을 사용하는 또 다른 트리 성장 방법이 포함되어 있습니다. 그러나 근사 방법은 반복당 새로운 빈 세트를 생성하지만 히스토그램 기반 접근 방식은 여러 반복에서 빈을 재사용합니다. 이 접근 방식을 사용하면 빈 및 상위 및 형제 히스토그램 빼기^{xx}를 캐시하는 기능과 같이 근사 방법으로는 달성할 수 없는 추가 최적화가 가능합니다. 정렬 작업을 최적화하기 위해 XGBoost는 정렬된 데이터를 인메모리 블록 단위로 저장합니다. 정렬 블록은 병렬 CPU 코어에 의해 효율적으로 분산되고 수행될 수 있습니다. XGBoost는 가중 분위수 스케치 알고리즘을 통해 가중 데이터를 효과적으로 처리할 수 있고, 희소 데이터를 효율적으로 처리할 수 있으며, 캐시를 인식하고, 데이터가 메모리에 맞지 않도록 대용량 데이터 세트에 디스크 공간을 활용하여 코어 외 컴퓨팅을 지원합니다. XGBoost는 핵심 Spark MLlib 라이브러리의 일부가 아니지만 외부 패키지로 사용할 수 있습니다.

라이트GBM

수년 동안 XGBoost는 분류 및 회귀에 대해 모두가 가장 좋아하는 알고리즘이었습니다. 최근 LightGBM이 왕좌를 향한 새로운 도전자로 떠올랐습니다. XGBoost와 유사한 비교적 새로운 트리 기반 그래디언트 부스팅 변형입니다. LightGBM은 Microsoft의 DMTK(Distributed Machine Learning Toolkit) 프로젝트의 일부로 2016년 10월 17일에 출시되었습니다. 빠르고 분산되도록 설계되어 훈련 속도가 빨라지고 메모리 사용량이 적습니다. GPU 및 병렬 학습과 대규모 데이터 세트를 처리하는 기능을 지원합니다.

LightGBM은 여러 벤치마크와 공개 실험에서 보여졌습니다.

XGBoost보다 더 빠르고 더 정확한 데이터 세트를 제공합니다. XGBoost에 비해 몇 가지 장점이 있습니다. LightGBM은 히스토그램을 사용하여 연속적인 기능을 개별 빈으로 묶습니다. 이를 통해 LightGBM은 메모리 사용량 감소, 각 분할에 대한 이득 계산 비용 감소, 병렬 학습을 위한 통신 비용 감소와 같은 XGBoost(기본적으로 트리 학습에 사전 정렬 기반 알고리즘을 사용함)에 비해 몇 가지 성능 이점을 제공합니다. LightGBM은 노드의 히스토그램을 계산하기 위해 형제 및 부모에 대해 히스토그램 빼기를 수행하여 추가 성능 향상을 달성합니다. 온라인 벤치마크에서는 LightGBM이 일부 작업에서 XGBoost(비닝 없음)보다 11배에서 15배 더 빠릅니다.^{xxi} LightGBM은 일반적으로 나무를 잎사귀 방향(최상 우선)으로 성장시켜 정확도 면에서 XGBoost를 능가합니다. 의사 결정 트리를 훈련하는 데에는 레벨별 및 리프별의 두 가지 주요 전략이 있습니다. 레벨별 트리 성장은 대부분의 트리 기반 앙상블(XGBoost 포함)에 대한 의사 결정 트리를 성장시키는 전통적인 방법입니다. LightGBM은 잎사귀 성장 전략을 도입했습니다. 레벨별 성장과 달리 리프별 성장은 일반적으로 더 빠르게 수렴되고 ^{xxii} 더 낮은 손실을 달성합니다. ^{xxiii} [XGBoost](#) 와 마찬가지로 LightGBM은 핵심 Spark MLlib 라이브러리의 일부가 아니지만 외부 패키지로 사용할 수 있습니다. 대부분의 LightGBM 기능은 최근 XGBoost로 이식되었습니다. 3 장에서 두 알고리즘에 대해 더 자세히 설명 합니다.

회귀

회귀는 연속 숫자 값을 예측하기 위한 지도 머신 러닝 작업입니다. 인기 있는 사용 사례에는 판매 및 수요 예측, 주식, 주택 또는 상품 가격 예측, 일기 예보 등이 있습니다. Decision Tree, Random Forest, Gradient-Boosted Tree, XGBoost 및 LightGBM도 회귀에 사용할 수 있습니다.

회귀에 대해서는 3장에서 더 자세히 설명 합니다.

1장 기계 학습 소개

선형 회귀

선형 회귀는 하나 이상의 독립 변수와 종속 변수 간의 선형 관계를 조사하는 데 사용됩니다. 단일 독립 변수와 단일 연속 종속 변수 간의 관계 분석을 단순 선형 회귀라고 합니다. 다중 회귀는 다중 독립 변수를 기반으로 종속 변수의 값을 예측하기 위한 단순 선형 회귀의 확장입니다.

생존 회귀

사망까지의 시간 분석 또는 실패 시간 분석이라고도 하는 생존 회귀는 특정 이벤트가 발생할 시간을 예측하는 데 사용됩니다. 선형 회귀와 생존 회귀를 구별하는 주요 기능은 중도절단을 처리하는 능력입니다. 중도 절단은 사건 발생 시간을 알 수 없는 결측 데이터 문제 유형입니다.

비지도 학습

비지도 학습은 레이블이 지정된 응답의 도움 없이 데이터 세트에서 숨겨진 패턴과 구조를 찾는 기계 학습 작업입니다. 비지도 학습은 입력 데이터에만 액세스할 수 있고 훈련 데이터를 사용할 수 없거나 얻기 어려울 때 이상적입니다. 일반적인 방법에는 클러스터링, 주제 모델링, 이상 감지, 권장 사항 및 주성분 분석이 포함됩니다.

클러스터링

클러스터링은 몇 가지 유사점이 있는 레이블이 지정되지 않은 관찰을 그룹화하기 위한 감독되지 않은 기계 학습 작업입니다. 인기 있는 클러스터링 사용 사례에는 고객 세분화, 사기 분석 및 이상 감지가 포함됩니다. 클러스터링은 훈련 데이터가 부족하거나 사용할 수 없는 경우 분류기에 대한 훈련 데이터를 생성하는 데에도 자주 사용됩니다.

K-평균

K-Means는 클러스터링을 위한 가장 인기 있는 비지도 학습 알고리즘 중 하나입니다. K-평균은 각 클러스터의 시작점으로 사용되는 중심을 무작위로 할당하여 작동합니다. 알고리즘은 유클리드 거리를 기반으로 각 데이터 포인트를 가장 가까운 중심에 반복적으로 할당합니다. 그런 다음 각 클러스터에 대한 새 중심을 계산합니다.

1장 기계 학습 소개

해당 클러스터의 일부인 모든 포인트의 평균을 계산합니다. 알고리즘은 사전 정의된 반복 횟수에 도달하거나 모든 데이터 포인트가 가장 가까운 중심에 할당되고 더 이상 수행할 수 있는 재할당이 없을 때 반복을 중지합니다.

주제 모델링

주제 모델은 문서 그룹에서 주제(또는 주제)를 자동으로 파생합니다.

이러한 주제는 콘텐츠 기반 권장 사항, 문서 분류, 차원 축소 및 기능화에 사용할 수 있습니다.

잠재 디리클레 할당

LDA(Latent Dirichlet Allocation)는 2003년 David M. Blei, Andrew Ng 및 Michael Jordan에 의해 개발되었지만 인구 유전학에 사용된 유사한 알고리즘은 2000년 Jonathan K. Pritchard, Matthew Stephens 및 Peter Donnelly도 제안했습니다. LDA, 머신 러닝에 적용되는 그래픽 모델을 기반으로 하며 GraphX를 기반으로 구축된 Spark MLlib가 포함된 최초의 알고리즘입니다. Latent Dirichlet Allocation은 토픽 모델링에 널리 사용됩니다.

이상 감지

비정상 또는 이상값 탐지는 데이터 세트의 대부분에서 크게 벗어나 눈에 띠는 드문 관찰을 식별합니다. 몇 가지 사용 사례를 언급하자면 사기성 금융 거래를 발견하고 사이버 보안 위협을 식별하거나 예측 유지 관리를 수행하는 데 자주 사용됩니다.

고립의 숲

Isolation Forest는 Fei Tony Liu, Kai Ming Ting 및 Zhi-Hua Zhou가 개발한 트리 기반의 이상 감지 양상을 알고리즘입니다.^{xxiv} 대부분의 이상 감지 기술과 달리 Isolation Forest는 정상 데이터를 식별하는 대신 실제 이상값을 명시적으로 감지하려고 합니다 포인트들. Isolation Forest는 일반적으로 데이터 세트에 적은 수의 이상값이 있으므로 isolation 프로세스가 발생하기 쉽다는 사실을 기반으로 작동합니다.^{xxv} 일반 데이터 포인트에서 이상값을 격리하는 것은 더 적은 조건이 필요하기 때문에 효율적입니다. 대조적으로, 일반 데이터 포인트를 분리하려면 일반적으로 더 많은 조건이 필요합니다.

다른 트리 기반 양상을과 마찬가지로 Isolation Forest는 의사 결정 모음을 기반으로 합니다.

1장 기계 학습 소개

각 트리에는 전체 데이터 세트의 하위 집합이 있는 격리 트리로 알려진 트리가 있습니다. 이상 점수는 숲에 있는 나무의 평균 이상 점수로 계산됩니다. 이상 점수는 데이터 포인트를 분할하는 데 필요한 조건의 수에서 파생됩니다.

1에 가까운 이상 점수는 이상을 의미하고 0.5 미만의 점수는 비이상 관찰을 의미합니다.

단일 클래스 서포트 벡터 머신

지원 벡터 머신(SVM)은 가장 최적의 초평면을 사용하여 가능한 한 넓은 간격으로 데이터 포인트를 별도의 클래스로 나누어 데이터를 분류합니다. 단일 클래스 SVM에서 모델은 "정상" 클래스가 하나만 있는 데이터에 대해 학습됩니다. 일반적인 예와 다른 데이터 포인트는 이상으로 간주됩니다.

차원 축소

데이터 세트에 많은 수의 기능이 있는 경우 차원 축소가 필수적입니다. 예를 들어, 유전체학 및 산업 분석 분야의 기계 학습 사용 사례에는 일반적으로 수천 또는 수백만 개의 기능이 포함됩니다. 높은 차원은 모델을 더 복잡하게 만들어 과적합 가능성을 높입니다. 특정 지점에 더 많은 기능을 추가하면 실제로 모델의 성능이 저하됩니다.

또한 고차원 데이터에 대한 교육에는 상당한 컴퓨팅 리소스가 필요합니다.

이것들은 집합적으로 차원의 저주로 알려져 있습니다. 차원 축소 기술은 차원의 저주를 극복하는 것을 목표로 합니다.

주요 구성 요소 분석

주성분 분석(PCA)은 특징 공간의 차원을 줄이는 데 사용되는 비지도 머신 러닝 기술입니다. 기능 간의 상관 관계를 감지하고 원래 데이터 세트에서 대부분의 분산을 유지하면서 선형으로 상관되지 않은 기능의 감소된 수를 생성합니다. 이러한 보다 간결하고 선형적으로 관련이 없는 기능을 주성분이라고 합니다. 주성분은 설명된 분산의 내림차순으로 정렬됩니다. 다른 차원 축소 기술에는 특이값 분해(SVD) 및 선형 판별 분석(LDA)이 포함됩니다.

권장 사항

개인화된 추천을 제공하는 것은 기계 학습의 가장 인기 있는 응용 프로그램 중 하나입니다. Amazon, Alibaba, Walmart 및 Target과 같은 거의 모든 주요 소매업체는 고객 행동을 기반으로 일종의 개인화된 추천을 제공합니다. Netflix, Hulu 및 Spotify와 같은 스트리밍 서비스는 사용자의 취향과 선호도에 따라 영화 또는 음악 추천을 제공합니다. 협업 필터링, 콘텐츠 기반 필터링 및 연관 규칙 학습(장바구니 분석용)은 추천 시스템을 구축하는 데 가장 널리 사용되는 방법입니다. Spark MLLib는 협업 필터링을 위한 ALS(Alternating Least Squares)와 장바구니 분석을 위한 FP-Growth 및 PrefixSpan을 지원합니다.

반 지도 학습

경우에 따라 레이블이 지정된 데이터에 액세스하는 데 많은 비용과 시간이 소요됩니다. 레이블이 지정된 응답이 부족한 상황에서 semi-supervised learning은 지도 학습 기술과 unsupervised 학습 기술을 모두 결합하여 예측을 수행합니다. 준지도 학습에서 레이블이 지정되지 않은 데이터는 레이블이 지정된 데이터를 보강하여 모델 정확도를 향상시키는 데 사용됩니다.

강화 학습

강화 학습은 어떤 행동이 가장 큰 보상을 제공하는지 결정하기 위해 시행착오를 통해 배우려고 합니다. 강화 학습에는 에이전트(의사 결정자 또는 학습자), 환경(에이전트와 상호 작용하는 것) 및 작업(에이전트가 수행할 수 있는 것)의 세 가지 구성 요소가 있습니다.^{xxvi} 이러한 유형의 학습은 게임, 탐색, 그리고 로봇.

딥러닝

딥 러닝은 심층 다층 인공 신경망을 사용하는 기계 학습 및 인공 지능의 하위 분야입니다. 이는 최근 인공 지능의 많은 혁신을 주도하고 있습니다. 딥 러닝은 보다 평범한 분류 작업에 사용될 수 있지만 의료 진단, 얼굴 인식, 자율 주행 자동차, 사기 분석 및 지능형 음성 제어 비서와 같은 보다 복잡한 문제에 적용될 때 진정한 힘을 빛납니다.^{xxvii} 특정 영역에서, 딥 러닝은 컴퓨터가 인간의 능력과 일치하고 심지어 능가하는 것을 가능하게 했습니다.

1장 기계 학습 소개

신경망

신경망은 인간 두뇌의 상호 연결된 뉴런처럼 작동하는 알고리즘 클래스입니다. 신경망은 상호 연결된 노드로 구성된 여러 레이어를 포함합니다. 일반적으로 입력 레이어, 하나 이상의 은닉 레이어 및 출력 레이어가 있습니다.

데이터는 입력 레이어를 통해 신경망을 통과합니다. 은닉층은 가중치 연결 네트워크를 통해 데이터를 처리합니다. 은닉층의 노드는 입력에 가중치를 할당하고 그 과정에서 계수 세트와 결합합니다. 데이터는 계층의 출력을 결정하는 노드의 활성화 기능을 거칩니다. 마지막으로 데이터는 신경망의 최종 출력을 생성하는 출력 레이어에 도달합니다.^{xxviii} 여러 은닉 레이어가 있는 신경망을 "심층" 신경망이라고 합니다. 층이 많을수록 네트워크가 더 깊어지고 일반적으로 네트워크가 더 깊어질수록 학습이 더 정교해지고 해결할 수 있는 문제가 더 복잡해집니다.

컨볼루션 신경망

합성곱 신경망(줄여서 convnet 또는 CNN)은 이미지 분석에 특히 뛰어난 신경망 유형입니다(오디오 및 텍스트 데이터에도 적용할 수 있음). 컨볼루션 신경망 계층의 뉴런은 높이, 너비 및 깊이의 3차원으로 배열됩니다. CNN은 합성곱 계층을 사용하여 텍스처 및 가장자리와 같은 입력 특징 공간(이미지)의 로컬 패턴을 학습합니다. 대조적으로, 완전히 연결된(조밀한) 계층은 전역 패턴을 학습 합니다.^{xxix} 컨볼루션 계층의 뉴런은 조밀한 계층의 경우와 같이 모든 뉴런 대신에 선형 계층의 작은 영역에만 연결됩니다. 조밀한 계층의 완전히 연결된 구조는 비효율적이며 빠르게 과적 합으로 이어질 수 있는 매우 많은 수의 매개변수로 이어질 수 있습니다.

피처 엔지니어링

기능 엔지니어링은 데이터를 변환하여 기계 학습 모델을 훈련하는 데 사용할 수 있는 기능을 생성하는 프로세스입니다. 종종 원시 데이터는 여러 데이터 준비 및 추출 기술을 통해 변환해야 합니다. 예를 들어, 매우 큰 데이터 세트의 경우 차원 축소가 필요할 수 있습니다. 새로운 기능

1장 기계 학습 소개

다른 기능에서 만들어야 할 수도 있습니다. 거리 기반 알고리즘에는 기능 확장이 필요합니다. 일부 알고리즘은 범주형 기능이 원-핫 인코딩될 때 더 잘 수행됩니다.

텍스트 데이터에는 일반적으로 토큰화 및 기능 벡터화가 필요합니다. 원시 데이터를 기능으로 변환한 후 평가하여 예측력이 가장 높은 항목을 선택합니다.

피쳐 엔지니어링은 기계 학습의 중요한 측면입니다. 성공하려면 관련성이 높은 기능을 생성하려는 거의 모든 기계 학습 노력에서 공리입니다. 불행히도 기능 엔지니어링은 종종 도메인 전문 지식이 필요한 복잡하고 시간 소모적인 작업입니다. 기능을 브레인스토밍하고 생성하고 모델 정확도에 미치는 영향을 연구하는 반복적인 프로세스입니다. 사실, 일반적인 데이터 과학자들은 Forbes의 설문 조사에 따라 데이터를 준비하는 데 대부분의 시간을 보냅니다 ([그림 1-3\).xxxii](#)

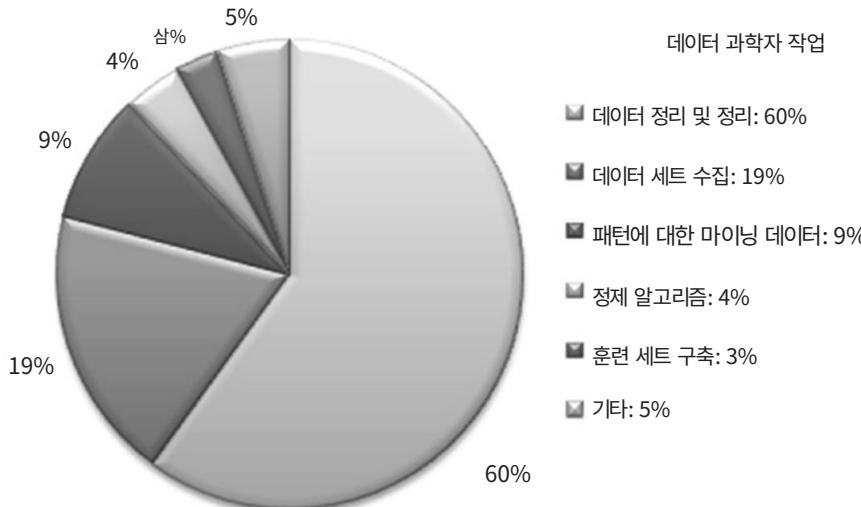


그림 1-3. 데이터 준비는 데이터 과학자 작업의 약 80%를 차지합니다.

기능 엔지니어링 작업은 다음과 같은 여러 범주로 나눌 수 있습니다.
기능 중요도, 기능 추출 및 기능 구성.[xxxii](#)

기능 선택

기능 선택은 중요한 기능을 식별하고 관련이 없거나 중복되는 기능을 제거하는 중요한 전처리 단계입니다. 예측 성능과 모델 훈련 효율성을 향상시키고 차원을 줄입니다. 관련 없는 기능은 다음과 같아야 합니다.

1장 기계 학습 소개

모델의 정확도에 부정적인 영향을 미치고 모델 학습 속도가 느려질 수 있으므로 제거되었습니다. 특정 기능은 예측력이 없거나 다른 기능과 중복될 수 있습니다. 그러나 기능이 관련성이 있는지 어떻게 판단합니까?

아니면? 도메인 지식은 중요합니다. 예를 들어, 대출 불이행 확률을 예측하는 모델을 구축하는 경우 신용 위험을 수량화할 때 고려해야 할 요소를 아는 것이 도움이 됩니다. 차용인의 소득 대비 부채 비율부터 시작할 수 있습니다.

차용인의 신용 점수, 고용 기간, 직위 및 결혼 상태와 같은 차용 관련 요인을 고려해야 합니다. 경제 성장과 같은 시장 전반의 고려 사항도 중요할 수 있습니다. 인구 통계 및 심리 정보도 고려해야 합니다. 기능 목록이 있으면 그 중요성을 객관적으로 결정할 수 있는 여러 가지 방법이 있습니다. 모델에 적합한 기능을 선택하는 데 도움이 되는 다양한 기능 선택 방법이 있습니다.^{xxxiii}

필터 방법

필터 방법은 카이 제곱 테스트, 상관 계수 및 정보 이득과 같은 통계 기술을 사용하여 각 기능에 순위를 지정합니다.

래퍼 메서드

래퍼 메서드는 기능의 하위 집합을 사용하여 모델을 훈련합니다. 그런 다음 모델의 성능에 따라 기능을 추가하거나 삭제할 수 있습니다. 래퍼 메서드의 일반적인 예로는 재귀적 기능 제거, 역방향 제거 및 순방향 선택이 있습니다.

임베디드 메소드

포함된 메서드는 필터 및 래퍼 메서드에서 사용하는 기술을 결합합니다. 인기 있는 예로는 LASSO 및 RIDGE 회귀, 정규화 트리 및 임의 다행 logit.^{xxxiv}가 있습니다.

기능 중요도

Random Forest, XGBoost, LightGBM과 같은 트리 기반 앙상블은 각 특성에 대해 계산된 특성 중요도 점수를 기반으로 특성 선택 방법을 제공합니다. 점수가 높을수록 모델 정확도를 높이는 데 기능이 더 중요합니다. 랜덤 포레스트의 기능 중요도는 지니 기반 중요도 또는 평균 불순물 감소(MDI)라고도 합니다. Random Forest의 일부 구현은 다음을 활용합니다.

1장 기계 학습 소개

정확도 기반 중요도 또는 평균 정확도 감소(MDA)로 알려진 특정 중요도를 계산하는 다른 방법.^{xxxv} 정확도 기반 중요도는 특성이 무작위로 치환될 때 예측 정확도의 감소를 기반으로 계산됩니다. 3 장에서 Random Forest, XGBoost 및 LightGBM의 기능 중요성에 대해 자세히 설명 합니다.

상관 계수는 특징 선택 방법의 기본적인 형태입니다.

상관 계수는 두 변수 간의 선형 관계의 강도를 나타냅니다. 선형 문제의 경우 상관 관계를 사용하여 관련 기능(기능 클래스 상관 관계)을 선택하고 중복 기능(기능 내 상관 관계)을 식별할 수 있습니다.

특징 추출

데이터 세트에 많은 수의 기능이 있는 경우 기능 추출이 필수적입니다.

예를 들어, 유전체학 및 산업 분석 분야의 기계 학습 사용 사례에는 일반적으로 수천 또는 수백만 개의 기능이 포함됩니다. 높은 차원은 모델을 더 복잡하게 만들어 과적합 가능성을 높입니다. 또한 고차원 데이터에 대한 교육에는 상당한 컴퓨팅 리소스가 필요합니다. 특징 추출은 일반적으로 차원 축소 기술을 사용합니다. 주성분 분석(PCA), 선형 판별 분석(LDA) 및 특이값 분해(SVD)는 특징 추출에도 사용되는 가장 널리 사용되는 차원 축소 알고리즘 중 일부입니다.

기능 구성

모델의 정확도를 향상시키기 위해 기존 기능에서 새 기능을 구성해야 하는 경우가 있습니다. 이를 수행하는 방법에는 여러 가지가 있습니다. 기능을 결합하거나 집계할 수 있습니다. 경우에 따라 분할할 수 있습니다. 예를 들어, 모델은 대부분의 트랜잭션 데이터에서 일반적으로 사용되는 타임스탬프 속성을 초, 분, 시간, 일, 월 및 연도와 같은 몇 가지 더 세분화된 속성으로 분할하여 이점을 얻을 수 있습니다. 그런 다음 이러한 속성을 사용하여 dayofweek, weekofmonth, monthofyear 등과 같은 더 많은 기능을 구성할 수 있습니다. 기능 구성은 일부는 예술이고 일부는 과학이며 기능 엔지니어링에서 가장 어렵고 시간이 많이 소요되는 측면 중 하나입니다. 기능 구성의 숙달은 일반적으로 노련한 데이터 과학자와 초보자를 구별하는 것입니다.

1장 기계 학습 소개

모델 평가

분류에서 각 데이터 포인트에는 알려진 레이블과 모델 생성 예측 클래스가 있습니다. 알려진 레이블과 각 데이터 포인트에 대한 예측된 클래스를 비교하여 결과를 네 가지 범주 중 하나로 분류할 수 있습니다. 예측된 클래스가 양성이고 레이블이 양성인 경우 참 양성(TP), 예측된 경우 참 음성(TN) 클래스가 음수이고 레이블이 음수, 예측된 클래스가 양수이지 만 레이블이 음수인 경우 위양성(FP), 예측된 클래스가 음수이고 레이블이 양수인 경우 위음성(FN)입니다. 이 네 가지 값은 분류 작업에 대한 대부분의 평가 메트릭의 기초를 형성합니다. 그들은 종종 혼동 행렬이라는 표에 표시됩니다(표 1-1).

표 1-1. 혼란 매트릭스

		음수(예측)	긍정적(예상)
음수(실제)	트루 네거티브	거짓 긍정	
	거짓 부정	트루 포지티브	

정확성

정확도는 분류 모델에 대한 평가 메트릭입니다. 그것은 정확한 예측의 수를 전체 예측의 수로 나눈 값으로 정의됩니다.

$$\text{정확성} = \frac{\text{참 긍정} + \text{참 부정}}{\text{참 양성} + \text{참 음성} + \text{거짓 양성} + \text{거짓 음성}}$$

데이터 세트가 불균형한 상황에서는 정확도가 이상적인 메트릭이 아닙니다.

예를 들어 설명하기 위해 90개의 음성 샘플과 10개의 양성 샘플이 있는 가상의 분류 작업을 고려합니다. 모두 부정으로 분류하면 0.90의 정확도 점수를 제공합니다. 정밀도와 재현율은 클래스 불균형 데이터로 훈련된 모델을 평가하기 위한 더 나은 지표입니다.

정도

정밀도는 참 양성의 수를 참 양성의 수에 거짓 양성의 수를 더한 것으로 나눈 값으로 정의됩니다. 정밀도는 예측이 긍정적일 때 모델이 얼마나 자주 올바른지 보여줍니다. 예를 들어, 모델에서 100건의 암 발생을 예측했지만 그 중 10건이 잘못된 예측이었다면 모델의 정밀도는 90%입니다.

정밀도는 가양성의 비용이 높은 상황에서 사용하기에 좋은 메트릭입니다.

$$\text{정도} = \frac{\text{트루 포지티브}}{\text{참 긍정 거짓 긍정} +}$$

상기하다

재현율은 위음성의 비용이 높은 상황에서 사용하기에 좋은 지표입니다. 재현율은 참 양성의 수를 참 양성의 수와 거짓 음성의 수로 나눈 값으로 정의됩니다.

$$\text{정도} = \frac{\text{트루 포지티브}}{\text{참 긍정 거짓 부정} +}$$

F1 측정

F1 측정값 또는 F1 점수는 정밀도와 재현율의 조화 평균 또는 가중 평균입니다. 다중 클래스 분류기를 평가하기 위한 일반적인 성능 메트릭입니다. 불균등한 등급 분포가 있는 경우에도 좋은 척도입니다. 가장 좋은 F1 점수는 1이고 가장 나쁜 점수는 0입니다. 좋은 F1 측정은 낮은 거짓 음성과 낮은 거짓 양성이 있음을 의미합니다. F1 측정은 다음과 같이 정의됩니다.

$$F\text{ 측정} = \frac{\text{정밀도} \times \text{재현율}}{\text{정밀 리콜} +}$$

수신기 작동 특성 아래 영역 (오록)

AUROC(수신기 작동 특성 아래 영역)는 이진 분류기를 평가하기 위한 일반적인 성능 메트릭입니다. ROC(수신기 작동 특성)는 거짓 긍정 비율에 대해 참 긍정 비율을 표시하는 그래프입니다. 곡선 아래 면적(AUC)은 ROC 곡선 아래 면적입니다. AUC는 다음과 같이 해석 될 수 있습니다.

1장 기계 학습 소개

모델이 임의의 부정적인 예보다 임의의 긍정적인 예를 더 높은 순위로 지정할 확률 .xxxvi 곡선 아래의 면적이 클수록(AUROC가 1.0에 가까울수록) 모델의 성능이 더 좋습니다. AUROC가 0.5인 모델은 예측 정확도가 무작위 추측만큼 좋기 때문에 쓸모가 없습니다.

과대적합과 과소적합

모델의 저조한 성능은 과대적합 또는 과소적합으로 인해 발생합니다. 과적합은 훈련 데이터에 너무 잘 맞는 모델을 말합니다. 과적합된 모델은 훈련 데이터에 대해서는 잘 수행되지만 새로운 보이지 않는 데이터에 대해서는 성능이 좋지 않습니다. 과적합의 반대는 과소적합이다. underfitting을 사용하면 모델이 너무 단순하고 모델이 과도하게 정규화되었거나 더 오래 훈련되어야 하기 때문에 훈련 데이터 세트에서 관련 패턴을 학습하지 못했습니다. 모델이 보이지 않는 새로운 데이터에 잘 맞는 능력을 일반화라고 합니다. 이것은 모든 모델 튜닝 연습의 목표입니다. 과적합을 방지하기 위한 몇 가지 확립된 방법에는 더 많은 데이터 또는 기능의 하위 집합 사용, 교차 검증, 삭제, 가지치기, 조기 중지 및 정규화가 포함 됩니다.xxxxvii 딥 러닝의 경우 데이터 증대는 일반적인 형식의 정규화입니다. underfitting을 줄이기 위해 더 많은 관련 기능을 추가하는 것이 권장되는 옵션입니다. 딥 러닝의 경우 계층에 더 많은 노드를 추가하거나 신경망에 더 많은 계층을 추가하여 모델의 용량을 늘리는 것을 고려하십시오 .xxxxviii

모델 선택

모델 선택에는 피팅된 기계 학습 모델을 평가하고 기본 추정기를 하이퍼파라미터의 사용자 지정 조합으로 피팅하여 최상의 모델을 출력하는 작업이 포함됩니다. Spark MLlib를 사용하면 CrossValidator 및 TrainValidationSplit 추정기로 모델 선택이 수행됩니다. CrossValidator는 하이퍼파라미터 튜닝 및 모델 선택을 위해 k-겹 교차 검증 및 그리드 검색을 수행합니다. 데이터 세트를 훈련 및 테스트 데이터 세트로 활용되는 겹치지 않는 무작위 분할 폴드 세트로 분할합니다. 예를 들어 k=3 폴드인 경우 k-폴드 교차 검증은 3개의 교육 및 테스트 데이터 세트 쌍을 생성합니다(각 접는 테스트 데이터 세트로 한 번만 사용됨). [test.xxxxix](#) 의 경우 TrainValidationSplit은 초매개변수 조정을 위한 또 다른 추정기입니다. k-겹 교차 검증(비용이 많이 드는 작업)과 달리 TrainValidationSplit은 매개변수의 각 조합을 k 번이 아니라 한 번만 평가합니다.

요약

이 장에서는 기계 학습에 대한 빠른 소개를 제공합니다. 보다 철저한 치료를 위해 Trevor Hastie 등 의 Elements of Statistical Learning, 2nd ed.를 제안합니다.

(Springer, 2016) 및 Gareth James, et al.의 통계 학습 소개 .
(Springer, 2013). 딥러닝 입문용으로는 딥러닝 을 추천합니다

Ian Goodfellow 외 (MIT 언론, 2016). 머신 러닝은 오랫동안 사용되어 왔지만 빅 데이터를 사용하여 머신 러닝 모델을 훈련시키는 것은 상당히 최근에 개발된 것입니다. 가장 널리 사용되는 빅 데이터 프레임워크인 Spark 는 대규모 엔터프라이즈급 머신 러닝 애플리케이션을 구축하기 위한 탁월한 플랫폼으로 독보적인 위치에 있습니다. 다음 장에서는 Spark 및 Spark MLlib에 대해 자세히 알아보겠습니다.

참고문헌

- 나. 라피 카차두리안; "둠스데이 발명품"
newyoker.com, 2015년, www.newyoker.com/Magazine/2015/11/23/doomsday-invention-artificial-intelligence-nick-bostrom
- ii. 존 매카시; "인공 지능이란 무엇입니까?", stanford.edu, 2007, www-formal.stanford.edu/jmc/whatisai/node1.html
- iii. 크리스 니콜슨; "인공 지능(AI) 대 기계 학습 대 딥 러닝", skimind.ai, 2019, https://skymind.ai/wiki/인공_지능_대_기계_학습_대_딥_러닝
- iv. Marta Garnelo 및 Murray Shanahan; "심볼 인공 지능과 딥 러닝의 조화: 객체와 관계 표현", sciencedirect.com, 2019, www.sciencedirect.com/science/article/pii/S2352154618301943
- v. 크리스 니콜슨; "Symbolic Reasoning(Symbolic AI)과 기계 학습"; 스카이마인드.ai, 2019, https://skymind.ai/wikis/상징적_추론

1장 기계 학습 소개

vi. 마이클 코플랜드; “인공과의 차이점은 무엇입니까?
지능, 기계 학습 및 딥 러닝?”, nvidia.com, 2016, <https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>

vii. Ian MacKenzie, et al.; “소매업체가 소비자를 따라잡을 수 있는 방법”,
mckinsey.com, 2013, www.mckinsey.com/
산업/소매/우리의 통찰력/소매업체가 소비자를 따라갈 수 있는 방법

viii. CB Insights; “약물 R&D에서 진단까지: 의료 분야의 90개 이상의 인공 지능 스타트업”, cbinsights.com, 2019년, www.cbinsights.com/research/artificial-intelligence-startups-healthcare/

ix. 라고타만 예나말리; “생물학에서 기계 학습의 응용”, kolabtree.com, 2019,
www.kolabtree.com/blog/
생물학에서 기계 학습의 응용 프로그램/

엑스. 자비에 아마트리아인; “머신 러닝에서 더 나은 점:
더 많은 데이터 또는 더 나은 알고리즘,” kdnuggets.com, 2015, www.kdnuggets.com/2015/06/machine-learning-more-data-better-algorithm.html

xi. 모하메드 굴러; “Spark를 사용한 빅 데이터 분석”, Apress, 2015년

xii. 아파치 스파크; “다항 로지스틱 회귀”, spark.apache.org, 2019, <https://spark.apache.org/docs/latest/ml-classification-regression.html#multinomial-logistic-회귀>

xiii. 아파치 스파크; “다층 퍼셉트론 분류기”, spark.apache.org, 2019, <https://spark.apache.org/docs/latest/ml-classification-regression.html#multilayer-perceptron-분류기>

xiv. 분석 Vidhya 콘텐츠 팀; “나무에 대한 완전한 튜토리얼
스크래치 기반 모델링(R 및 Python),” AnalyticsVidhya.com, 2016년, www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/#one

1장 기계 학습 소개

xv. 라이트GBM; "범주별 기능을 위한 최적의 분할", lightgbm.

readthedocs.io, 2019, <https://lightgbm.readthedocs.io/en/latest/Features.html>

16. Joseph Bradley와 Manish Amde; "랜덤 포레스트와

Mlib에서 부스팅," Databricks, 2015, <https://databricks.com/blog/2015/01/21/random-forests-and-boosting-in-mllib.html>

xvii. 분석 Vidhya 컨텐츠 팀; "에 대한 엔드 투 엔드 가이드

XGBoost 이면의 수학 이해", analyticsvidhya.com, 2018, www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/

xviii. 벤 고먼; "A Kaggle Master가 Gradient Boosting에 대해 설명합니다."

Kaggle.com, 2017, <http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/>

xix. 리나 쇼; "XGBoost: 간결한 기술 개요,"

KDNuggets, 2017, www.kdnuggets.com/2017/10/xgboost-concrete-technical-overview.html

더블 엑스. 조현수; "Fast Histogram Optimized Grower, 8배에서 10배까지의 속도 향상", DMLC,

2017, <https://github.com/dmlc/xgboost/tree/1950>

xxi. 로라; "LightGBM 벤치마킹: LightGBM 대 xgboost는 얼마나 빠릅니까?",

medium.com, 2017, <https://medium.com/implodinggradients/benchmarking-lightgbm-how-fast-is-lightgbm-vs-xgboost-15d224568031>

xxii. 라이트GBM; "속도 및 메모리 사용 최적화", lightgbm.

readthedocs.io, 2019, <https://lightgbm.readthedocs.io/en/latest/Features.html>

xxiii. 데이비드 막스; "의사결정 트리: 리프 방식(최상 우선) 및 레벨 방식 트리 탐색",

stackexchange.com, 2018년, <https://datascience.stackexchange.com/questions/26699/decision-trees-leaf-wise-best-first-and-level-wise-tree-traverse>

1장 기계 학습 소개

- xxiv. Fei Tony Liu, Kai Ming Ting, Zhia-Hua Zhou, "Isolation Forest", acm.org, 2008, <https://dl.acm.org/citation.cfm?id=1511387>
- xxv. 알레한드로 코레아 반센; "격리 포리스트를 사용한 이상 탐지의 이점" easysol.net, 2016, <https://blog.easysol.net/using-isolation-forests-anomaly-detection/>
- xxvi. SAS; "기계 학습", sas.com, 2019년, www.sas.com/en_us/insights/analytics/machine-learning.html
- xxvii. 엔비디아; "딥 러닝", developer.nvidia.com, 2019년, <https://developer.nvidia.com/deep-learning>
- xxviii. SAS; "신경망 작동 방식", sas.com, 2019년, www.sas.com/ko_us/insights/analytics/neural-networks.html
- xxix. 프링수아 솔레; "컴퓨터 비전을 위한 딥 러닝", 2018, Python을 사용한 딥 러닝 트리플 엑스. 안드레이 카르파티; "컨볼루션 신경망(CNN/ConvNets)", github.io, 2019, <http://cs231n.github.io/convolutional-neural-networks/>
- xxx. 길프레스; "빅 데이터 정리: 가장 시간이 많이 걸리는, 최소한의 즐거운 데이터 과학 작업, 설문 조사 결과", forbes.com, 2016년, www.forbes.com/sites/gilpress/2016/03/23/data-preparation/ 가장 시간이 많이 걸리는 가장 적은 시간을 즐길 수 있는 데이터 과학 작업 조사 말/#680347536f63
- xxxii. 제이슨 브라운리; machinelearningmastery, "기능 엔지니어링, 기능 엔지니어링 방법 및 능숙하게 활용하는 방법을 알아보세요." com, 2014년, <https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/>
- xxxiii. 제이슨 브라운리; "기능 선택 소개", MachineLearningMastery.com, 2014, <https://machinelearningmastery.com/an-introduction-to-feature-selection/>

1장 기계 학습 소개

xxxiv. 사우라프 카우식; "특징 선택 방법 소개

예," Analyticsvidhya.com, 2016, www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variable/

xxxv. 제이크 호어; "Random Forest에 대한 변수 중요도는 어떻게 계산됩니까?" DisplayR, 2018, www.displayr.com/how-is-variable-importance-calculated-for-a-random-forest/

xxxvi. Google; "분류: ROC 곡선 및 AUC" 개발자.

google.com, 2019년, <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>

xxxvii. 웨인 톰슨; "머신 러닝 모범 사례:

"일반화 이해", blogs.sas.com, 2017, <https://blogs.sas.com/content/subconsciousmusings/2017/09/05/guide-machine-learning-best-practices-understanding-generalization/>

xxxviii. 제이슨 브라운리; "딥 러닝에서 과적합을 피하는 방법

신경망," machinelearningmaster.com, 2018, <https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/>

xxxix. 불꽃; "CrossValidator", spark.apache.org, 2019, <https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.ml.tuning.CrossValidator>

제 2 장

Spark 및 Spark MLlib 소개

단순한 모델과 많은 데이터가 더 적은 데이터를 기반으로 하는 보다 정교한 모델보다 우선 합니다.

—피터 노비기

Spark는 대규모 데이터 세트를 처리하고 분석하기 위한 통합 빅 데이터 처리 프레임워크입니다. Spark는 기계 학습을 위한 MLlib, SQL 지원을 위한 Spark SQL, 실시간 스트리밍을 위한 Spark Streaming, 그래프 처리를 위한 GraphX를 포함한 강력한 라이브러리와 함께 Scala, Python, Java 및 R의 고급 API를 제공합니다. ii Spark는 Matei에 의해 설립되었습니다. 캘리포니아 대학교 버클리의 AMPLab에 있는 Zaharia는 나중에 Apache Software Foundation에 기부되어 2014년 2월 24일에 최상위 프로젝트가 되었습니다. iii 첫 번째 버전은 2017년 5월 30일에 릴리스되었습니다. iv

개요

Spark는 Hadoop의 원래 데이터 처리 프레임워크인 MapReduce의 한계를 해결하기 위해 개발되었습니다. Matei Zaharia는 UC Berkeley와 Facebook(그가 인턴십을 했던 곳)에서 MapReduce의 한계를 보았고 반복적이고 상호 작용하는 응용 프로그램을 처리할 수 있는 더 빠르고 일반화된 다목적 데이터 처리 프레임워크를 만들려고 했습니다. v 통합 플랫폼을 제공합니다(그림 2-1) 스트리밍, 대화형, 그래프 처리, 기계 학습 및 batch.vi와 같은 여러 유형의 워크로드를 지원합니다.

Spark 작업은 빠른 인메모리 기능과 고급 DAG(방향성 비순환 그래프) 실행 엔진으로 인해 동등한 MapReduce 작업보다 몇 배 더 빠르게 실행할 수 있습니다.

Spark는 Scala로 작성되었으며 결과적으로 Spark는

2장 Spark 및 Spark MLlib 소개

불꽃. 우리는 이 책 전체에서 스칼라를 사용할 것입니다. 7 장에서 분산 딥 러닝을 위해 Python API인 PySpark를 사용할 것입니다. 이 장은 필자의 저서 '차세대 빅데이터(Next-Generation Big Data)' (Apress, 2018) 의 5장을 업데이트한 것이다.



그림 2-1. Apache Spark 에코시스템

클러스터 관리자

클러스터 관리자는 클러스터 리소스를 관리하고 할당합니다. Spark는 Spark(Standalone Scheduler), YARN, Mesos 및 Kubernetes와 함께 제공되는 독립 실행형 클러스터 관리자를 지원합니다.

건축물

높은 수준에서 Spark는 클러스터 노드 전체에 Spark 애플리케이션의 작업 실행을 분산합니다(그림 2-2). 모든 Spark 응용 프로그램에는 해당 드라이버 프로그램 내에 SparkContext 객체가 있습니다. SparkContext는 Spark 애플리케이션에 컴퓨팅 리소스를 제공하는 클러스터 관리자에 대한 연결을 나타냅니다. 예 연결한 후

2장 Spark 및 Spark MLlib 소개

클러스터에서 Spark는 작업자 노드에서 실행자를 획득합니다. 그런 다음 Spark는 애플리케이션 코드를 실행자에게 보냅니다. 애플리케이션은 일반적으로 Spark 작업에 대한 응답으로 하나 이상의 작업을 실행합니다. 그런 다음 각 작업은 Spark에 의해 단계 또는 작업의 더 작은 DAG(방향성 비순환 그래프)로 나뉩니다. 그런 다음 각 작업이 분산되어 실행을 위해 작업자 노드 전체의 실행기로 전송됩니다.

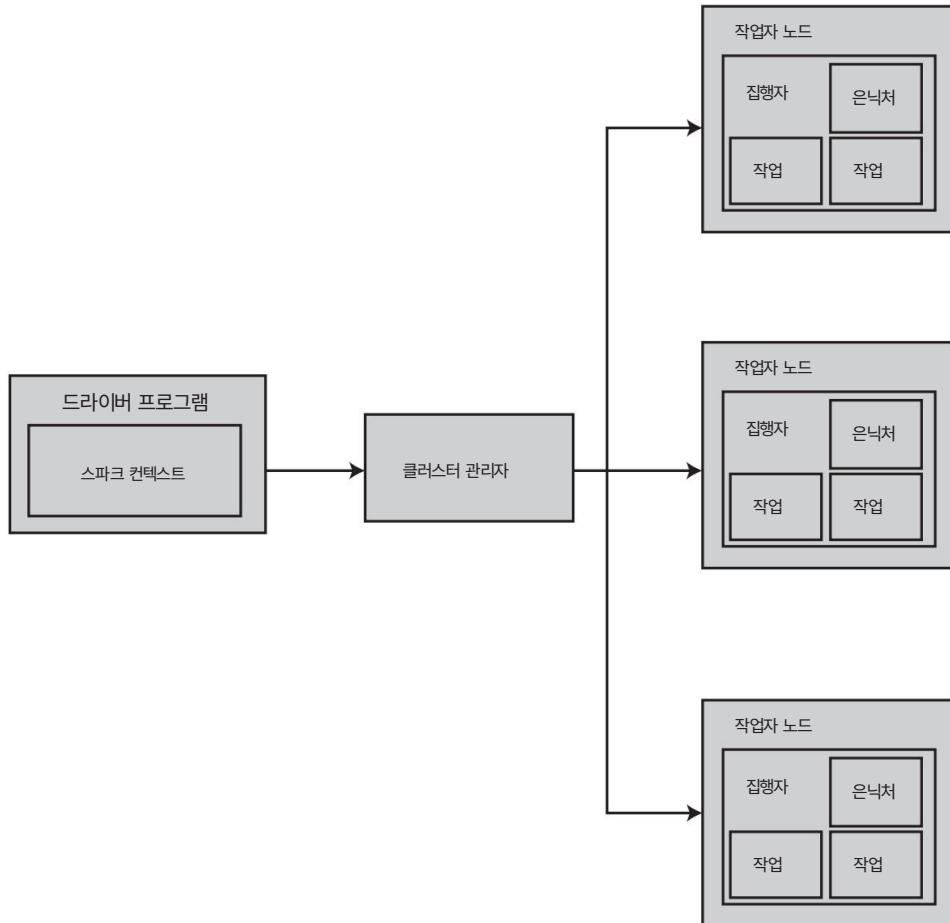


그림 2-2. 아파치 스파크 아키텍처

각 Spark 응용 프로그램은 고유한 실행기 집합을 가져옵니다. 다른 응용 프로그램의 작업이 다른 JVM에서 실행되기 때문에 Spark 응용 프로그램은 다른 Spark 응용 프로그램을 방해할 수 없습니다. 이것은 또한 HDFS 또는 S3와 같은 외부 데이터 소스를 사용하지 않고 Spark 애플리케이션이 데이터를 공유하기 어렵다는 것을 의미합니다. Tachyon(Alluxio라고도 함)과 같은 오프 힙 메모리 저장소를 사용하면 데이터 공유를 더 빠르고 쉽게 만들 수 있습니다. 나는 이 장의 뒷부분에서 Alluxio에 대해 더 자세히 설명합니다.

스파크 애플리케이션 실행

대화형 셸(spark-shell 또는 pyspark)을 사용하거나 응용 프로그램을 제출(spark-submit)하여 Spark 응용 프로그램을 실행합니다. 일부는 Apache Zeppelin 및 Jupyter와 같은 대화형 웹 기반 노트북을 사용하여 Spark 와 상호 작용하는 것을 선호합니다. Databricks 및 Cloudera와 같은 상용 공급업체는 자체 대화형 노트북 환경도 제공합니다. 이 장 전체에서 spark-shell을 사용할 것입니다. YARN과 같은 클러스터 관리자가 있는 환경에서 Spark 애플리케이션을 시작하기 위한 두 가지 배포 모드가 있습니다.

클러스터 모드

클러스터 모드에서 드라이버 프로그램은 YARN이 관리하는 애플리케이션 마스터 내에서 실행됩니다. 클라이언트는 응용 프로그램 실행에 영향을 주지 않고 종료할 수 있습니다. 클러스터 모드에서 애플리케이션 또는 spark-shell을 시작하려면:

```
spark-shell --master yarn --deploy-mode 클러스터
```

```
spark-submit --class mypath.myClass --master 원서 --deploy-mode 클러스터
```

고객 패션

클라이언트 모드에서 드라이버 프로그램은 클라이언트에서 실행됩니다. 애플리케이션 마스터는 YARN에서 리소스를 요청하는 데만 사용됩니다. 클라이언트 모드에서 애플리케이션 또는 spark-shell을 시작하려면:

```
spark-shell --master yarn --deploy-mode 클라이언트
```

```
spark-submit --class mypath.myClass --master 원서 --deploy-mode 클라이언트
```

스파크 셸 소개

일반적으로 임시 데이터 분석 또는 탐색을 위해 대화형 셸을 사용합니다. 또한 Spark API를 배우기에 좋은 도구입니다. Spark의 대화형 셸은 Spark 또는 Python에서 사용할 수 있습니다. 다음 예에서는 도시의 RDD 를 만들고 모두 대문자로 변환합니다. Listing 2-1 과 같이 스파크 셸을 시작하면 "spark"라는 SparkSession 이 자동으로 생성됩니다 .

목록 2-1. 스파크 쉘 소개

불꽃 껍질

Spark 컨텍스트 웹 UI는 <http://10.0.2.15:4041>에서 사용 가능합니다.

Spark 컨텍스트는 'sc'(마스터 = 로컬[*], 앱 ID = 로컬-1574144576837)로 사용 가능합니다.

Spark 세션은 'spark'로 사용할 수 있습니다.

예 오신 것을 환영합니다

```
_____
 / __/______/_/__
 \V`/_\`_/
 /____/._\_,_/_//_/\ 버전 2.4.4
 ///

```

Scala 버전 2.11.12 사용(OpenJDK 64비트 서버 VM, Java 1.8.0_212)

평가할 표현식을 입력합니다.

자세한 내용을 보려면 다음을 입력하십시오.

```
scala>val myCities = sc.parallelize(목록(
    "도쿄",
    "뉴욕",
    "시드니",
    "샌프란시스코"))
```

```
scala>val uCities = myCities.map {x =>x.toUpperCase}
```

```
스칼라>uCities.collect.foreach(println)
```

도쿄

뉴욕

시드니

샌프란시스코

스파크세션

그림 2-2에서 볼 수 있듯이 SparkContext는 모든 Spark 기능에 대한 액세스를 가능하게 합니다. 드라이버 프로그램은 SparkContext를 사용하여 StreamingContext, SQLContext 및 HiveContext와 같은 다른 컨텍스트에 액세스합니다. Spark 2.0부터 SparkSession

2장 Spark 및 Spark MLlib 소개

Spark와 상호 작용할 수 있는 단일 진입점을 제공합니다. Spark 1.x의 `SparkContext`, `SQLContext`, `HiveContext` 및 `StreamingContext`를 통해 사용할 수 있는 모든 기능은 이제 `SparkSession`.`vii`를 통해 액세스할 수 있습니다. Spark 1.x로 작성된 코드는 여전히 발생할 수 있습니다. Spark 1.x에서는 다음과 같이 작성합니다.

```
val sparkConf = new SparkConf().setAppName("MyApp").setMaster("로컬")
val sc = 새로운 SparkContext(sparkConf).set("spark.executor.cores", "4")
val sqlContext = 새로운 org.apache.spark.sql.SQLContext(sc)
```

Spark 2.x에서는 모든 기능이 이미 `SparkSession`에 포함되어 있으므로 `SparkConf`, `SparkContext` 또는 `SQLContext`를 명시적으로 만들 필요가 없습니다.

```
발 스파크 = 스파크세션.
빌더().
앱 이름("마이앱").
config("spark.executor.cores", "4").
getOrCreate()
```

탄력적인 분산 데이터 세트(RDD)

RDD는 클러스터의 하나 이상의 노드에 분할된 개체의 탄력적인 불변 분산 컬렉션입니다. RDD는 변환 및 작업의 두 가지 작업 유형으로 별별로 처리 및 운영될 수 있습니다.

참고 RDD는 Spark 1.x에서 Spark의 기본 프로그래밍 인터페이스였습니다. `Dataset`은 Spark 2.0부터 기본 API로 RDD를 대체했습니다. 사용자는 더 풍부한 프로그래밍 인터페이스와 더 나은 성능으로 인해 RDD에서 `Dataset`/`DataFrame`으로 전환하는 것이 좋습니다. 이 장의 뒷부분에서 `Dataset` 및 `DataFrame`에 대해 설명합니다.

RDD 생성

RDD를 만드는 것은 간단합니다. 기존 Scala 컬렉션에서 또는 HDFS 또는 S3에 저장된 외부 파일을 읽어서 RDD를 생성할 수 있습니다.

병렬화

Parallelize는 Scala 컬렉션에서 RDD를 생성합니다.

```
val 데이터 = (1 ~ 5).toList
val rdd = sc.parallelize(데이터)
val 도시 = sc.parallelize(List("도쿄", "뉴욕", "시드니", "샌프란시스코"))
```

텍스트 파일

TextFile은 HDFS 또는 S3에 저장된 텍스트 파일에서 RDD를 생성합니다.

```
val rdd = sc.textFile("hdfs://master01:9000/파일/mydirectory")
val rdd = sc.textFile("s3a://mybucket/files/mydata.csv")
```

RDD는 변경할 수 없습니다. 데이터 변환은 대신 다른 RDD를 생성합니다.
현재 RDD를 수정합니다. RDD 작업은 변환과 작업의 두 가지 범주로 분류할 수 있습니다.

변환

변환은 새 RDD를 만드는 작업입니다. 가장 일반적인 변형 중 일부를 설명합니다. 전체 목록은 온라인 Spark 설명서를 참조하십시오.

지도

Map은 RDD의 각 요소에 대해 기능을 실행합니다. 결과의 새 RDD를 생성하고 반환합니다. 맵의 반환 유형이 반드시 원래 RDD와 동일한 유형일 필요는 없습니다.

```
val 도시 = sc.parallelize(List("도쿄", "뉴욕", "파리", "샌프란시스코"))
```

```
val upperCaseCities = myCities.map {x => x.toUpperCase}
```

```
upperCaseCities.collect.foreach(println)
```

도쿄

뉴욕

파리

샌프란시스코

2장 Spark 및 Spark MLlib 소개

지도의 다른 예를 보여 드리겠습니다.

```
발 라인 = sc.parallelize(List("마이클 조던", "아이폰"))
```

```
val 단어 = lines.map(line => line.split(" "))
```

단어 수집

```
res2: 배열[배열[문자열]] = 배열(배열(마이클, 조던), 배열(아이폰))
```

플랫맵

FlatMap은 RDD의 각 요소에 대해 함수를 실행한 다음 결과를 평면화합니다.

```
발 라인 = sc.parallelize(List("마이클 조던", "아이폰"))
```

```
val 단어 = lines.flatMap(line => line.split(" "))
```

단어 수집

```
res3: Array[String] = Array(Michael, Jordan, iPhone)
```

필터

필터는 지정된 조건과 일치하는 요소만 포함하는 RDD를 반환합니다.

```
val lines = sc.parallelize(List("마이클 조던", "아이폰", "마이클 끌레오네"))
```

```
val 단어 = lines.map(line => line.split(" "))
```

```
val 결과 = words.filter(w => w.contains("마이클"))
```

결과.수집

```
res9: 배열[배열[문자열]] = Array(배열(마이클, 조던), 배열(마이클,  
사자의 심장))
```

별개의

Distinct는 고유한 값만 반환합니다.

```
val city1 = sc.parallelize(List("도쿄", "도쿄", "파리", "시드니"))
```

```
발 도시2 = sc.parallelize(List("퍼스", "도쿄", "캔버라", "시드니"))
```

```
val 도시3 = 도시1.union(도시2)
```

```
city3.distinct.collect.foreach(println)
```

시드니 퍼
스 캔버라

도쿄 파
리

리듀스바이키

ReduceByKey는 지정된 reduce 함수를 사용하여 동일한 키와 값을 결합합니다.

```
val pairRDD = sc.parallelize(목록(("a", 1), ("b", 2), ("c", 3), ("a", 30), ("b", 25), ("a", 20))) val  
sumRDD = pairRDD.reduceByKey((x,y) =>x+y) sumRDD.collect
```

```
res15: 배열[(문자열, 정수)] = 배열((b,27), (a,51), (c,3))
```

열쇠

Keys는 키만 포함하는 RDD를 반환합니다.

```
val rdd = sc.parallelize(List(("a", "Larry"), ("b", "Curly"), ("c", "Moe")))
```

```
val 키 = rdd.keys
```

```
keys.collect.foreach(println)
```

ㅏ
비
씨

가치

값은 값만 포함하는 RDD를 반환합니다.

```
val rdd = sc.parallelize(List(("a", "Larry"), ("b", "Curly"), ("c", "Moe")))
```

```
val 값 = rdd.values
```

2장 Spark 및 Spark MLlib 소개

```
value.collect.foreach(println)
```

래리

골슬

Moe

내부 조인

내부 조인은 조인 조건자를 기반으로 두 RDD의 모든 요소에 대한 RDD를 반환합니다.

```
val 데이터 = Array((100,"Jim Hernandez"), (101,"Shane King")) val 직원 =  
sc.parallelize(data)
```

```
val data2 = Array((100,"Glendale"), (101,"Burbank")) val 도시 =  
sc.parallelize(data2)
```

```
val data3 = Array((100,"CA"), (101,"CA"), (102,"NY")) val 상태 =  
sc.parallelize(data3)
```

```
val 레코드 =Employees.join(cities).join(states)
```

```
record.collect.foreach(println)
```

(100,((Jim Hernandez,Glendale),CA))

(101,((셰인 킹, 버뱅크), CA))

RightOuterJoin 및 LeftOuterJoin

RightOuterJoin은 왼쪽 RDD에 일치하는 행이 없더라도 오른쪽 RDD에서 요소의 RDD를 반환합니다.

LeftOuterJoin은 열의 순서가 다른 RightOuterJoin과 동일합니다.

```
val 레코드 = employee.join(cities).rightOuterJoin(states)
```

```
record.collect.foreach(println)
```

(100,(Some((Jim Hernandez,Glendale)),CA))

(102, (없음, NY))

(101,(Some((Shane King,Burbank)),CA))

노동 조합

Union은 둘 이상의 RDD 조합을 포함하는 RDD를 반환합니다.

```
val data = Array((103,"Mark Choi","Torrance","CA"),(104,"Janet Reyes","RollingHills","CA"))
val 직원 = sc.parallelize(data) val 데이터 = Array((105,"Lester Cruz","VanNuys","CA"),
(106,"John White","Inglewood","CA")) val Employees2 = sc.parallelize(data) val rdd =
sc.union([직원, 직원2]) rdd.collect.foreach(println)
```

```
(103, MarkChoi, Torrance, CA)
(104,JanetReyes,RollingHills,CA)
(105, Lester Cruz, Van Nuys, CA)
(106,JohnWhite,잉글우드,CA)
```

덜다

빼기는 첫 번째 RDD에 있는 요소만 포함하는 RDD를 반환합니다.

```
val 데이터 = Array((103,"Mark Choi","Torrance","CA"),(104,"Janet
Reyes","Rolling Hills","CA"),(105,"Lester Cruz","Van Nuys","CA"))

val rdd = sc.parallelize(데이터)

val data2 = Array((103,"Mark Choi","Torrance","CA")) val rdd2 =
sc.parallelize(data2)

val 직원 = rdd.subtract(rdd2)

Employees.collect.foreach(println)

(105,LesterCruz,Van Nuys,CA)
(104,JanetReyes,Rolling Hills,CA)
```

2장 Spark 및 Spark MLlib 소개

합체

Coalesce는 RDD의 파티션 수를 줄입니다. 대형 RDD에서 필터를 수행한 후 병합을 사용할 수 있습니다. 필터링은 새 RDD에서 소비하는 데이터의 양을 줄이는 반면 원래 RDD의 파티션 수를 상속합니다. 새 RDD가 원래 RDD보다 훨씬 작은 경우 수백 또는 수천 개의 작은 파티션이 있을 수 있으며 이로 인해 성능 문제가 발생할 수 있습니다.

Coalesce는 HDFS에 쓸 때 Spark에서 생성하는 파일 수를 줄여 두려운 "작은 파일" 문제를 방지하려는 경우에도 유용합니다. 각 파티션은 HDFS에 별도의 파일로 작성됩니다. HDFS에 쓰는 동안 병렬 처리 수준을 효과적으로 줄이기 때문에 병합을 사용할 때 성능 문제가 발생할 수 있습니다. 이 경우 파티션 수를 늘려 보십시오. 다음 예에서는 HDFS에 Parquet 파일을 하나만 쓰고 있습니다.

```
df.coalesce(1).write.mode("추가").parquet("/사용자/하이브/웨어하우스/マイテイブル")
```

분포

재분할은 RDD의 파티션 수를 줄이거나 늘릴 수 있습니다. 재분할보다 효율적이기 때문에 일반적으로 분할을 줄일 때 병합을 사용합니다.

파티션 수를 늘리면 HDFS에 쓸 때 병렬도를 높이는 데 유용합니다. 다음 예에서는 6개의 Parquet 파일을 HDFS에 작성합니다.

```
df.repartition(6).write.mode("추가").parquet("/사용자/하이브/웨어하우스/  
マイテイブル")
```

참고 병합은 일반적으로 다시 분할보다 빠릅니다. 재분할은 전체 셔플을 수행하여 새 파티션을 만들고 작업자 노드 전체에 데이터를 균등하게 배포합니다.

Coalesce는 데이터 이동을 최소화하고 기존 파티션을 사용하여 전체 셔플을 방지합니다.

행위

작업은 드라이버 프로그램에 값을 반환하는 RDD 작업입니다. 나는 가장 일반적인 행동 중 일부를 나열합니다. 전체 작업 목록은 온라인 Spark 설명서를 참조하세요.

수집

수집은 전체 데이터 세트를 배열로 드라이버 프로그램에 반환합니다.

```
val myCities = sc.parallelize(List("tokyo", "new york", "paris", "san francisco")) myCities.collect
res2: Array[String] = Array(도쿄, 뉴욕, 파리, 샌프란시스코)
```

세다

Count는 데이터세트의 요소 수를 반환합니다.

```
val myCities = sc.parallelize(List("tokyo", "new york", "paris", "san francisco")) myCities.count
res3: Long = 4
```

가져 가다

Take는 데이터 세트의 처음 n 개 요소를 배열로 반환합니다.

```
val myCities = sc.parallelize(List("도쿄", "뉴욕", "파리", "샌프란시스코")) myCities.take(2) res4:
Array[String] = Array(도쿄, 뉴욕)
```

각각

Foreach는 데이터 세트의 각 요소에 대해 함수를 실행합니다.

```
val myCities = sc.parallelize(List("도쿄", "뉴욕", "파리", "샌프란시스코"))
```

```
myCities.collect.foreach(println)
```

도쿄 뉴

욕 파리 샌

프란시스코

2장 Spark 및 Spark MLlib 소개

게으른 평가

Spark는 빅 데이터 처리에 중요한 자연 평가를 지원합니다. Spark의 모든 변환은 느리게 평가됩니다.

Spark는 변환을 즉시 실행하지 않습니다. 계속해서 더 많은 변환을 정의할 수 있습니다. 최종 결과를 원할 때 변환이 실행되도록 하는 작업을 실행합니다.

캐싱

기본적으로 작업을 실행할 때마다 각 변환이 다시 실행됩니다. 변환을 여러 번 다시 실행하지 않도록 캐시 또는 지속 방법을 사용하여 메모리에 RDD를 캐시할 수 있습니다.

누산기

누산기는 "추가"되는 변수입니다. 그들은 일반적으로 카운터를 구현하는 데 사용됩니다. 이 예에서는 누산기를 사용하여 배열의 요소를 더합니다.

```
val accum = sc.longAccumulator("누적기 01")
sc.parallelize(배열(10, 20, 30, 40)).foreach(x => accum.add(x))
누적 가치
res2: 긴 = 100
```

브로드캐스트 변수

브로드캐스트 변수는 각 노드의 메모리에 저장된 읽기 전용 변수입니다. Spark는 고속 브로드캐스트 알고리즘을 사용하여 브로드캐스트 변수 복사의 네트워크 대기 시간을 줄입니다. HDFS 또는 S3와 같은 느린 스토리지 엔진에 데이터를 저장하는 대신 브로드캐스트 변수를 사용하는 것이 각 노드에 데이터세트 사본을 저장하는 더 빠른 방법입니다.

```
발 방송Var = sc.broadcast(배열(10, 20, 30))
```

브로드캐스트 변수 값

```
res0: 배열[Int] = 배열(10, 20, 30)
```

Spark SQL, Dataset 및 DataFrames API

Spark SQL은 구조화된 데이터를 보다 쉽게 처리하고 분석할 수 있도록 개발되었습니다.

Dataset은 강력한 유형 지정을 지원한다는 점에서 RDD와 유사하지만 내부적으로 Dataset에는 훨씬 더 효율적인 엔진이 있습니다. Spark 2.0부터 Dataset API는 이제 기본 프로그래밍 인터페이스입니다. DataFrame은 관계형 테이블과 유사한 명명된 열이 있는 Dataset입니다. Spark SQL과 DataFrames는 함께 구조화된 데이터를 처리하고 분석하기 위한 강력한 프로그래밍 인터페이스를 제공합니다. 다음은 DataFrames API를 사용하는 방법에 대한 간단한 예입니다.

```
val jsonDF = spark.read.json("/jsondata/customers.json")
```

```
jsonDF.show
```

나이	도시	이름	주	사용자 ID	지퍼
35	프리스코	조나단	웨스트	텍사스	200 75034
28	달라스	안드레아	포먼	텍사스	201 75001
69	플라노	커스틴	정	텍사스	202 75025
52	알렌	제시카	응우옌	텍사스	203 75002

```
jsonDF.select("나이","도시").show
```

나이	도시
35	프리스코
28	달라스
69	계획
52	알렌

2장 Spark 및 Spark MLlib 소개

```
jsonDF.filter($"사용자 ID" < 202).show()
+---+-----+-----+-----+
|나이| 도시| 이름|주|사용자 ID| 지퍼|
+---+-----+-----+-----+
| 35|프리스코| 조나단 웨스트| 텍사스| 200|75034|
| 28|달라스|안드레아 포먼| 텍사스| 201|75001|
+---+-----+-----+-----+
```

```
jsonDF.createOrReplaceTempView("jsonDF")
```

```
val df = spark.sql("사용자 ID 선택, jsonDF에서 압축")
```

```
df.show
```

```
+-----+
|사용자 ID| 지퍼|
+-----+
| 200|75034|
| 201|75001|
| 202|75025|
| 203|75002|
+-----+
```

참고 DataFrame 및 Dataset API는 Spark 2.0에서 통합되었습니다. DataFrame은 이제 Dataset of Row에 대한 유형 별칭일 뿐입니다. 여기서 Row는 형식이 지정되지 않은 일반 개체입니다. 대조적으로 Dataset은 강력한 형식의 개체 Dataset[T]의 모음입니다. Scala는 강력한 유형의 API와 유형이 지정되지 않은 API를 지원하지만 Java에서는 Dataset[T]가 주요 추상화입니다. DataFrames는 컴파일 시간 형식 안전성에 대한 지원이 부족하기 때문에 R 및 Python의 주요 프로그래밍 인터페이스입니다.

스파크 데이터 소스

다양한 파일 형식 및 데이터 소스에 대한 읽기 및 쓰기는 가장 일반적인 데이터 처리 작업 중 하나입니다. 예제에서는 RDD와 DataFrames API를 모두 사용합니다.

CSV

Spark는 CSV 파일에서 데이터를 읽는 다양한 방법을 제공합니다. 데이터를 읽을 수 있습니다 먼저 RDD로 변환한 다음 DataFrame으로 변환합니다.

```
val dataRDD = sc.textFile("/sparkdata/customerdata.csv")
val parsedRDD = dataRDD.map {_.split(",")}
케이스 클래스 CustomerData(customerid: Int, 이름: 문자열, 도시: 문자열, 상태: 문자열, 우편번호: 문자열)

val dataDF = parsedRDD.map{ a =>CustomerData (a(0).toInt, a(1).toString,
a(2).toString,a(3).toString,a(4).toString) }.toDF
```

Spark 2.0부터 CSV 커넥터가 이미 내장되어 있습니다.

```
val dataDF = spark.read.format("csv")
.option("헤더", "참")
.load("/ sparkdata / customerdata.csv")
```

XML

Databricks에는 XML 데이터를 쉽게 읽을 수 있는 Spark XML 패키지가 있습니다.

고양이 사용자.xml

```
<userid>100</userid><name>Wendell Ryan</name><city>샌디에이고</city> <state>CA</
state><zip>92102</zip>
<userid>101</userid><name>Alicia Thompson</name><city>Berkeley</city> <state>CA</
state><zip>94705</zip>
<userid>102</userid><name>Felipe Drummond</name><city>팔로 알토</city> <state>CA</
state><zip>94301</zip>
<userid>103</userid><name>Teresa Levine</name><city>월넛 크릭</city> <state>CA</
state><zip>94507</zip>
```

하둡 fs -mkdir /xmldata

hadoop fs -put users.xml /xmldata

스파크 쉘 --패키지 com.databricks:spark-xml_2.10:0.4.1

2장 Spark 및 Spark MLlib 소개

Spark XML을 사용하여 DataFrame을 만듭니다. 이 예에서는 XML 파일이 있는 HDFS의 행 태그와 경로를 지정합니다.

com.databricks.spark.xml._ 가져오기

```
val xmlDF = spark.read
    .option("rowTag", "사용자")
    .xml("/xmldata/users.xml");
```

xmlDF: org.apache.spark.sql.DataFrame = [도시: 문자열, 이름: 문자열, 상태: 문자열, 사용자 ID: bigint, 우편번호: bigint]

데이터도 살펴보자.

xmlDF.show

도시	이름	주	사용자 ID	지폐
샌디에이고	웬델 라이언	캘리포니아	100	92102
버클리	앨리시아 톰슨	캘리포니아	101	94705
팔로 알토	펠리페 드러먼드	캘리포니아	102	94301
월넛 크릭	테레사 레빈	캘리포니아	103	94507

JSON

이 예제의 샘플 데이터로 JSON 파일을 생성합니다. 파일이 /jsonodata라는 HDFS의 폴더에 있는지 확인하십시오.

고양이 사용자.json

```
{"userid": 200, "name": "Jonathan West", "city": "Frisco", "state": "TX", "zip": "75034", "age": 35}
```

```
{"userid": 201, "name": "Andrea Foreman", "city": "Dallas", "state": "TX", "zip": "75001", "age": 28}
```

```
{"userid": 202, "name": "Kirsten Jung", "city": "Plano", "state": "TX", "zip": "75025", "age": 69}
```

```
{"userid": 203, "name": "Jessica Nguyen", "city": "Allen", "state": "TX", "zip": "75002", "age": 52}
```

JSON 파일에서 DataFrame을 만듭니다.

```
val jsonDF = spark.read.json("/jsondata/users.json")
```

jsonDF: org.apache.spark.sql.DataFrame = [나이: bigint, 도시: 문자열, 이름: 문자열, 주: 문자열, 사용자 ID: bigint, 우편번호: 문자열]

데이터를 확인하십시오.

```
jsonDF.show
```

나이	도시	이름	주	사용자 ID	지퍼
35	프리스코	조나단 웨스트	텍사스	200	75034
28	달라스	안드레아 포먼	텍사스	201	75001
69	플라노	커스틴 정	텍사스	202	75025
52	알렌	제시카 응우옌	텍사스	203	75002

관계형 및 MPP 데이터베이스

이 예에서는 MySQL을 사용하지만 Oracle, Snowflake, Redshift, Impala, Presto 및 Azure DW와 같은 다른 관계형 데이터베이스 및 MPP 엔진도 지원됩니다.

일반적으로 관계형 데이터베이스에 JDBC 드라이버가 있는 한 Spark에서 액세스할 수 있어야 합니다. 성능은 일괄 작업에 대한 JDBC 드라이버의 지원에 따라 다릅니다. 자세한 내용은 JDBC 드라이버 설명서를 확인하십시오.

```
mysql -u 루트 -pmypassword
```

데이터베이스 생성 salesdb;

salesdb를 사용하십시오.

```
테이블 생성 고객( customerid
INT, 이름 VARCHAR(100), 도시
VARCHAR(100), 상태 CHAR(3),
우편 번호 CHAR(5));
```

스파크 쉘 --드라이버 클래스 경로 mysql-connector-java-5.1.40-bin.jar

2장 Spark 및 Spark MLlib 소개

스파크 쉘을 시작하십시오.

CSV 파일을 RDD로 읽어서 DataFrame으로 변환합니다.

```
val dataRDD = sc.textFile("/home/hadoop/test.csv") val parsedRDD
= dataRDD.map {_.split(",")}
```

케이스 클래스 CustomerData(customerid: Int, 이름: 문자열, 도시: 문자열, 상태: 문자열, 우편번호: 문자열)

```
val dataDF = parsedRDD.map{ a =>CustomerData (a(0).toInt, a(1).toString,
a(2).toString,a(3).toString,a(4).toString) }.toDF
```

데이터 프레임을 임시 테이블로 등록하여 이에 대해 SQL 쿼리를 실행할 수 있습니다.

```
dataDF.createOrReplaceTempView("dataDF")
```

연결 속성을 설정해 보겠습니다.

```
val jdbcUsername = "myuser" val
jdbcPassword = "mypass" val
jdbcHostname = "10.0.1.112" val jdbcPort
= 3306 val jdbcDatabase ="salesdb" val
jdbcrewriteBatchedStatements = "true"
val jdbc${name:mysql }:${jdbcPort}/${jdbcDatabase}?us
er=${jdbcUsername}&password=${jdbcPassword}&rewriteBatchedStatements=${jdbc
rewriteBatchedStatements}"
```

```
val connectionProperties = 새로운 java.util.Properties()
```

이를 통해 추가, 덮어쓰기 등의 올바른 저장 모드를 지정할 수 있습니다.

```
org.apache.spark.sql.SaveMode 가져오기
```

SELECT 문에서 반환된 데이터를 MySQL salesdb 데이터베이스에 저장된 고객 테이블에 삽입합니다.

```
spark.sql("dataDF에서 * 선택")
.c쓰다
```

```
.mode(SaveMode.Append) .jdbc(jdbcUrl, "고객", 연결 속성)
```

2장 Spark 및 Spark MLlib 소개

JDBC를 사용하여 테이블을 읽어봅시다. 몇 가지 테스트 데이터로 MySQL의 users 테이블을 채우자. users 테이블이 salesdb 데이터베이스에 있는지 확인하십시오.

```
mysql -u 루트 -pmypassword
```

salesdb를 사용하십시오.

사용자를 설명합니다.

필드	유형	널	키	기본값	추가
사용자 아이디	빅틴트(20)	예		이름	없는
varchar(100)	예		도시	varchar(100)	예
상태	문자(3)				없는
				예 지퍼	없는
문자(5)				예 나이	없는
초소형(4)	예				없는

사용자 중에서 *를 선택하십시오.

빈 세트(0.00초)

사용자 값에 삽입(300,'Fred Stevens','Torrance','CA',90503,23);

사용자 값에 삽입(301,'Nancy Gibbs','Valencia','CA',91354,49);

사용자 값에 삽입(302,'Randy Park','Manhattan Beach','CA',90267,21);

사용자 값에 삽입(303,'Victoria Loma','Rolling Hills','CA',90274,75);

사용자 중에서 *를 선택하십시오.

사용자 아이디	이름	도시	상태	지퍼	나이
300	프레드 스티븐스	토런스		저것	90503 23
301	낸시 깁스	발렌시아		저것	91354 49
302	랜디 파크	맨해튼 비치	맨해튼 비치	캘리포니아	90267 21
303	빅토리아 로마	롤링 힐스	빅토리아 로마	캘리포니아.	90274 75

```
spark-shell --driver-class-path mysql-connector-java-5.1.40-bin.jar --jars mysql-connector-java-5.1.40-bin.jar
```

2장 Spark 및 Spark MLlib 소개

jdbcurl 및 연결 속성을 설정해 보겠습니다.

```
val jdbcURL = s"jdbc:mysql://10.0.1.101:3306/salesdb?user=myuser&password= mypass"
```

```
val connectionProperties = 새로운 java.util.Properties()
```

전체 테이블에서 DataFrame을 만들 수 있습니다.

```
val df = spark.read.jdbc(jdbcURL, "사용자", 연결 속성)
```

```
df.show
```

사용자 ID	이름	시 주 우편번호 나이
300 프레드 스티븐스	토랜스 CA 90503 23	
301 낸시 깁스	발렌시아 CA 91354 49	
302 랜디 파크 맨해튼 비치 CA 90267 21		
303 빅토리아 홀리데이 롤링 힐스 캘리포니아 90274 75		

쪽매 세공

Parquet에 읽고 쓰는 것은 간단합니다.

```
val df = spark.read.load("/ sparkdata/Employees.parquet")
df.select("아이디", "이름", "성", "급여") .write
    .format("parquet") .save("/")
    sparkdata / myData.parquet")
```

Parquet 파일에서 직접 SELECT 문을 실행할 수 있습니다.

```
val df = spark.sql("SELECT * FROM parquet.` /sparkdata/myData.parquet` ")
```

HBase

Spark에서 HBase에 액세스하는 방법에는 여러 가지가 있습니다. 예를 들어 SaveAsHadoopDataset을 사용하여 HBase에 데이터를 쓸 수 있습니다. HBase 셀을 시작합니다.

HBase 테이블을 만들고 테스트 데이터로 채웁니다.

hbase 쉘

'사용자', 'cf1' 생성

스파크 쉘을 시작하십시오.

불꽃 껍질

```
val hconf = HBaseConfiguration.create() val
jobConf = new JobConf(hconf, this.getClass)
jobConf.setOutputFormat(classOf[TableOutputFormat])
jobConf.set(TableOutputFormat.OUTPUT_TABLE, "사용자")

값 번호 = sc.parallelize(목록(1,2,3,4,5,6))

val theRDD = num.filter.map(x=>{
    val rowkey = "행" + x
    val put = new Put(Bytes.toBytes(rowkey))
    put.add(Bytes.toBytes("cf1"), Bytes.toBytes("fname"), Bytes.toBytes("my
    fname" + x))
    (new ImmutableBytesWritable, 넣어)
})
theRDD.saveAsHadoop 데이터셋(jobConf)
```

Spark의 HBase 클라이언트 API를 사용하여 HBase에 데이터를 읽고 쓸 수도 있습니다. 같이 앞서 논의한 것처럼 Scala는 모든 Java 라이브러리에 액세스할 수 있습니다.

HBase 쉘을 시작합니다. 다른 HBase 테이블을 만들고 테스트 데이터로 채웁니다.

hbase 쉘

'직원', 'cf1' 생성

```
put 'employees','400','cf1:name', 'Patrick Montalban' put
'employees','400','cf1:city', 'Los Angeles' put 'employees','400','cf1:
state', 'CA'는 '직원','400','cf1:zip', '90010'은 '직원','400','cf1:age', '71'을
입력합니다.
```

2장 Spark 및 Spark MLlib 소개

'직원', '401', 'cf1:name', 'Jillian Collins'를 입력하세요.

'직원', '401', 'cf1:city', 'Santa Monica'를 입력합니다.

'직원', '401', 'cf1:state', 'CA' 입력

'직원', '401', 'cf1:zip', '90402'를 입력하세요.

'직원', '401', 'cf1:나이', '45'를 입력하세요.

'직원', '402', 'cf1:name', 'Robert Sarkisian'를 입력하세요.

'직원', '402', 'cf1:city', 'Glendale' 입력

'직원', '402', 'cf1:state', 'CA'를 입력하세요.

'직원', '402', 'cf1:zip', '91204'를 입력하세요.

'직원', '402', 'cf1:나이', '29'를 입력하세요.

'직원', '403', 'cf1:name', 'Warren Porcaro'를 입력하세요.

'직원', '403', 'cf1:city', 'Burbank'를 입력합니다.

'직원', '403', 'cf1:state', 'CA'를 입력하세요.

'직원', '403', 'cf1:zip', '91523'를 입력하세요.

'직원', '403', 'cf1:나이', '62'를 입력하세요.

데이터가 HBase 테이블에 성공적으로 삽입되었는지 확인하겠습니다.

'직원' 스캔

열	컬럼+셀
400	column=cf1:나이, 타임스탬프=1493105325812, 값=71
400	column=cf1:도시, 타임스탬프=1493105325691, 값=로스앤젤레스
400	column=cf1:이름, 타임스탬프=1493105325644, 값=패트릭 몬탈반
400	열=cf1:상태, 타임스탬프=1493105325738, 값=CA
400	column=cf1:zip, 타임스탬프=1493105325789, 값=90010
401	column=cf1:나이, 타임스탬프=1493105334417, 값=45
401	column=cf1:도시, 타임스탬프=1493105333126, 값=산타모니카
401	column=cf1:이름, 타임스탬프=1493105333050, 값=질리언 콜린스
401	열=cf1:상태, 타임스탬프=1493105333145, 값=CA
401	column=cf1:zip, 타임스탬프=1493105333165, 값=90402
402	column=cf1:나이, 타임스탬프=1493105346254, 값=29
402	column=cf1:도시, 타임스탬프=1493105345053, 값=Glendale
402	column=cf1:이름, 타임스탬프=1493105344979, 값=Robert Sarkisian
402	열=cf1:상태, 타임스탬프=1493105345074, 값=CA

2장 Spark 및 Spark MLlib 소개

```

402     column=cf1:zip, timestamp=1493105345093, value=91204
403     column=cf1:age, timestamp=1493105353650, value=62 column=cf1:city,
403     timestamp=1493105352467, value=Burbank column=cf1=5, value=Burbank
403     column=cf1=5, value=Burbank column=cf1=5 =Warren Porcaro 열=cf1:상태, 타임스탬프
403     =1493105352513, 값=CA 열=cf1:zip, 타임스탬프=1493105352549, 값=91523
403

```

스파크 쉘을 시작하십시오.

불꽃 껍질

```

가져오기 org.apache.hadoop.fs.Path;
org.apache.hadoop.hbase 가져오기{HBaseConfiguration, HTableDescriptor} 가져오기|
org.apache.hadoop.hbase.client.HBaseAdmin 가져오기
org.apache.hadoop.hbase.mapreduce.TableInputFormat 가져오기
org.apache.hadoop.hbase.HColumnDescriptor 가져오기 조직 .apache.hadoop.hbase.client.Put;
가져오기 org.apache.hadoop.hbase.client.Get; org.apache.hadoop.hbase.client.HTable 가져오기;
가져오기 org.apache.hadoop.conf.Configuration; org.apache.hadoop.hbase.client.Result 가져오
기; org.apache.hadoop.hbase.util.Bytes 가져오기; 가져오기 java.io.IOException;

```

```
val 구성 = HBaseConfiguration.create()
```

HBase 테이블과 rowkey를 지정합니다.

```
val 테이블 = new HTable(구성, "직원"); val g = new Get(Bytes.toBytes("401"))
val 결과 = 테이블.get(g);
```

테이블에서 값을 추출합니다.

```
val val2 = 결과.getValue(Bytes.toBytes("cf1"), Bytes.toBytes("이름")); val val3 =
    결과.getValue(Bytes.toBytes("cf1"), Bytes.toBytes("도시")); val val4 =
    결과.getValue(Bytes.toBytes("cf1"), Bytes.toBytes("상태")); val val5 =
    결과.getValue(Bytes.toBytes("cf1"), Bytes.toBytes("zip")); val val6 =
    결과.getValue(Bytes.toBytes("cf1"), Bytes.toBytes("나이"));
```

2장 Spark 및 Spark MLlib 소개

값을 적절한 데이터 유형으로 변환합니다.

```
유효 ID = Bytes.toString(결과.getRow())
값 이름 = Bytes.toString(val2);
발 도시 = Bytes.toString(val3);
발 상태 = Bytes.toString(val4);
발 zip = Bytes.toString(val5);
유효 기간 = Bytes.toShort(val6);
```

값을 인쇄합니다.

```
println(" 사원 ID: 주: + 주 + 우편 번호: " + 아이디 + " 이름: " + 이름 + " 도시: " + 도시 + " +
" + 우편번호 + " 나이: " + 나이);
```

직원 ID: 401 이름: Jillian Collins 시: 산타모니카 주: CA 우편번호: 90402 나이: 13365

HBase API를 사용하여 HBase에 작성해 보겠습니다.

```
val 구성 = HBaseConfiguration.create()
val 테이블 = new HTable(구성, "직원");
```

새 행 키를 지정하십시오.

```
val p = new Put(new String("404").getBytes());
```

새 값으로 셀을 채웁니다.

```
p.add("cf1".getBytes(), "이름".getBytes(), new String("Denise Shulman").
getBytes());
p.add("cf1".getBytes(), "city".getBytes(), new String("라호야").
getBytes());
p.add("cf1".getBytes(), "상태".getBytes(), new String("CA").getBytes());
p.add("cf1".getBytes(), "zip".getBytes(), new String("92093").getBytes());
p.add("cf1".getBytes(), "나이".getBytes(), new String("56").getBytes());
```

HBase 테이블에 씁니다.

```
table.put(p);
테이블.닫기();
```

값이 HBase 테이블에 성공적으로 삽입되었는지 확인합니다.

HBase 셸을 시작합니다.

hbase 셸

'직원' 스캔

열	컬럼+셀
400	column=cf1:나이, 타임스탬프=1493105325812, 값=71
400	column=cf1:도시, 타임스탬프=1493105325691, 값=로스앤젤레스
400	column=cf1:이름, 타임스탬프=1493105325644, 값=패트릭 몬탈반
400	열=cf1:상태, 타임스탬프=1493105325738, 값=CA
400	column=cf1:zip, 타임스탬프=1493105325789, 값=90010
401	column=cf1:나이, 타임스탬프=1493105334417, 값=45
401	column=cf1:도시, 타임스탬프=1493105333126, 값=산타모니카
401	column=cf1:이름, 타임스탬프=1493105333050, 값=질리언 콜린스
401	열=cf1:상태, 타임스탬프=1493105333145, 값=CA
401	column=cf1:zip, 타임스탬프=1493105333165, 값=90402
402	column=cf1:나이, 타임스탬프=1493105346254, 값=29
402	column=cf1:도시, 타임스탬프=1493105345053, 값=Glendale
402	column=cf1:이름, 타임스탬프=1493105344979, 값=Robert Sarkisian
402	열=cf1:상태, 타임스탬프=1493105345074, 값=CA
402	column=cf1:zip, 타임스탬프=1493105345093, 값=91204
403	column=cf1:나이, 타임스탬프=1493105353650, 값=62
403	column=cf1:도시, 타임스탬프=1493105352467, 값=바뱅크
403	column=cf1:이름, 타임스탬프=1493105352445, 값=워렌 포카로
403	열=cf1:상태, 타임스탬프=1493105352513, 값=CA
403	column=cf1:zip, 타임스탬프=1493105352549, 값=91523
404	column=cf1:나이, 타임스탬프=1493123890714, 값=56
404	column=cf1:도시, 타임스탬프=1493123890714, 값=라호야
404	column=cf1:이름, 타임스탬프=1493123890714, 값=Denise Shulman
404	열=cf1:상태, 타임스탬프=1493123890714, 값=CA
404	column=cf1:zip, 타임스탬프=1493123890714, 값=92093

일반적으로 느리지만 Impala 또는 Presto와 같은 SQL 쿼리 엔진을 통해 HBase에 액세스할 수도 있습니다.

2장 Spark 및 Spark MLlib 소개

아마존 S3

Amazon S3는 임시 클러스터의 데이터 저장소로 자주 사용되는 인기 있는 객체 저장소입니다.

또한 백업 및 골드 데이터를 위한 비용 효율적인 스토리지입니다. S3에서 데이터를 읽는 것은 HDFS 또는 다른 파일 시스템에서 데이터를 읽는 것과 같습니다.

Amazon S3에서 CSV 파일을 읽습니다. S3 자격 증명을 구성했는지 확인합니다.

```
val myCSV = sc.textFile("s3a://mydata/customers.csv")
```

CSV 데이터를 RDD에 매핑합니다.

```
org.apache.spark.sql.Row 가져오기
```

```
val myRDD = myCSV.map(_.split(',')).map(e => 행(r(0).trim.toInt, r(1), r(2).trim.toInt, r(3)))
```

スキ마를 생성합니다.

```
import org.apache.spark.sql.types.{StructType, StructField, StringType, IntegerType};
```

```
val mySchema = StructType(배열(
  StructField("customerid", IntegerType, false),
  StructField("고객 이름", StringType, false),
  StructField("나이", IntegerType, false),
  StructField("도시", StringType, false)))
```

```
val myDF = spark.createDataFrame(myRDD, mySchema)
```

솔라

SolrJ.viii를 사용하여 Spark에서 Solr와 상호 작용할 수 있습니다.

```
가져오기 java.net.MalformedURLException; 가져오기
org.apache.solr.client.solrj.SolrServerException;
org.apache.solr.client.solrj.impl.HttpSolrServer 가져오기;
org.apache.solr.client.solrj.SolrQuery 가져오기;
org.apache.solr.client.solrj.response.QueryResponse 가져오기;
org.apache.solr.common.SolrDocumentList 가져오기;
```

```
val solr = 새로운 HttpSolrServer("http://master02:8983/solr/mycollection");
```

```
val 쿼리 = 새로운 SolrQuery();
```

```
query.setQuery("*:*");
```

```
query.addFilterQuery("사용자 ID:3");
```

```
query.setFields("사용자 ID", "이름", "나이", "도시"); 쿼리.setStart(0); query.set("defType", "edismax");
```

```
발 응답 = solr.query(쿼리); 값 결과 = 응
```

```
답.getResults();
```

```
println(결과);
```

Spark에서 Solr 컬렉션에 액세스하는 훨씬 쉬운 방법은 spark-solr 패키지를 사용하는 것입니다.

Lucidworks는 Spark-Solr 통합을 제공하기 위해 spark-solr 프로젝트를 시작했습니다. ix spark-solr를 사용하는 것은 SolrJ에 비해 훨씬 쉽고 강력하므로 Solr 컬렉션에서 DataFrame을 생성할 수 있습니다.

먼저 spark-shell에서 JAR 파일을 가져옵니다.

```
spark-shell --jars spark-solr-3.0.1-shaded.jar
```

컬렉션 및 연결 정보를 지정합니다.

```
val 옵션 = Map( "컬렉션" -> "mycollection", "zkhost" -> "{ master02:8983/solr}" )
```

데이터프레임을 생성합니다.

```
val solrDF =
```

```
    spark.read.format("solr") .options(옵션) .load
```

マイ크로 소프트 엑셀

Spark에서 Excel 스프레드시트에 액세스하는 것은 일반적으로 권장하지 않는 기능이지만 특정 사용 사례에는 해당 기능이 필요합니다. Crealytics라는 회사는 Excel과 상호 작용하기 위한 Spark 플러그인을 개발했습니다. 라이브러리에는 Spark 2.x가 필요합니다. --packages 명령줄 옵션을 사용하여 패키지를 추가할 수 있습니다.

```
spark-shell --packages com.crealytics:spark-excel_2.11:0.9.12
```

2장 Spark 및 Spark MLlib 소개

Excel 워크시트에서 DataFrame을 만듭니다.

```
val ExcelDF = spark.read
    .format("com.crealytics.spark.excel") .option("sheetName",
    "sheet1") .option("useHeader",
    "true") .option("inferSchema", "true") .option(
    "TreatEmptyValuesAsNulls", "true") .load("예
    산.xlsx")
```

Excel 워크시트에 DataFrame을 씁니다.

```
엑셀DF2.write
    .format("com.crealytics.spark.excel") .option("sheetName",
    "sheet1") .option("useHeader", "true") .mode("덮
    어쓰기") .save("예산2.xlsx" )
```

자세한 내용은 GitHub 페이지(github.com/crealytics)에서 확인할 수 있습니다.

보안 FTP

SFTP 서버에서 파일을 다운로드하고 SFTP 서버에 DataFrame을 쓰는 것도 인기 있는 요청입니다. SpringML은 Spark SFTP 커넥터 라이브러리를 제공합니다. 라이브러리에는 Spark 2.x가 필요하며 SSH2의 Java 구현인 jsch를 활용합니다. SFTP 서버에서 읽고 쓰는 것은 단일 프로세스로 실행됩니다.

스파크 쉘 --패키지 com.springml:spark-sftp_2.11:1.1.

SFTP 서버의 파일에서 DataFrame을 만듭니다.

```
불 sftpDF = spark.read. 형식
    ("com.springml.spark.sftp"). 옵션("호스트",
    "sftpserver.com"). 옵션("사용자 이름", "내 사용자
    이름"). 옵션("비밀번호", "마이패스워드"). 옵션
    ("inferSchema", "true"). 옵션("파일 형식", "csv").
```

2장 Spark 및 Spark MLlib 소개

```
옵션("구분자", ","). 로드("/myftp/
myfile.csv")
```

DataFrame을 CSV 파일로 FTP 서버에 씁니다.

sftpDF2.write.

```
형식("com.springml.spark.sftp"). 옵션
("호스트", "sftpserver.com"). 옵션("사용
자 이름", "내 사용자 이름"). 옵션("비밀번
호", "마이패스워드"). 옵션("파일 형식",
"csv"). 옵션("구분자", ","). 저장("/myftp/
myfile.csv")
```

GitHub 페이지: github.com/springml/spark-sftp에서 자세한 내용을 찾을 수 있습니다.

스파크 MLlib 소개

기계 학습은 Spark의 주요 응용 프로그램 중 하나입니다. Spark MLlib에는 회귀, 분류, 클러스터링, 협업 필터링 및 빈번한 패턴 마이닝을 위한 인기 있는 기계 학습 알고리즘이 포함되어 있습니다. 또한 파이프라인 구축, 모델 선택 및 조정, 기능 선택, 추출 및 변환을 위한 다양한 기능을 제공합니다.

Spark MLlib 알고리즘

Spark MLlib에는 다양한 작업을 위한 수많은 기계 학습 알고리즘이 포함되어 있습니다. 다음 장에서 대부분을 다룹니다.

분류

- 로지스틱 회귀(이항 및 다항)
- 의사결정 트리
- 랜덤 포레스트
- 그래디언트 부스트 트리
- 다중 퍼셉트론

2장 Spark 및 Spark MLlib 소개

- 선형 서포트 벡터 머신
- 나이브 베이즈
- 일대일 휴식

회귀

- 선형 회귀
- 의사결정 트리
- 랜덤 포레스트
- 그래디언트 부스트 트리
- 생존 회귀
- 등장 회귀

클러스터링

- K-평균
- K-평균 이등분
- 가우스 혼합 모델
- 잠재 디리클레 할당(LDA)

협업 필터링

- ALS(교대 최소 제곱)

빈번한 패턴 마이닝

- FP-성장
- 접두사 스판

ML 파이프라인

Spark MLlib의 초기 버전에는 RDD 기반 API만 포함되었습니다. DataFrame 기반 API는 이제 Spark의 기본 API입니다. RDD 기반 API는 DataFrames 기반 API가 가능 parity.x에 도달하면 Spark 2.3에서 더 이상 사용되지 않습니다. RDD 기반 API는 Spark 3.0에서 제거됩니다. DataFrames 기반 API를 사용하면 관계형 데이터베이스 테이블과 유사한 테이블 형식 데이터를 나타내는 상위 수준 추상화를 제공하여 기능을 쉽게 변환할 수 있으므로 파이프라인 구현을 위한 자연스러운 선택이 됩니다.

Spark MLlib API는 기계 학습을 만들기 위한 몇 가지 개념을 소개합니다. 파일파이. 그림 2-3 은 텍스트 데이터 처리를 위한 간단한 Spark MLlib 파일파이를 보여줍니다. 토크나이저는 텍스트를 단어 모음으로 나누고 단어를 출력 DataFrame에 추가합니다. TF-IDF(Term Frequency-Inverse Document Frequency)는 DataFrame을 입력으로 받아 단어 모음을 특징 벡터로 변환하고 세 번째 DataFrame에 추가합니다.

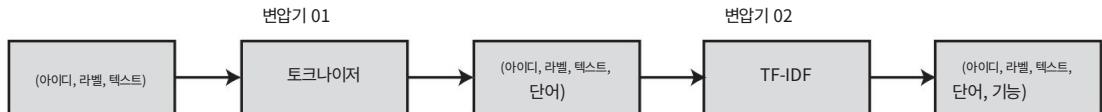


그림 2-3. 간단한 Spark MLlib 파일파이

관로

파일파이은 기계 학습 워크플로를 생성하기 위한 일련의 연결된 단계입니다. 단계는 변환기 또는 추정기가 될 수 있습니다.

변신 로봇

변환기는 DataFrame을 입력으로 사용하고 새 DataFrame에 추가 열이 추가된 새 DataFrame을 출력합니다. 새 DataFrame에는 입력 DataFrame의 열과 추가 열이 포함됩니다.

평가자

추정기는 훈련 데이터에 모델을 맞추는 기계 학습 알고리즘입니다. Estimators는 훈련 데이터를 받아들이고 기계 학습 모델을 생성합니다.

ParamGridBuilder

ParamGridBuilder는 매개변수 그리드를 작성하는 데 사용됩니다. CrossValidator는 그리드 검색을 수행하고 매개변수 그리드에서 사용자 지정 하이퍼파라미터의 조합으로 모델을 훈련합니다.

교차검증기

CrossValidator는 적합한 기계 학습 모델을 교차 평가하고 기본 추정기를 사용자가 지정한 하이퍼파라미터 조합으로 맞추려고 시도하여 최상의 모델을 출력합니다. 모델 선택은 CrossValidator 또는 TrainValidationSplit 추정기로 수행됩니다.

평가자

평가자는 기계 학습 모델의 성능을 계산합니다. 피팅된 모델이 얼마나 잘 수행되는지 측정하기 위해 정밀도 및 재현율과 같은 메트릭을 출력합니다. 평가자의 예로는 이진 및 다중 클래스 분류 작업에 대해 각각 BinaryClassificationEvaluator 및 MulticlassClassificationEvaluator가 있고 회귀 작업에 대해 RegressionEvaluator가 있습니다.

특징 추출, 변환 및 선택

대부분의 경우 원시 데이터를 사용하여 모델을 맞추기 전에 추가 전처리가 필요합니다. 예를 들어 거리 기반 알고리즘은 기능을 표준화해야 합니다.

일부 알고리즘은 범주형 데이터가 원-핫 인코딩될 때 더 잘 수행됩니다. 텍스트 데이터에는 일반적으로 토큰화 및 가능한 벡터화가 필요합니다. 매우 큰 데이터 세트의 경우 차원 축소가 필요할 수 있습니다. Spark MLlib에는 이러한 유형의 작업에 대한 광범위한 변환기 및 추정기 컬렉션이 포함되어 있습니다. Spark MLlib에서 사용할 수 있는 가장 일반적으로 사용되는 변환기와 추정기에 대해 논의하겠습니다.

문자열 인덱서

대부분의 기계 학습 알고리즘은 문자열을 직접 사용할 수 없으며 데이터가 숫자 형식이어야 합니다.

StringIndexer는 레이블의 문자열 열을 인덱스로 변환하는 추정기입니다. 인덱스를 생성하는 네 가지 방법(alphabetDesc, alphabetAsc, frequencyDesc 및 frequencyAsc)을 지원합니다. 기본값은 frequencyDesc로 설정되며 가장 빈번한 레이블은 0으로 설정되며 결과는 레이블 빈도의 내림차순으로 정렬됩니다.

org.apache.spark.ml.feature.StringIndexer 가져오기

```
발 df = spark.createDataFrame(  
Seq((0, "자동차"), (1, "자동차"), (2, "트럭"), (3, "밴"), (4, "밴"), (5, "밴")) ).toDF("아이디",  
"클래스")
```

df.show

```
+---+-----+  
| 아이디|클래스|  
+---+-----+  
| † 0| 차| † 1|  
| 차| † 2|트럭|  
| † 3| 에서| †  
| 4| 에서| † 5|  
| 에서|
```

+---+-----+

val 모델 = 새로운

```
StringIndexer() .setInputCol("클  
래스") .setOutputCol("classIndex")
```

발 인덱서 = model.fit(df)

val indexed = indexer.transform(df)

indexed.show()

```
+---+-----+  
| 아이디|클래스|클래스 인덱스|  
+---+-----+  
| † 0| 차| † 1| 1.0|  
| 차| † 2|트럭| 1.0|  
| † 3| 에서| † 2| 2.0|  
| 4| 에서| † 5| 0.0|  
| 에서| 0.0|  
| 0.0|  
+---+-----+
```

2장 Spark 및 Spark MLlib 소개

토크나이저

텍스트 데이터를 분석할 때 일반적으로 문장을 개별 용어나 단어로 분할하는 것이 필수적입니다. 토크나이저는 정확히 그 일을 합니다. RegexTokenizer를 사용하여 정규식을 사용하여 고급 토큰화를 수행할 수 있습니다. 토큰화는 일반적으로 기계 학습 NLP 파이프라인의 첫 번째 단계 중 하나입니다. 자연어 처리(NLP)에 대해서는 4장에서 더 자세히 설명하겠습니다.

`org.apache.spark.ml.feature.Tokenizer 가져오기`

```
val df = spark.createDataFrame(Seq(
  (0, "마크는 어젯밤 라구나 비치에서 연설했습니다"),
  (1, "오렌지는 영양소가 풍부하고 칼로리가 낮습니다"),
  (2, "에디 반 헤일런은 대단하다")
)).toDF("아이디", "문장")

df.show(거짓)
```

아이디	문장
0	마크는 어젯밤 라구나 비치에서 연설을 했다
1	오렌지는 영양이 풍부하고 칼로리가 낮습니다
2	에디 반 헤일런은 훌륭하다

```
val 토크나이저 = new Tokenizer().setInputCol("문장").
  setOutputCol("단어")
```

토큰화된 val = tokenizer.transform(df)

tokenized.show(거짓)

아이디	문장
0	마크는 어젯밤 라구나 비치에서 연설을 했다
1	오렌지는 영양이 풍부하고 칼로리가 낮습니다
2	에디 반 헤일런은 훌륭하다

2장 Spark 및 Spark MLlib 소개

```
+-----+
|단어|
+-----+
|[마크, 준, 연설, 마지막, 밤, 인, 라구나, 해변] |
|[오렌지, ~는, 가득, ~, ~로, ~으로, ~으로, 칼로리]|
|[에디, 밴, 겟, 광장하다] |
+-----+
```

벡터어셈블러

Spark MLlib 알고리즘을 사용하려면 단일 벡터 열에 기능을 저장해야 합니다.

일반적으로 훈련 데이터는 데이터가 별도의 열에 저장되는 표 형식으로 제공됩니다. VectorAssembler는 열 집합을 단일 벡터 열로 병합하는 변환기입니다.

`org.apache.spark.ml.feature.VectorAssembler 가져오기`

```
발 df = spark.createDataFrame(
    시퀀스((0, 50000, 7, 1))
).toDF("ID", "수입", "고용 기간", "결혼 상태")

val 어셈블러 = 새로운 VectorAssembler()
.setInputCols(Array("수입", "고용 기간", "결혼 상태"))
.setOutputCol("기능")

발 df2 = 어셈블러.변환(df)

df2.show(거짓)
```

```
+---+-----+-----+-----+
|id |소득|고용 기간|결혼 상태|특징
+---+-----+-----+-----+
|0 |50000 |7           |1           |[50000.0,7.0,1.0]
+---+-----+-----+-----+
```

2장 Spark 및 Spark MLlib 소개

스탠다드스케일러

1 장에서 논의한 것처럼 일부 기계 학습 알고리즘은 기능이 제대로 작동하도록 정규화되어야 합니다.

StandardScaler는 단위 표준 편차 및/또는 0 평균을 갖도록 기능을 정규화하는 추정기입니다. withStd 및 withMean의 두 매개변수를 허용합니다. withStd는 기능을 단위 표준 편차로 확장합니다. 이 매개변수는 기본적으로 true로 설정됩니다. withMean을 true로 설정하면 스케일링 전에 데이터의 중앙에 평균이 표시됩니다.

이 매개변수는 기본적으로 false로 설정됩니다.

```
org.apache.spark.ml.feature.StandardScaler 가져오기
org.apache.spark.ml.feature.VectorAssembler 가져오기
```

```
발 df = spark.createDataFrame(
    시퀀스((0, 186, 200, 56),(1, 170, 198, 42))
).toDF("아이디", "키", "체중", "나이")
```

```
val 어셈블러 = 새로운 VectorAssembler()
.setInputCols(Array("키", "체중", "나이"))
.setOutputCol("기능")
```

```
발 df2 = 어셈블러.변환(df)
```

```
df2.show(거짓)
```

```
+---+-----+-----+-----+
|아이디|키|체중|나이|특징|
+---+-----+-----+-----+
|0|186|200|56|[186.0,200.0,56.0]|
|1|170|198|42|[170.0,198.0,42.0]|
+---+-----+-----+-----+
```

```
발 스케일러 = 새로운 StandardScaler()
.setInputCol("기능")
.setOutputCol("scaledFeatures")
.setWithStd(참)
.setWithMean(거짓)
```

```
val 모델 = scaler.fit(df2)
```

```
val scaledData = model.transform(df2)
```

```
scaledData.select("기능","scaledFeatures").show(거짓)
+-----+-----+
|특징          |크기 조정된 기능
+-----+-----+
|[186.0,200.0,56.0][16.440232662587228,141.42135623730948,5.656854249492]|
|[170.0,198.0,42.0][15.026019100214134,140.0071426749364,4.2426406871192]|
+-----+-----+
```

데이터 크기 조정을 위한 추가 변환기에는 Normalizer, MinMaxScaler 및 MaxAbsScaler가 있습니다. 자세한 내용은 Apache Spark 온라인 설명서를 확인하십시오.

StopWordsRemover

종종 텍스트 분석에 사용되는 StopWordsRemover는 문자열 시퀀스에서 중지 단어를 제거합니다. I,,,와 같은 불용어는 문서의 의미에 크게 기여하지 않습니다.

org.apache.spark.ml.feature.StopWordsRemover 가져오기

```
val 제거기 = 새로운 StopWordsRemover().setInputCol("데이터").
setOutputCol("출력")
```

```
val dataSet = spark.createDataFrame(Seq(
  (0, Seq("그녀", "이다", "아", "귀엽다", "아기")),
  (1, Seq("밥", "절대", "가서", "까지", "시애틀"))
)).toDF("아이디", "데이터")
```

```
val df = remover.transform(dataSet)
```

```
df.show(거짓)
```

```
+---+-----+-----+
|아이디 |데이터           |출력
+---+-----+-----+
|0 |[그녀는, 이다, 귀엽다, 베이비]|[귀엽다, 베이비] |
|1 |[밥, 안 가, 가, 시애틀]|[밥, 안 가, 가, 시애틀]|
+---+-----+-----+
```

2장 Spark 및 Spark MLlib 소개

n-그램

텍스트 분석을 수행할 때 용어를 문서의 용어 조합인 n-그램으로 결합하는 것이 때때로 유리합니다. n-gram을 생성하면 문서에서 보다 의미 있는 정보를 추출하는 데 도움이 됩니다. 예를 들어, "San"과 "Diego"라는 단어는 개별적으로 큰 의미가 없지만 빅그램 "San Diego"로 결합하면 더 많은 컨텍스트를 제공합니다. 4 장 뒷부분에서 n-gram을 사용 합니다.

org.apache.spark.ml.feature.NGram 가져오기

```
val df = spark.createDataFrame(Seq(
  (0, Array("Los", "Angeles", "Wolfs", "Saint", "Francisco")),
  (1, Array("스탠드", "책", "케이스", "전화", "모바일", "잡지")),
  (2, Array("Deep", "Learning", "Machine", "Algorithm", "Pizza")))
)).toDF("아이디", "단어")
```

발 ngram = 새로운 NGram().setN(2).setInputCol("단어").setOutputCol("ngrams")

발 df2 = ngram.transform(df)

df2.select("ngrams").show(거짓)

엔그램
[로스앤젤레스, 앙헬레스 로보스, 로보스 산, 샌프란시스코]
[스탠드북, 북케이스, 케이스폰, 폰모바일, 모바일매거진]
[딥 러닝, 러닝 머신, 머신 알고리즘, 알고리즘 피자]

OneHotEncoderEstimator

원-핫 인코딩은 범주형 기능을 모든 기능 집합 중에서 특정 기능 값의 존재를 나타내는 최대 단일 값을 갖는 이진 벡터로 변환합니다. xi 원-핫 인코딩 범주형 변수는 다음과 같은 많은 기계 학습 알고리즘에 대한 요구 사항입니다. 로지스틱 회귀 및 지원 벡터 머신으로.

OneHotEncoderEstimator는 여러 열을 변환하여 각 입력 열에 대해 원-핫 인코딩된 벡터 열을 생성할 수 있습니다.

org.apache.spark.ml.feature.StringIndexer 가져오기

```
val df = spark.createDataFrame( Seq((0,
    "남성"), (1, "남성"), (2, "여성"), (3, "여성"), (4, "여성"), (5, "남성")) ).toDF("ID",
    "성별")
```

df.show()

```
+---+-----+
| 아이디|성별|
+---+-----+
| 0| 남성|
| 1| 남성|
| 2|여자|
| 3|여자|
| 4|여자|
| 5| 남성|
+---+-----+
```

발 인덱서 = 새로운

```
StringIndexer().setInputCol("성별").setOutputCol("성별 인덱스")
```

```
val indexed = indexer.fit(df).transform(df)
```

indexed.show()

```
+---+-----+
| 아이디|성별|성별색인|
+---+-----+
| 0| 남성| 1.0|
| 1| 남성| 1.0|
| 2|여자| 0.0|
| 3|여자| 0.0|
| 4|여자| 0.0|
| 5| 남성| 1.0|
+---+-----+
```

2장 Spark 및 Spark MLlib 소개

org.apache.spark.ml.feature.OneHotEncoderEstimator 가져오기

```
val 인코더 = new OneHotEncoderEstimator()
    .setInputCols(배열("성별 인덱스"))
    .setOutputCols(배열("genderEnc"))
```

인코딩된 val = encoder.fit(indexed).transform(indexed)

인코딩된.show()

아이디	성별	성별색인	성별엔코
0	남성		1.0 (1,[],[])
1	남성		1.0 (1,[],[])
2	여자		0.0 (1,[0],[1.0])
3	여자		0.0 (1,[0],[1.0])
4	여자		0.0 (1,[0],[1.0])
5	남성		1.0 (1,[],[])

SQL변환기

SQLTransformer를 사용하면 SQL을 사용하여 데이터 변환을 수행할 수 있습니다. 가상 테이블 "__THIS__"는 입력 데이터 세트에 해당합니다.

org.apache.spark.ml.feature.SQLTransformer 가져오기

```
발 df = spark.createDataFrame(
    시퀀스((0, 5.2, 6.7), (2, 25.5, 8.9))).toDF("id", "col1", "col2")

val 변환기 = new SQLTransformer().setStatement("SELECT ABS(col1 - col2) as c1,
MOD(col1, col2) as c2 FROM __THIS__")

발 df2 = 변환기.변환(df)

df2.show()
```

c1	c2
1.5	5.2
16.6	7.699999999999999

기간 빈도-역 문서 빈도(TF-IDF)

TF-IDF 또는 용어 빈도-역 문서 빈도는 텍스트 분석에서 일반적으로 사용되는 특징 벡터화 방법입니다. 코퍼스의 문서에 대한 용어나 단어의 중요성을 나타내는 데 자주 사용됩니다. 변환기인 HashingTF는 특성 해싱을 사용하여 용어를 특성 벡터로 변환합니다. 추정기 IDF는 HashingTF(또는 CountVectorizer)에 의해 생성된 벡터의 크기를 조정합니다. 4 장에서 TF-IDF에 대해 자세히 설명 합니다.

```
import org.apache.spark.ml.feature.{HashingTF, IDF, Tokenizer}
```

```
val df = spark.createDataFrame(Seq(
  (0, "Kawhi Leonard는 리그 MVP입니다"),
  (1, "Caravaggio는 바로크 기법을 개척했다"),
  (2, "아파치 스파크를 사용하는 것이 멋지다"))
).toDF("레이블", "문장")
```

```
df.show(거짓)
```

라벨	문장	
0	Kawhi Leonard는 리그 MVP입니다	
1	카라바조는 바로크 기법의 개척자	
2	Apache Spark를 사용하는 것이 좋습니다.	

2장 Spark 및 Spark MLlib 소개

```
val 토크나이저 = 새로운 토크나이저()
    .setInputCol("문장")
    .setOutputCol("단어")
```

발 df2 = 토크나이저.변환(df)

```
df2.select("레이블","단어").show(거짓)
+---+-----+
|라벨|단어
+---+-----+
|0|[카와이, 레너드, 이즈, 더, 리그, mvp] |
|1|[卡拉巴佐, 개척, 바로크, 기술]|
|2|[아파치, 스파크 사용, 멋지다] |
+---+-----+
```

```
val hashingTF = 새로운 HashingTF()
    .setInputCol("단어")
    .setOutputCol("기능")
    .setNumFeatures (20)
```

발 df3 = hashingTF.transform(df2)

```
df3.select("라벨","기능").show(거짓)
+---+-----+
|라벨|기능
+---+-----+
|0|(20,[1,4,6,10,11,18],[1.0,1.0,1.0,1.0,1.0,1.0])|
|1|(20,[1,5,10,12],[1.0,1.0,2.0,1.0])|
|2|(20,[1,4,5,15],[1.0,1.0,1.0,2.0])|
+---+-----+
```

```
val idf = 새로운 IDF()
    .setInputCol("기능")
    .setOutputCol("scaledFeatures")
```

발 idf모델 = idf.fit(df3)

```
발 df4 = idfModel.transform(df3)

df4.select("label", "scaledFeatures").show(3,50)
+---+-----+
|라벨|          스케일 기능|
+---+-----+
| 0|[20,[1,4,6,10,11,18],[0.0,0.28768207245178085,0...| 1|[20,[1,5,10,12],
| [0.0,0.28768207245178053,0] ...| 2|[20,[1,4,5,15],
| [0.0,0.28768207245178085,0.28768...|
+---+-----+
```

주성분 분석(PCA)

주성분 분석(PCA)은 상관된 특징을 주성분이라고 하는 선형적으로 상관되지 않은 특징의 더 작은 세트로 결합하는 차원 축소 기술입니다. PCA는 이미지 인식 및 이상 감지와 같은 여러 분야에서 응용 프로그램이 있습니다. PCA에 대해서는 4 장에서 더 자세히 설명 합니다.

```
org.apache.spark.ml.feature.PCA 가져오기
org.apache.spark.ml.linalg.Vectors 가져오기

val 데이터 = 배열
  (Vectors.dense(4.2, 5.4, 8.9, 6.7, 9.1), Vectors.dense(3.3,
  8.2, 7.0, 9.0, 7.2), Vectors.dense(6.1, 1.4, 2.2, 4.3, 2.9)
  val df = spark.createDataFrame(data.map(Tuple1.apply)).toDF("기
  능")
```

```
발 pca = 새로운
  PCA() .setInputCol("기
  능") .setOutputCol("pcaFeatures") .setK(2) .fit(df)

값 결과 = pca.transform(df).select("pcaFeatures")

result.show(거짓)
```

2장 Spark 및 Spark MLlib 소개

```
+-----+
|pca특징 |
+-----+
|[13.62324332562565,3.1399510055159445] ||
|[14.130156836243236,-1.432033103462711]|| 
|[3.4900743524527704,0.6866090886347056]|
+-----+
```

ChiSqSelector

ChiSqSelector는 기능 선택을 위해 카이제곱 독립성 테스트를 사용합니다. 카이제곱 검정은 두 범주 형 변수의 관계를 검정하는 방법입니다. numTopFeatures 는 기본 선택 방법입니다. 카이 제곱 검정을 기반으로 하는 설정된 수의 기능 또는 가장 예측 영향이 큰 기능을 반환합니다. 다른 선택 방법에는 백분위수, fpr, fdr 및 fwe가 있습니다.

org.apache.spark.ml.feature.ChiSqSelector 가져오기

org.apache.spark.ml.linalg.Vectors 가져오기

```
값 데이터 = Seq(
  (0, Vector.dense(5.1, 2.9, 5.6, 4.8), 0.0),
  (1, Vectors.dense(7.3, 8.1, 45.2, 7.6), 1.0),
  (2, Vectors.dense(8.2, 12.6, 19.5), 9.21), 1.0
)
```

val df = spark.createDataset(데이터).toDF("ID", "기능", "클래스")

값 선택기 = 새로운

```
ChiSqSelector() .setNumTopFeatures(1) .setFeaturesCol("기능") .setLabelCol("클래스") .set
```

val df2 = selector.fit(df).transform(df)

df2.show()

2장 Spark 및 Spark MLlib 소개

기능	클래스	선택된 기능
[0] [5.1,2.9,5.6,4.8]	0.0	1 [5.1]
[7.3,8.1,45.2,7.6]	1.0	2 [7.3]
[8.2,12.6,19.5,9.21]	1.0	3 [8.2]

상관 관계

상관 관계는 두 변수 간의 선형 관계의 강도를 평가합니다.

선형 문제의 경우 상관 관계를 사용하여 관련 기능(기능 클래스 상관 관계)을 선택하고 중복 기능(기능 내 상관 관계)을 식별할 수 있습니다. Spark MLlib는 Pearson과 Spearman의 상관 관계를 모두 지원합니다. 다음 예에서 상관은 입력 벡터에 대한 상관 행렬을 계산합니다.

가져오기 org.apache.spark.ml.linalg.{매트릭스, 벡터}

org.apache.spark.ml.stat.Correlation 가져오기

org.apache.spark.sql.Row 가져오기

```
값 데이터 = Seq(
    벡터.밀도(5.1, 7.0, 9.0, 6.0),
    벡터.밀도(3.2, 1.1, 6.0, 9.0),
    벡터.밀도(3.5, 4.2, 9.1, 3.0),
    벡터.밀도(9.1, 2.6, 7.2, 1.8)
)
```

```
val df = data.map(Tuple1.apply).toDF("기능")
```

기능
[5.1,7.0,9.0,6.0]
[3.2,1.1,6.0,9.0]
[3.5,4.2,9.1,3.0]
[9.1,2.6,7.2,1.8]

2장 Spark 및 Spark MLlib 소개

```
val 행(c1: 행렬) = Correlation.corr(df, "기능").head
```

```
c1: org.apache.spark.ml.linalg.Matrix =
1.0 -0.01325851107237613 -0.08794286922175912 -0.6536434849076798
-0.01325851107237613 1.0 0.8773748081826724 -0.1872850762579899
-0.08794286922175912 0.8773748081826724 1.0 -0.46050932066780714
-0.6536434849076798 -0.1872850762579899 -0.46050932066780714 1.0
```

```
val Row(c2: Matrix) = Correlation.corr(df, "features", "spearman").head
```

```
c2: org.apache.spark.ml.linalg.Matrix =
1.0 0.3999999999999999 0.1999999999999898 -0.80000000000000014
0.3999999999999999 1.0 0.8000000000000035 -0.1999999999999743
0.19999999999999898 0.8000000000000035 1.0 -0.3999999999999486
-0.8000000000000014 -0.1999999999999743 -0.3999999999999486 1.0
```

DataFrame 열에 저장된 값의 상관 관계를 다음과 같이 계산할 수도 있습니다.

다음에 표시됩니다.

```
dataDF.show
```

	꽃잎 길이	꽃잎 너비	꽃잎 길이	꽃잎 너비	클래스 라벨
	+-----+	+-----+	+-----+	+-----+	+-----+
.	5.1	3.5	1.4	0.2	아이리스-세토사 0.0
.	4.9	3.0	1.4	0.2	아이리스-세토사 0.0
.	4.7	3.2	1.3	0.2	아이리스-세토사 0.0
.	4.6	3.1	1.5	0.2	아이리스-세토사 0.0
.	5.0	3.6	1.4	0.2	아이리스-세토사 0.0
.	5.4	3.9	1.7	0.4	아이리스세토사 0.0
.	4.6	3.4	1.4	0.3	아이리스 세토사 0.0
.	5.0	3.4	1.5	0.2	아이리스-세토사 0.0
.	4.4	2.9	1.4	0.2	아이리스-세토사 0.0
.	4.9	3.1	1.5	0.1	아이리스세토사 0.0
.	5.4	3.7	1.5	0.2	아이리스-세토사 0.0
.	4.8	3.4	1.6	0.2	아이리스-세토사 0.0
.	4.8	3.0	1.4	0.1	아이리스세토사 0.0
.	4.3	3.0	1.1	0.1	아이리스세토사 0.0
.	5.8	4.0	1.2	0.2	아이리스-세토사 0.0

2장 Spark 및 Spark MLlib 소개

5.7	4.4	1.5	0.4 아이리스세토사 0.0
5.4	3.9	1.3	0.4 아이리스세토사 0.0
5.1	3.5	1.4	0.3 아이리스 세토사 0.0
5.7	3.8	1.7	0.3 아이리스세토사 0.0
5.1	3.8	1.5	0.3 아이리스 세토사 0.0

```
dataDF.stat.corr("꽃잎 길이","라벨")
```

```
res48: 더블 = 0.9490425448523336
```

```
dataDF.stat.corr("꽃잎 너비","라벨")
```

```
res49: 더블 = 0.9564638238016178
```

```
dataDF.stat.corr("sepal_length","레이블")
```

```
res50: 더블 = 0.7825612318100821
```

```
dataDF.stat.corr("sepal_width","레이블")
```

```
res51: 더블 = -0.41944620026002677
```

평가 지표

1 장에서 논의한 바와 같이 정밀도, 재현율 및 정확도는 모델의 성능을 평가하기 위한 중요한 평가 지표입니다. 그러나 특정 문제에 대한 최상의 측정 기준이 아닐 수도 있습니다.

수신기 동작 특성 아래 영역 (오록)

AUROC(수신기 작동 특성 아래 영역)는 이진 분류기를 평가하기 위한 일반적인 성능 메트릭입니다. ROC(수신기 작동 특성)는 거짓 긍정 비율에 대해 참 긍정 비율을 표시하는 그래프입니다. 곡선 아래 면적(AUC)은 ROC 곡선 아래 면적입니다. AUC는 모델이 임의의 부정적인 예보다 임의의 긍정적인 예를 더 높은 순위로 지정할 확률로 해석될 수 있습니다 .xii 곡선 아래 영역이 클수록 (AUROC가 1.0에 가까울수록) 모델의 성능이 더 좋습니다. AUROC가 0.5인 모델은 예측 정확도가 무작위 추측만큼 좋기 때문에 쓸모가 없습니다.

2장 Spark 및 Spark MLlib 소개

org.apache.spark.ml.evaluation.BinaryClassificationEvaluator 가져오기

```
val 평가자 = 새로운 BinaryClassificationEvaluator()
    .setMetricName("areaUnderROC")
    .setRawPredictionCol("rawPrediction")
    .setLabelCol("레이블")
```

F1 측정

F1 측정값 또는 F1 점수는 정밀도와 재현율의 조화 평균 또는 가중 평균입니다. 다중 클래스 분류기를 평가하기 위한 일반적인 성능 메트릭입니다. 불균등한 등급 분포가 있는 경우에도 좋은 척도입니다. 가장 좋은 F1 점수는 1이고 가장 나쁜 점수는 0입니다. 좋은 F1 측정은 낮은 거짓 음성과 낮은 거짓 양성이 있음을 의미합니다. F1 측정의 공식은 다음과 같습니다. $F1\text{-측정} = 2 * (\text{정밀도} * \text{재현율}) / (\text{정밀도} + \text{재현율})$.

org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator 가져오기

```
가치 평가자 = 새로운 MulticlassClassificationEvaluator()
    .setMetricName("f1")
    .setLabelCol("레이블")
    .setPredictionCol("예측")
```

RMSE(평균 제곱근 오차)

RMSE(Root Mean Squared Error)는 회귀 작업에 대한 가장 일반적인 메트릭입니다.

RMSE는 단순히 평균 제곱 오차(MSE)의 제곱근입니다. MSE는 점에서 회귀선까지의 거리 또는 "오차"를 취하고 제곱하여 회귀선이 데이터 점 집합에 얼마나 가까운지를 나타냅니다.^{xiii} MSE가 작을수록 적합성이 더 좋습니다. 그러나 MSE는 값이 제곱되기 때문에 원본 데이터의 단위와 일치하지 않습니다. RMSE는 출력과 동일한 단위를 갖습니다.

org.apache.spark.ml.evaluation.RegressionEvaluator 가져오기

```
val 평가자 = 새로운 RegressionEvaluator()
    .setLabelCol("레이블")
    .setPredictionCol("예측")
    .setMetricName("rmse")
```

WSSSE(Within Set Sum of Squared Errors)와 같은 다른 평가 메트릭을 다룹니다. 다음 장에서 실루엣 계수. Spark MLlib에서 지원하는 모든 평가 지표의 전체 목록은 Spark의 온라인 설명서를 참조하세요.

모델 지속성

Spark MLlib를 사용하면 모델을 저장하고 나중에 로드할 수 있습니다. 이는 모델을 타사 응용 프로그램과 통합하거나 팀의 다른 구성원과 공유하려는 경우에 특히 유용합니다.

단일 Random Forest 모델 저장

```
rf = RandomForestClassifier(numBins=10,numTrees=30)
모델 = rf.fit(훈련)
model.save("모델 경로")
```

단일 랜덤 포레스트 모델 로드

```
val model2 = RandomForestClassificationModel.load("모델 경로")
```

전체 파이프라인 저장

```
val 파이프라인 = new Pipeline().setStages(Array(labelIndexer,vectorAssembler, rf))

val cv = 새로운 CrossValidator().setEstimator(파이프라인)
val 모델 = cv.fit(훈련)
model.save("모델 경로")
```

전체 파이프라인 로드

```
val model2 = CrossValidatorModel.load("모델 경로")
```

Spark MLlib 예제

예제로 작업해 보겠습니다. UCI Machine Learning Repository 의 Heart Disease Data [Setxiv](#) 를 사용하여 심장 질환의 존재를 예측합니다. 데이터는 VA Medical Center, Long Beach 및 Cleveland Clinic Foundation의 Robert Detrano, MD, PhD와 그의 팀이 수집했습니다. 역사적으로 Cleveland 데이터셋은 수많은 연구의 주제였으므로 우리는 그 데이터셋을 사용할 것입니다. 원본 데이터 세트에는 76개의 속성이 있습니다.

2장 Spark 및 Spark MLlib 소개

그러나 그 중 14개만이 ML 연구에 사용됩니다(표 2-1). 우리는 이항 분류를 수행하고 환자가 심장병이 있는지 여부를 결정할 것입니다(목록 2-2).

표 2-1. 클리블랜드 심장병 데이터 세트 속성 정보

기인하다	설명
나이	나이
섹스	섹스
cp	가슴 통증 유형
trestbps	안정시 혈압
총	mg/dl의 혈청 콜레스테롤
fbs	공복 혈당 > 120mg/dl
재테크	안정시 심전도 결과
탈라크	최대 심박수 달성
엑상	운동 유발 협심증
올드피크	휴식에 비해 운동으로 유발된 ST 우울증
경사	피크 운동 ST 세그먼트의 기울기
저것	형광등으로 채색된 주요 혈관의 수(0~3)
탈	탈륨 스트레스 테스트 결과
하나에	예측 속성 – 심장병 진단

시작하자. 파일을 다운로드하여 HDFS에 복사합니다.

```
wget http://archive.ics.uci.edu/ml/machine-learning-databases/heart_disease/cleveland.data
```

```
헤드 -n 10 처리.클리블랜드.데이터
```

```
63.0,1.0,1.0,145.0,233.0,1.0,2.0,150.0,0.0,2.3,3.0,0.0,6.0,0
67.0,1.0,4.0,160.0,286.0,0.0,2.0,108.0,1.0,1.5,2.0,3.0,3.0,2
67.0,1.0,4.0,120.0,229.0,0.0,2.0,129.0,1.0,2.6,2.0,2.0,7.0,1
37.0,1.0,3.0,130.0,250.0,0.0,0.0,187.0,0.0,3.5,3.0,0.0,3.0,0
41.0,0.0,2.0,130.0,204.0,0.0,2.0,172.0,0.0,1.4,1.0,0.0,3.0,0
56.0,1.0,2.0,120.0,236.0,0.0,0.0,178.0,0.0,0.8,1.0,0.0,3.0,0
```

```
62.0,0.0,4.0,140.0,268.0,0.0,2.0,160.0,0.0,3.6,3.0,2.0,3.0,3
57.0,0.0,4.0,120.0,354.0,0.0,0.0,6,16,3.0,0
63.0,1.0,4.0,130.0,254.0,0.0,2.0,147.0,0.0,1.4,2.0,1.0,7.0,2
53.0,1.0,4.0,140.0,203.0,1.0,203.0,1.0,3.0,0.0,7.0,1
```

hadoop fs -put processing.cleveland.data /tmp/data

우리는 spark-shell을 사용하여 모델을 대화식으로 훈련합니다.

목록 2-2. 랜덤 포레스트를 사용하여 이진 분류 수행

불꽃 껍질

```
val dataDF = spark.read.format("csv") .option("해
더", "true") .option("inferSchema",
"true") .load(d("/tmp/data/
processed.cleveland.
data") .toDF("id","age","sex","cp","trestbps","chol","fbs","restecg",
"thalach","exang","oldpeak", "기울기", "ca", "thal", "num")
```

dataDF.printSchema

뿌리

```
|-- id: 문자열(nullable = false) |-- age:
float(nullable = true) |-- 성별: float(nullable =
true) |-- cp: float(nullable = true) |-- trestbps:
float (nullable = true) |-- chol: float(nullable =
true) |-- fbs: float(nullable = true) |-- restecg:
float(nullable = true) |-- thalach: float(nullable =
true) | -- exang: float(nullable = true) |-- oldpeak:
float(nullable = true) |-- slope: float(nullable = true)
|-- ca: float(nullable = true) |-- thal: float( nullable =
true) |-- num: float(nullable = true)
```

2장 Spark 및 Spark MLlib 소개

```
val myFeatures = Array("나이", "성별", "cp", "trestbps", "chol", "fbs",
    "retecg", "thalach", "exang", "oldpeak", "slope", "ca", "thal",
    "num")
```

org.apache.spark.ml.feature.VectorAssembler 가져오기

발 어셈블러 = 새로운

```
VectorAssembler() .setInputCols(myFeatures) .setOutputCol("기능")
```

```
val dataDF2 = assembler.transform(dataDF)
```

org.apache.spark.ml.feature.StringIndexer 가져오기

val labelIndexer = 새로운

```
StringIndexer() .setInputCol("숫자") .setOutputCol("레이블")
```

```
val dataDF3 = labelIndexer.fit(dataDF2).transform(dataDF2)
```

val dataDF4 =

```
dataDF3.where(dataDF3("ca").isNotNull) .where(dataDF3("thal").isNotNull) .where(dataDF3("
```

```
val Array(trainingData, testData) = dataDF4.randomSplit(Array(0.8, 0.2), 101) import
org.apache.spark.ml.classification.RandomForestClassifier 가져오기
```

val rf = 새로운

```
RandomForestClassifier() .setFeatureSubsetStrategy("자동") .setSeed(101)
```

org.apache.spark.ml.evaluation.BinaryClassificationEvaluator 가져오기

값 평가자 = 새로운 BinaryClassificationEvaluator().setLabelCol("레이블")

org.apache.spark.ml.tuning.ParamGridBuilder 가져오기

val pgrid = 새로운

```
ParamGridBuilder() .addGrid(rf.maxBins, Array(10,
20, 30)) .addGrid(rf.maxDepth, Array(5, 10, 15))
```

```
.addGrid(rf.numTrees, Array(20, 30,
40)) .addGrid(rf.impurity, Array("gini", "엔트로피")) .build()
```

org.apache.spark.ml.Pipeline 가져오기

```
val 파이프라인 = new Pipeline().setStages(Array(rf))
```

org.apache.spark.ml.tuning.CrossValidator 가져오기

```
val cv = 새로운
```

```
CrossValidator() .setEstimator(파
이프라인) .setEvaluator(평가
```

```
자) .setEstimatorParamMaps(pgrid) .setNumFolds(3)
```

이제 모델을 맞출 수 있습니다.

```
val 모델 = cv.fit(trainingData)
```

테스트 데이터에 대한 예측을 수행합니다.

```
val 예측 = model.transform(testData)
```

모델을 평가해 봅시다.

org.apache.spark.ml.param.ParamMap 가져오기

```
val pm = ParamMap(evaluator.metricName -> "areaUnderROC")
```

```
val aucTestData = evaluator.evaluate(예측, 오후)
```

그래프 처리

Spark에는 GraphX라는 그래프 처리 프레임워크가 포함되어 있습니다. DataFrames를 기반으로 하는 GraphFrames라는 별도의 패키지가 있습니다. GraphFrames는 현재 핵심 Apache Spark의 일부가 아닙니다. 이 글을 쓰는 시점에서 GraphX와 GraphFrames는 여전히 활발히 개발 중입니다 .xv 6 장에서 GraphX를 다룹니다 .

2장 Spark 및 Spark MLlib 소개

Spark MLlib 너머: 타사 기계 학습 통합

Spark는 수많은 오픈 소스 기여자와 Microsoft 및 Google과 같은 회사 덕분에 풍부한 타사 프레임워크 및 라이브러리 에코시스템에 액세스할 수 있습니다.

핵심 Spark MLlib 알고리즘을 다루는 동안 이 책은 XGBoost, LightGBM, Isolation Forest, Spark NLP 및 분산 딥 러닝과 같은 보다 강력한 차세대 알고리즘 및 프레임워크에 중점을 둡니다. 다음 장에서 다루겠습니다.

Alluxio로 Spark 및 Spark MLlib 최적화

이전에 Tachyon으로 알려졌던 Alluxio는 UC Berkeley AMPLab의 오픈 소스 프로젝트입니다.

Alluxio는 원래 2012년 연구 프로젝트로 개발된 Haoyuan Li, AMPLab.xvi의 창립 Apache Spark 커미터가 연구 프로젝트로 개발한 분산 메모리 중심 스토리지 시스템입니다. 이 프로젝트는 BDAS(Berkeley Data Analytics Stack)의 스토리지 계층입니다. 2015년에 Alluxio, Inc.는 Alluxio를 상용화하기 위해 Li에 의해 설립되었으며 Andreessen Horowitz로부터 750만 달러의 현금을 받았습니다. 오늘날 Alluxio에는 Intel, IBM, Yahoo, Red Hat과 같은 전 세계 50개 조직의 200명 이상의 기여자가 있습니다. 현재 Baidu, Alibaba, Rackspace 및 Barclays와 같은 여러 유명 기업이 프로덕션에서 Alluxio를 사용하고 있습니다.xvii

Alluxio는 초고속 빅 데이터 스토리지를 초대형 데이터 세트로 활성화하여 Spark 머신 러닝 및 딥 러닝 워크로드를 최적화하는 데 사용할 수 있습니다. Alluxio에서 수행한 딥 러닝 벤치마크는 S3.xviii 대신 Alluxio에서 데이터를 읽을 때 상당한 성능 향상을 보여줍니다.

건축물

Alluxio는 메모리 중심의 분산 스토리지 시스템이며 빅 데이터를 위한 사실상의 스토리지 통합 레이어가 되는 것을 목표로 합니다. Local FS, HDFS, S3, NFS와 같은 다양한 스토리지 엔진과 Spark, MapReduce, Hive, Presto와 같은 컴퓨팅 프레임워크에 대한 액세스를 통합하는 가상화 계층을 제공합니다. 그림 2-4는 Alluxio의 아키텍처에 대한 개요를 제공합니다.

2장 Spark 및 Spark MLlib 소개



그림 2-4. 알록시오 아키텍처 개요

Alluxio는 데이터 공유를 조정하고 데이터 액세스를 지시하는 동시에 컴퓨팅 프레임워크와 빅 데이터 애플리케이션에 고성능 저지연 메모리 속도를 제공하는 중간 계층입니다. Alluxio는 Spark 및 Hadoop과 원활하게 통합되며 약간의 구성 변경만 필요합니다. Alluxio의 통합 네임스페이스 기능을 활용함으로써 애플리케이션은 지원되는 스토리지 엔진에 저장된 데이터에 액세스하기 위해 Alluxio에 연결하기만 하면 됩니다. Alluxio에는 자체 기본 API와 Hadoop 호환 파일 시스템 인터페이스가 있습니다. 편의 클래스를 사용하면 코드 변경 없이 원래 Hadoop용으로 작성된 코드를 실행할 수 있습니다. REST API는 다른 언어에 대한 액세스를 제공합니다. 이 장의 뒷부분에서 API를 살펴보겠습니다.

Alluxio의 통합 네임스페이스 기능은 관계형 데이터베이스 및 MPP를 지원하지 않습니다. Redshift 또는 Snowflake와 같은 엔진 또는 MongoDB와 같은 문서 데이터베이스. 물론, Alluxio 및 언급된 스토리지 엔진에 대한 쓰기가 지원됩니다. 개발자는 Spark와 같은 컴퓨팅 프레임워크를 사용하여 Redshift 테이블에서 데이터 프레임을 생성하고 이를 Parquet 또는 CSV 형식으로 Alluxio 파일 시스템에 저장할 수 있으며 그 반대의 경우도 마찬가지입니다(그림 2-5).

2장 Spark 및 Spark MLlib 소개

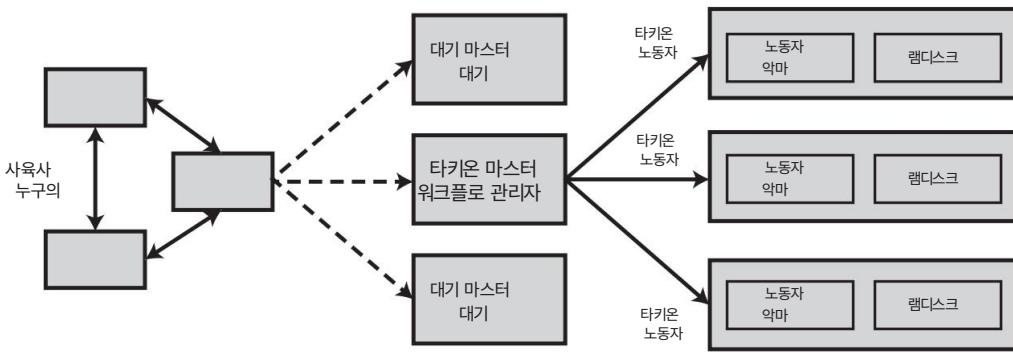


그림 2-5. 알룩시오 기술 아키텍처

왜 Alluxio를 사용합니까?

빅 데이터 처리 성능 및 확장성을 대폭 향상

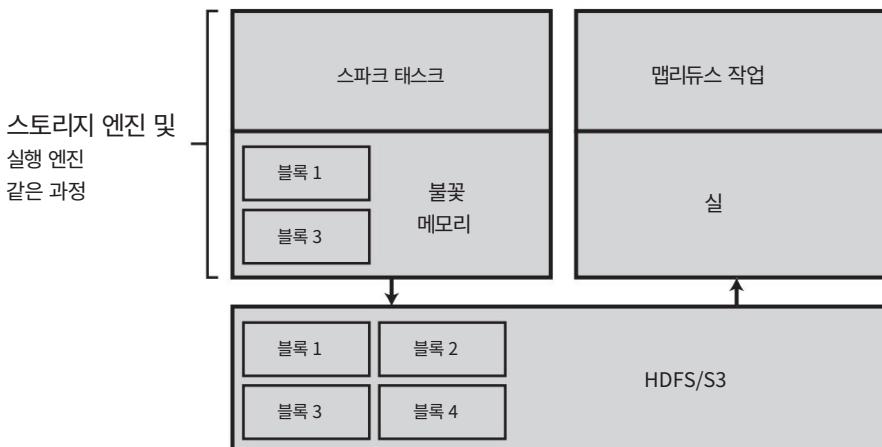
수년에 걸쳐 메모리는 더 저렴해지고 성능은 더 빨라졌습니다.

한편, 하드 드라이브의 성능은 약간만 향상되었습니다. 메모리에서 데이터를 처리하는 것이 디스크에서 데이터를 처리하는 것보다 훨씬 빠릅니다. 거의 모든 프로그래밍 패러다임에서 성능을 향상시키기 위해 메모리에 데이터를 캐시하는 것이 좋습니다. MapReduce에 비해 Apache Spark의 주요 장점 중 하나는 데이터를 캐시하는 기능입니다. Alluxio는 이를 한 단계 더 발전시켜 빅 데이터 애플리케이션을 캐싱 레이어뿐만 아니라 완전한 분산형 고성능 메모리 중심 스토리지 시스템으로 제공합니다.

Baidu는 2PB 이상의 데이터를 처리하는 1000개의 작업자 노드와 함께 세계에서 가장 큰 Alluxio 클러스터 중 하나를 운영하고 있습니다. Baidu는 Alluxio를 사용하여 쿼리 및 처리 시간에서 평균 10배, 최대 30배의 성능 향상을 보고 있으며 중요한 비즈니스 결정을 내리는 Baidu의 능력을 크게 향상시키고 있습니다.^{xix} Barclays는 Alluxio에 대한 경험을 설명하는 기사를 게시했습니다. Barclays 데이터 과학자 Gianmario Spacagna와 고급 분석 책임자인 Harry Powell은 Alluxio.xx를 사용하여 Spark 작업을 몇 시간에서 몇 초로 조정할 수 있었습니다. 중국 최대 여행 검색 엔진 중 하나인 Qunar.com은 Alluxio를 사용하여 성능이 15~300배 향상되었습니다. ^{.xxi}

여러 프레임워크와 애플리케이션이 메모리 속도로 데이터를 공유할 수 있음

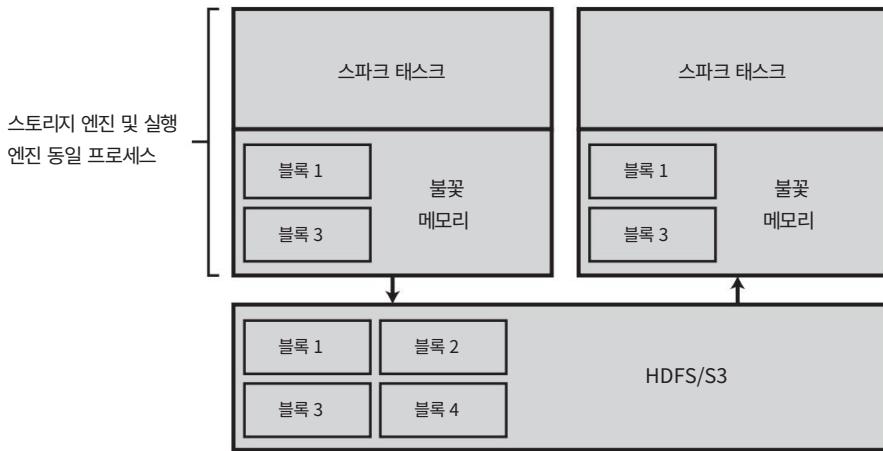
일반적인 빅 데이터 클러스터에는 Spark 및 MapReduce와 같은 서로 다른 컴퓨팅 프레임워크를 실행하는 여러 세션이 있습니다. Spark의 경우 각 애플리케이션은 자체 실행기 프로세스를 가져오고 실행기 내의 각 작업은 자체 JVM에서 실행되어 Spark 애플리케이션을 서로 격리합니다. 즉, Spark(및 MapReduce) 애플리케이션은 HDFS 또는 S3와 같은 스토리지 시스템에 쓰는 것 외에는 데이터를 공유할 수 없습니다. 그림 2-6과 같이 Spark 작업과 MapReduce 작업은 HDFS 또는 S3에 저장된 동일한 데이터를 사용하고 있습니다. 그림 2-7에서 여러 Spark 작업은 각 작업이 자체 힙 공간에 자체 버전의 데이터를 저장하는 동일한 데이터를 사용하고 있습니다.^{xxii} 데이터가 복제될 뿐만 아니라 HDFS 또는 S3를 통한 데이터 공유가 느려질 수 있습니다. 많은 양의 데이터를 공유하고 있습니다.



네트워크 및 디스크 IO로 인해 프로세스 간 데이터 공유 지연 및 처리량이 느려짐

그림 2-6. HDFS 또는 S3를 통해 데이터를 공유하는 다양한 프레임워크

2장 Spark 및 Spark MLlib 소개



네트워크 및 디스크 I/O로 인해 프로세스 간 데이터 공유 지연 및 처리량이 느려짐

그림 2-7. HDFS 또는 S3를 통해 데이터를 공유하는 다양한 작업

Alluxio를 오프 힙 저장소로 사용하면(그림 2-8) 여러 프레임워크와 작업이 메모리 속도로 데이터를 공유하여 데이터 중복을 줄이고 처리량을 늘리며 대기 시간을 줄일 수 있습니다.

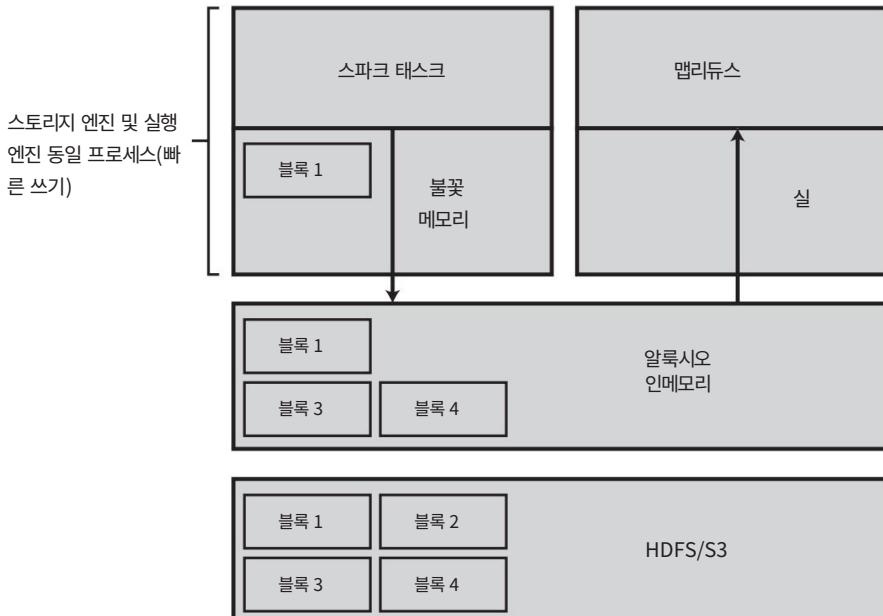


그림 2-8. 메모리 속도로 데이터를 공유하는 다양한 작업 및 프레임워크

애플리케이션 종료 또는 실패 시 고가용성 및 지속성 제공

Spark에서 실행기 프로세스와 실행기 메모리는 동일한 JVM에 상주하며 모든 캐시된 데이터는 JVM 힙 공간에 저장됩니다(그림 2-9).

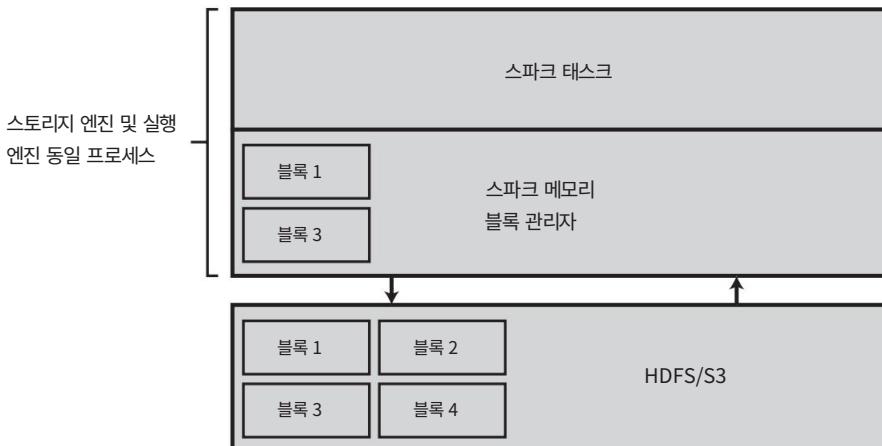


그림 2-9. 자체 힙 메모리가 있는 Spark 작업

작업이 완료되거나 어떤 이유로 JVM이 런타임 예외로 인해 충돌할 때,
힙 공간에 캐시된 모든 데이터는 그림 2-10 및 2-11과 같이 손실됩니다.

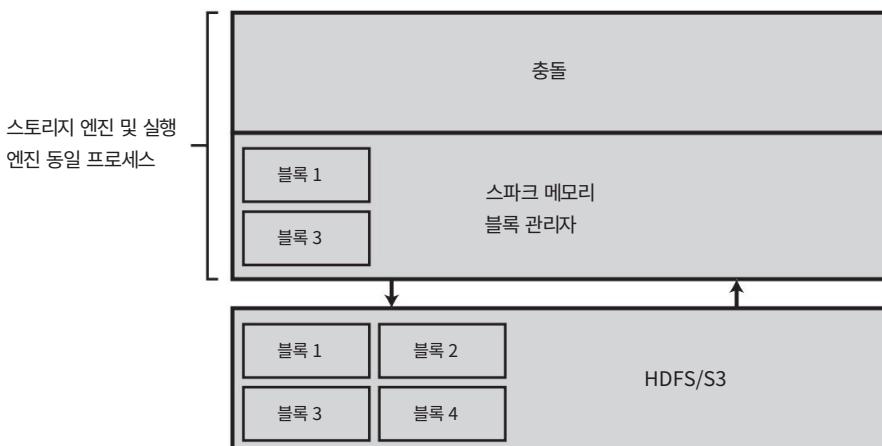


그림 2-10. Spark 작업이 충돌하거나 완료됨

2장 Spark 및 Spark MLlib 소개

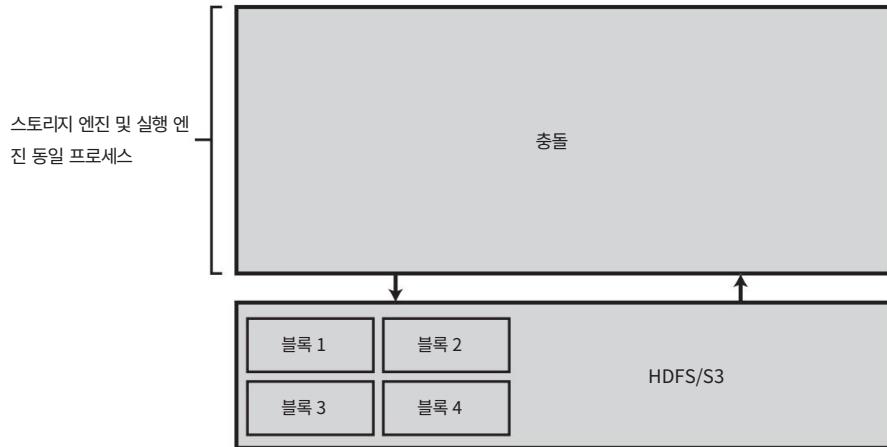


그림 2-11. Spark 작업이 총돌하거나 완료됩니다. 힙 공간이 손실됨

해결책은 Alluxio를 오프힙 저장소로 사용하는 것입니다(그림 2-12).

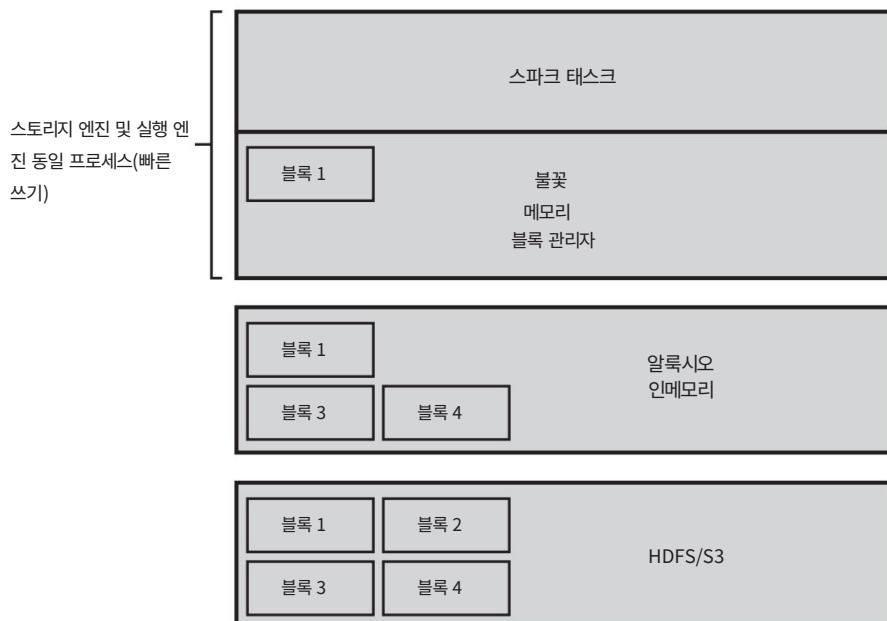


그림 2-12. Alluxio를 오프 힙 스토리지로 사용하는 Spark

이 경우 Spark JVM이 총돌하더라도 데이터는 Alluxio에서 계속 사용할 수 있습니다(그림 2-13 및 2-14).

2장 Spark 및 Spark MLlib 소개

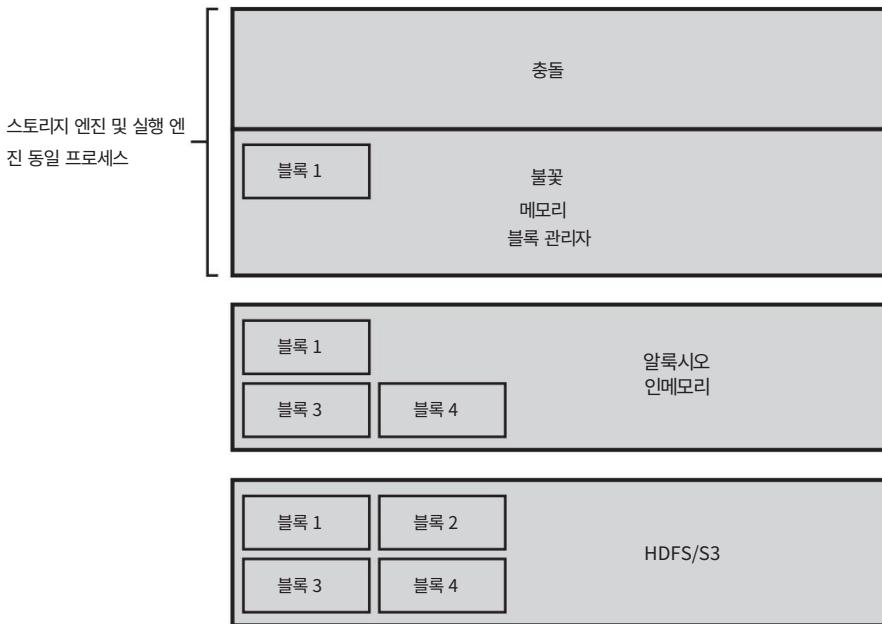


그림 2-13. Spark 작업이 충돌하거나 완료됨

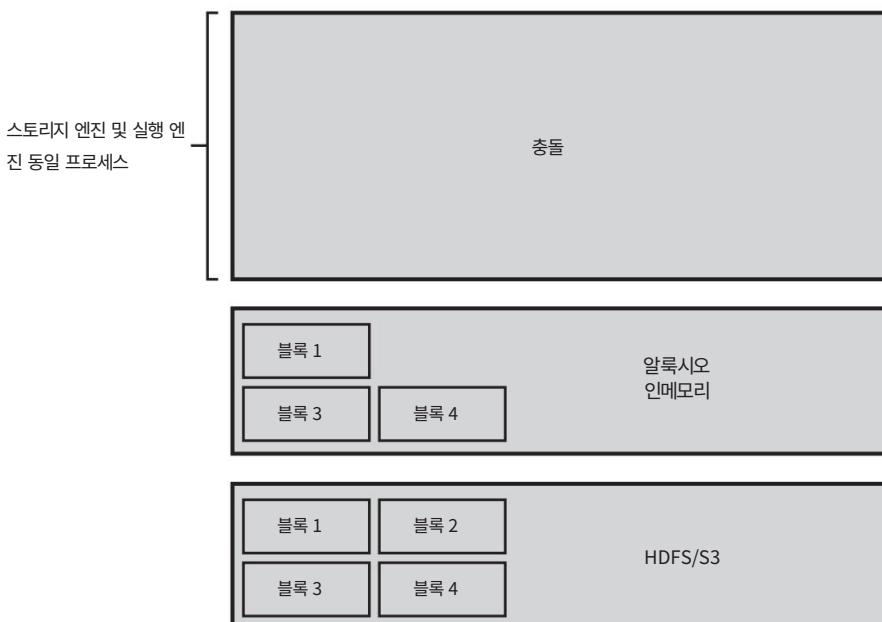


그림 2-14. Spark 작업이 충돌하거나 완료됩니다. 힙 공간이 손실됩니다. 오프 힙 메모리는 여전히 사용 가능합니다.

2장 Spark 및 Spark MLlib 소개

전체 메모리 사용량 최적화 및 최소화 쓰레기 수거

Alluxio를 사용하면 작업과 프레임워크 간에 데이터가 공유되기 때문에 메모리 사용이 훨씬 더 효율적이고 데이터가 오프힙에 저장되기 때문에 가비지 수집도 최소화되어 작업 및 애플리케이션의 성능을 더욱 향상시킵니다 (그림 2-15).

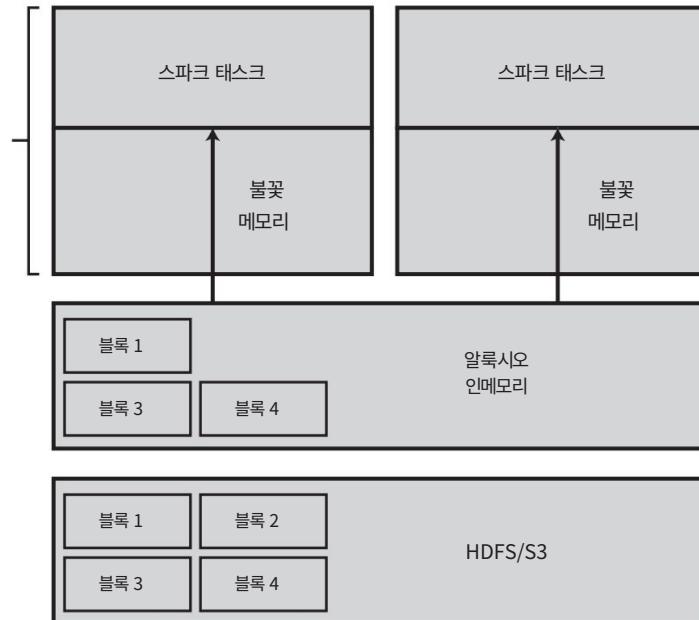


그림 2-15. 여러 Spark 및 MapReduce 작업이 Alluxio에 저장된 동일한 데이터에 액세스할 수 있음

하드웨어 요구 사항 감소

Alluxio를 사용한 빅 데이터 처리는 HDFS 및 S3보다 훨씬 빠릅니다. IBM의 테스트에 따르면 Alluxio는 쓰기 IO에 대해 HDFS보다 110배 더 우수한 것으로 나타났습니다 .xxiii 이러한 종류의 성능을 사용하면 추가 하드웨어에 대한 요구 사항이 줄어들어 인프라 및 라이센스 비용을 절약할 수 있습니다.

아파치 스파크와 알록시오

Spark에서 HDFS 및 S3에 저장된 데이터에 액세스하는 것과 유사하게 Alluxio의 데이터에 액세스합니다.

```
val dataRDD = sc.textFile("alluxio://로컬 호스트: 19998/test01.csv")

val parsedRDD = dataRDD.map {_.split(",")}

케이스 클래스 CustomerData(userid: Long, city: String, state: String, age: Short)

val dataDF = parsedRDD.map{ a =>CustomerData(a(0).toLong, a(1).toString, a(2).toString,
a(3).toShort )}.toDF

dataDF.show()

+-----+-----+-----+
|사용자 ID|          시|주|나이|
+-----+-----+-----+
| 300|      토랜스| 캘리포니아| 23|
| 302|맨해튼 비치| 캘리포니아| 21|
+-----+-----+-----+
```

요약

이 장에서는 일반적인 데이터 처리 및 기계 학습 작업을 수행하는 데 필요한 기술을 제공하기에 충분한 Spark 및 Spark MLlib에 대한 간략한 소개를 제공했습니다. 내 목표는 가능한 한 빨리 속도를 높이는 것이었습니다. 보다 철저한 치료를 위해 Bill Chambers와 Matei Zaharia 의 Spark: Definitive Guide (O'Reilly, 2018)에서 Spark에 대한 철저한 소개를 제공합니다. Irfan Elahi 의 Scala Programming for Big Data Analytics (Apress, 2019), 학습 Scala by Jason Swartz(O'Reilly, 2014), Programming in Scala by Martin Odersky, Lex Spoon, Bill Veners(Artima, 2016)가 좋은 소개입니다. 스칼라로. 또한 대규모 머신 러닝 및 딥 러닝 워크로드를 최적화하는 데 사용할 수 있는 인메모리 분산 컴퓨팅 플랫폼인 Alluxio에 대해서도 소개했습니다.

참고문헌

- 나. Peter Norvig et al.; "데이터의 불합리한 효율성",
googleusercontent.com, 2009년, <https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/35179.pdf>
- ii. 불꽃; "스파크 개요", spark.apache.org, 2019, <https://spark.apache.org/docs/2.2.0/>
- iii. 아파치 소프트웨어 재단; "Apache Software Foundation, Apache Spark를 최상
위 프로젝트로 발표" blogs.apache.org, 2014, https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces50
- iv. 불꽃; "스파크 뉴스", spark.apache.org, 2019, <https://spark.apache.org/news/>
- v. 레딧; "Matei Zaharia AMA," reddit.com, 2015년, www.reddit.com/r/IAmA/comments/31bkue/im_matei_zaharia_creator_of_spark_and_cto_at/?st=j1svbrx9&sh=a8b9698e
- 우리. 데이터브릭; "아파치 스파크," Databricks.com, 2019년, <https://databricks.com/spark/about>
- vii. 데이터브릭; "Apache Spark 2.0에서 SparkSession을 사용하는 방법",
Databricks.com, 2016, <https://databricks.com/blog/2016/08/15/how-to-use-sparksession-in-apache-spark-2-0.html>
- viii. 솔라; "SolrJ 사용", lucene.apache.org, 2019, https://lucene.apache.org/solr/guide/6_6/using-solrj.html
- ix. 루시드워克斯; "Lucidworks Spark/Solr 통합", github.com, 2019년, <https://github.com/lucidworks/spark-solr>
- 엑스. 불꽃; "MLlib(기계 학습 라이브러리) 가이드", spark.apache.org, <http://spark.apache.org/docs/latest/ml-guide.html>

2장 Spark 및 Spark MLlib 소개

- xi. 불꽃; "OneHotEncoderEstimator", spark.apache.org, 2019,
<https://spark.apache.org/docs/latest/ml-features#onehotencoderestimator>
- xii. Google; "분류: ROC 곡선 및 AUC" 개발자.
google.com, 2019년, [https://developers.google.com/machine learning/crash-course/classification/roc-and-auc](https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc)
- xiii. 스테파니 글렌; "평균 제곱 오차: 정의 및 예," statisticshowto.datasciencecentral.com, 2013,
www.statisticshowto.datasciencecentral.com/mean-squared error/
- xiv. 안드拉斯 야노시, 윌리엄 스타인브룬, 마티아스 피스터리, 로버트 데트라노; "심장 질환 데이터 세트", archive.ics.uci.edu, 1988, <http://archive.ics.uci.edu/ml/datasets/heart+Disease>
- xv. 불꽃; "GraphX", spark.apache.org, 2019, <https://spark.apache.org/graph/>
- 16. 크리스 매트만; "인큐베이터용 Apache Spark" 메일 아카이브.
apache.org, 2013, http://mail-archives.apache.org/mod_mbox/incubator-general/201306.mbox/%3CCDD80F64.D5F9D%25chris.a.mattmann@jpl.nasa.gov%3E
- xvii. 리 하오위안; "Alluxio(구 Tachyon), 새로운 시대를 열다 1.0 릴리스와 함께", alluxio.io, 2016, www.alluxio.com/blog/alluxio-%25tachyon-is-entering-a-new-era-with-10-pulls
- xviii. 유평푸; "Alluxio를 사용한 딥 러닝을 위한 유연하고 빠른 스토리지", alluxio.io, 2018, www.alluxio.io/blog/flexible-and-fast-storage-for-deep-learning-with-alluxio/
- xix. 알록시오; "Alluxio, 인메모리 속도로 페타바이트 규모 컴퓨팅을 위한 분산 스토리지 가상화", Globenewswire.com, 2016, www.marketwired.com/press-release/alluxio-virtualizes-distributed-storage-petabyte-scale-computing-in-memory-speed-2099053.html

2장 Spark 및 Spark MLlib 소개

더블 엑스. 헨리 파월과 지안마리오 스파카나; "Tachyon으로 불가능을 가능하게 만들기:

스파크 작업을 몇 시간에서 몇 초로 가속화", dzone.com, 2016, <https://dzone.com/articles/>

타키온으로 몇 시간에서 몇 초로 스파크를 사용하여 메모리 내 처리 가속화

xxi. 리 하오위안; "Strata+Hadoop World의 Alluxio 기조 연설

베이징 2016", slideshare.net, 2016, www.slideshare.net/Alluxio/alluxio-keynote-at-stratahadoop-world-beijing-2016-65172341

xxii. Mingfei S.; "사용 사례로 Tachyon 시작하기", intel.com, 2016, <https://software.intel.com/en-us/blogs/2016/02/04/>

사용 사례별 tachyon 시작하기

xxiii. 길 베르니크, "초고속 빅 데이터 처리를 위한 Tachyon", ibm.com, 2015년,

[www.ibm.com/blogs/research/2015/08/tachyon-for ultra-fast-big-data-processing/](http://www.ibm.com/blogs/research/2015/08/tachyon-for-ultra-fast-big-data-processing/)

3 장

지도 학습

가장 확실한 종류의 지식은 스스로 구성하는 것입니다.

—주다 펄리

지도 학습은 훈련 데이터 세트를 사용하여 예측을 수행하는 기계 학습 작업입니다. 지도 학습은 분류 또는 회귀로 분류할 수 있습니다.

회귀는 가격, 온도 또는 거리와 같은 연속 값을 예측하기 위한 것이고 분류는 예 또는 아니오, 스팸 또는 스팸 아님, 악성 또는 양성과 같은 범주를 예측하기 위한 것입니다.

분류

분류는 아마도 가장 일반적인 지도 머신 러닝 작업일 것입니다. 인식하지 못한 채 분류를 활용한 애플리케이션을 이미 접했을 가능성이 큽니다. 인기 있는 사용 사례로는 의료 진단, 표적 마케팅, 스팸 감지, 신용 위험 예측, 감정 분석 등이 있습니다. 분류 작업에는 세 가지 유형이 있습니다.

이진 분류

작업은 범주가 두 개뿐인 경우 이진 또는 이항 분류입니다. 예를 들어 스팸 탐지에 이진 분류 알고리즘을 사용할 때 출력 변수에는 스팸 또는 스팸이 아닌 두 가지 범주가 있을 수 있습니다. 암을 감지하기 위해 범주는 악성 또는 양성일 수 있습니다. 타겟 마케팅의 경우 누군가가 우유와 같은 품목을 구매할 가능성을 예측하는 경우 카테고리는 단순히 예 또는 아니오일 수 있습니다.

3장 지도 학습

다중 클래스 분류

다중 클래스 또는 다항 분류 작업에는 3개 이상의 범주가 있습니다. 예를 들어 기상 조건을 예측하기 위해 비, 흐림, 화창, 눈, 바람의 5가지 범주가 있을 수 있습니다. 타겟 마케팅 예를 확장하기 위해 다중 클래스 분류를 사용하여 고객이 전유, 저지방 우유, 저지방 우유 또는 탈지 우유를 구매할 가능성이 더 높은지 예측할 수 있습니다.

다중 레이블 분류

다중 레이블 분류에서는 각 관찰에 여러 범주를 할당할 수 있습니다. 대조적으로, 다중 클래스 분류에서는 하나의 범주만 관찰에 할당될 수 있습니다.

표적 마케팅 예를 사용하여 다중 레이블 분류는 고객이 우유뿐만 아니라 쿠키, 버터, 핫도그 또는 빵과 같은 기타 품목을 구매할 가능성이 더 높은지 예측하는 데 사용됩니다.

Spark MLlib 분류 알고리즘

Spark MLlib에는 분류를 위한 여러 알고리즘이 포함되어 있습니다. 저는 가장 인기 있는 알고리즘에 대해 논의하고 2장에서 다룬 내용을 기반으로 하는 따라하기 쉬운 코드 예제를 제공할 것입니다. 이 장의 뒷부분에서 XGBoost 및 LightGBM과 같은 보다 발전된 차세대 알고리즘에 대해 논의할 것입니다.

로지스틱 회귀

로지스틱 회귀는 확률을 예측하는 선형 분류기입니다. 로지스틱(시그모이드) 함수를 사용하여 출력을 두 개의(이진) 클래스에 매핑할 수 있는 확률 값으로 변환합니다. 다중 클래스 분류는 다항 로지스틱(softmax) 회귀를 통해 지원됩니다.ⁱⁱ 이 장의 뒷부분에 나오는 예제 중 하나에서 로지스틱 회귀를 사용합니다.

서포트 벡터 머신

서포트 벡터 머신은 두 클래스 사이의 여백을 최대화하는 최적의 초평면을 찾고 가능한 한 넓은 간격으로 데이터 포인트를 별도의 클래스로 나누는 방식으로 작동하는 인기 있는 알고리즘입니다. 분류 경계에 가장 가까운 데이터 포인트는 지원 벡터로 알려져 있습니다(그림 3-1).

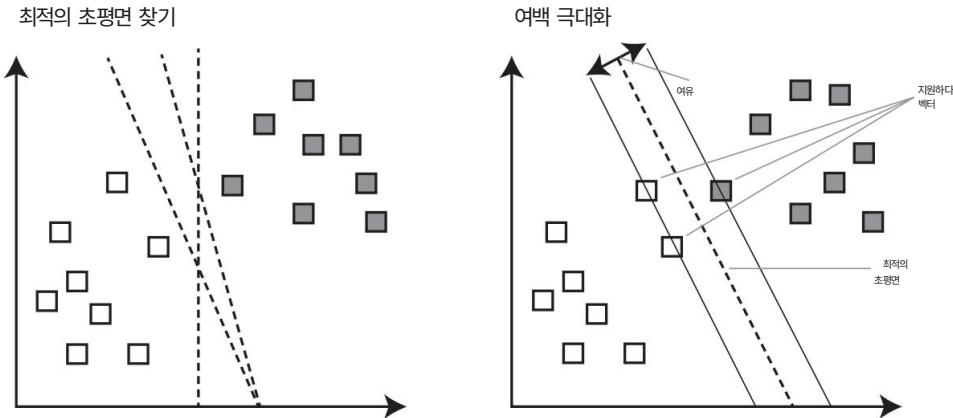


그림 3-1. 두 클래스 사이의 마진을 최대화하는 최적의 초평면 찾기ⁱⁱⁱ

나이브 베이즈

Naive Bayes는 Bayes의 정리를 기반으로 하는 간단한 다중 클래스 선형 분류 알고리즘입니다. Naive Bayes는 데이터 세트의 기능이 독립적이라고 순진하게 가정하고 기능 간의 가능한 상관 관계를 무시하기 때문에 그 이름을 얻었습니다. 이 장의 뒷부분에 나오는 감정 분석 예제에서 나이브 베이즈를 사용합니다.

다층 퍼셉트론

다층 퍼셉트론은 완전히 연결된 여러 노드 레이어로 구성된 피드포워드 인공 네트워크입니다. 입력 레이어의 노드는 입력 데이터 세트에 해당합니다.

중간 계층의 노드는 로지스틱(sigmoid) 함수를 사용하는 반면 최종 출력 계층의 노드는 softmax 함수를 사용하여 다중 클래스 분류를 지원합니다. 출력 레이어의 노드 수는 클래스 수와 일치해야 합니다.^{iv} 멀티플레이어 퍼셉트론은 7장에서 설명합니다.

의사결정나무

의사 결정 트리는 입력 변수에서 추론된 의사 결정 규칙을 학습하여 출력 변수의 값을 예측합니다.

시각적으로 결정 트리는 루트 노드가 맨 위에 있는 거꾸로 된 트리처럼 보입니다. 모든 내부 노드는 속성에 대한 테스트를 나타냅니다. 리프 노드는 클래스 레이블을 나타내고 개별 분기는 테스트 결과를 나타냅니다. 그림 3-2는 신용위험을 예측하기 위한 의사결정나무를 보여준다.

3장 지도 학습

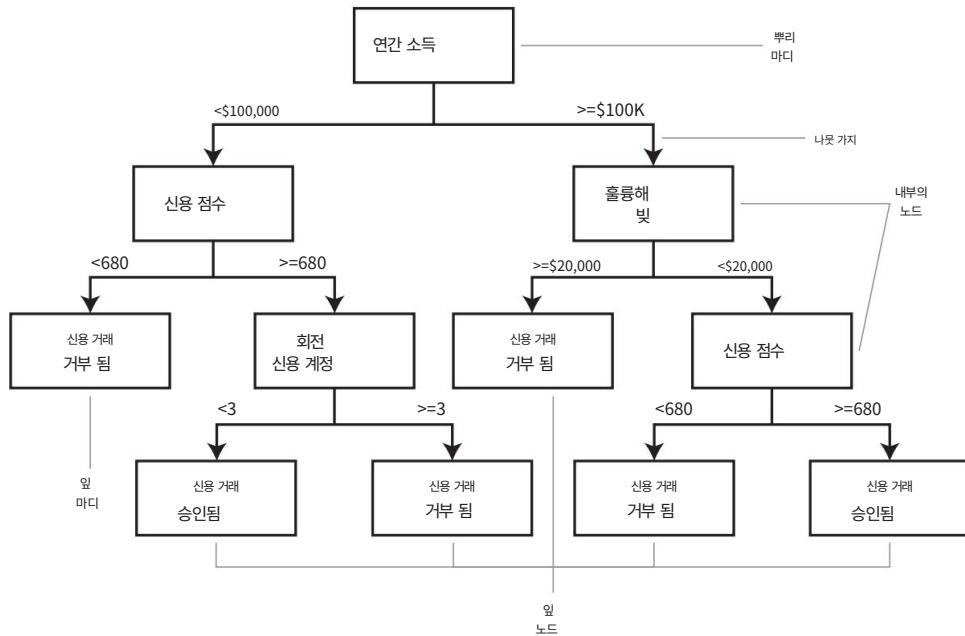


그림 3-2. 신용 위험 예측을 위한 의사 결정 트리

결정 트리는 기능 공간의 재귀적 이진 분할을 수행합니다. 정보 이득을 최대화하기 위해 불순물이 가장 많이 감소하는 분할이 가능한 분할 세트에서 선택됩니다. 정보 이득은 부모 노드의 불순물에서 자식 노드 불순물의 가중치 합을 빼서 계산됩니다. 자식 노드의 불순물이 낮을수록 정보 이득이 커집니다. 분할은 최대 트리 깊이(`maxDepth` 매개변수로 설정)에 도달하거나, `minInfoGain` 보다 큰 정보 이득을 더 이상 얻을 수 없거나, `minInstancesPerNode`가 자식 노드당 생성된 훈련 인스턴스와 같을 때까지 계속됩니다.

분류를 위한 두 가지 불순물 측정(지니 불순물 및 엔트로피)과 회귀(분산)에 대한 하나의 불순물 측정이 있습니다. 분류를 위해 Spark MLlib의 기본 불순물 측정은 지니 불순물입니다. 지니 점수는 노드의 순도를 수량화하는 메트릭입니다. 지니 점수가 0이면(노드가 순수함) 단일 데이터 클래스가 노드 내에 존재합니다. 지니 점수가 0보다 크면 노드에 다른 클래스에 속하는 데이터가 포함되어 있음을 의미합니다.

의사결정나무는 해석하기 쉽습니다. 로지스틱 회귀와 같은 선형 모델과 달리 의사 결정 트리에는 기능 확장이 필요하지 않습니다. 누락된 기능을 처리할 수 있으며 연속 및 범주 기능 모두에서 작동합니다.v 원-핫 인코딩

categorical featuresvi 는 필수 사항이 아니며 실제로 의사 결정 트리 및 트리 기반 양상들을 사용할 때 권장되지 않습니다. 원-핫 인코딩은 불균형 트리를 생성하고 우수한 예측 성능을 달성하기 위해 트리가 극도로 깊어야 합니다. 카디널리티가 높은 범주형 기능의 경우 특히 그렇습니다.

단점은 의사 결정 트리가 데이터의 노이즈에 민감하고 과적합 경향. 이러한 제한으로 인해 의사 결정 트리 자체는 실제 프로덕션 환경에서 거의 사용되지 않습니다. 오늘날 의사 결정 트리는 랜덤 포레스트 및 그라디언트 부스트 트리와 같은 보다 강력한 양상을 알고리즘의 기본 모델 역할을 합니다.

랜덤 포레스트

랜덤 포레스트는 분류 및 회귀를 위해 결정 트리 모음을 사용하는 양상을 알고리즘입니다. 낮은 편향을 유지하면서 분산을 줄이기 위해 배깅 (또는 부트스트랩 집계) 이라는 방법을 사용합니다. 배깅은 훈련 데이터의 하위 집합에서 개별 트리를 훈련합니다. 배깅 외에도 Random Forest는 기능 배깅이라는 또 다른 방법을 사용합니다. 배깅(관측의 하위 집합 사용)과 달리 기능 배깅은 기능(열)의 하위 집합을 사용합니다. 기능 배깅은 의사 결정 트리 간의 상관 관계를 줄이는 것을 목표로 합니다. 기능 배깅이 없으면 개별 트리는 특히 몇 가지 주요 기능만 있는 상황에서 매우 유사합니다.

분류의 경우 개별 트리의 출력 또는 모드의 과반수 투표가 모델의 최종 예측이 됩니다. 회귀분석의 경우 개별 트리 출력의 평균이 최종 출력이 됩니다(그림 3-3). 스파크는 각 트리가 랜덤 포레스트에서 독립적으로 훈련되기 때문에 여러 트리를 병렬로 훈련합니다. Random Forest에 대해서는 이 장의 뒷부분에서 더 자세히 설명합니다.

3장 지도 학습

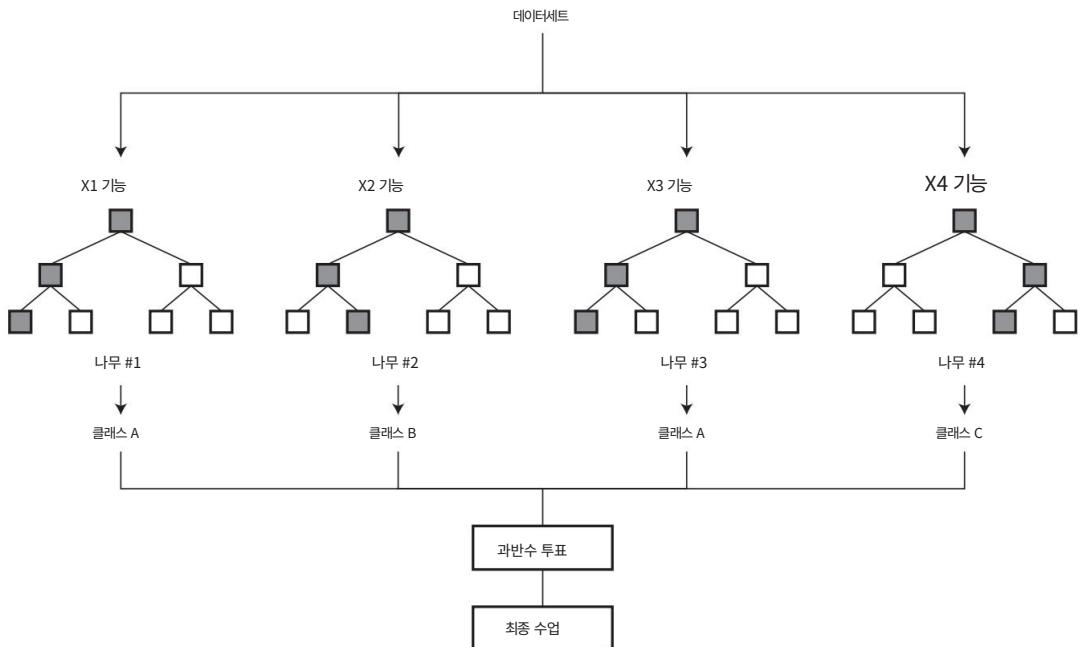


그림 3-3. 분류를 위한 랜덤 포레스트

그라디언트 부스팅 트리

GBT(Gradient-Boosted Tree)는 랜덤 포레스트와 유사한 또 다른 트리 기반 양상을 알고리즘입니다. GBT는 부스팅이라는 기술을 사용하여 약한 학습자(얕은 나무)에서 강한 학습자를 만듭니다. GBT는 각 후속 트리가 이전 트리의 오류를 줄이는 방식으로 결정 트리의 양상을 순차적으로 훈련합니다.

이것은 다음 모델에 맞추기 위해 이전 모델의 잔차를 사용하여 수행됩니다.^{viii} 이 잔차 수정 프로세스는 잔차가 완전히 최소화될 때까지 교차 검증에 의해 결정된 반복 횟수로 설정된 반복 횟수로 수행됩니다.

3장 지도 학습

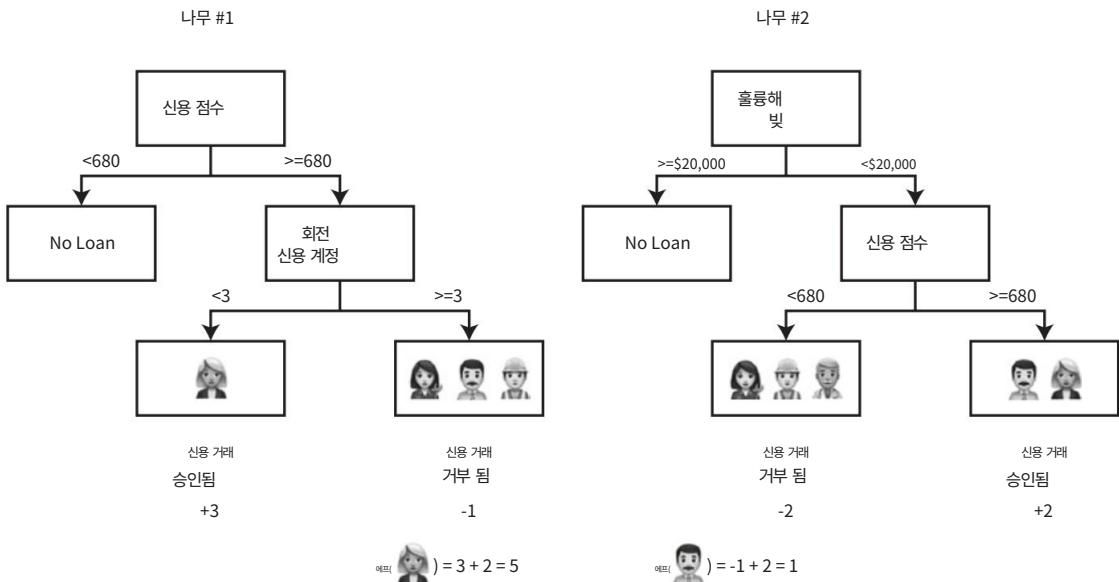


그림 3-4. GBT의 의사 결정 트리 양상을

그림 3-4는 GBT에서 의사 결정 트리 양상들이 작동하는 방식을 보여줍니다. 신용 위험 사례를 사용하여 개인은 신용도에 따라 다양한 휴가로 분류됩니다.

의사 결정 트리 내의 각 리프에는 점수가 할당됩니다. 여러 트리의 점수를 합산하여 최종 예측 점수를 얻습니다. 예를 들어, 그림 3-4는 여성에게 3점을 할당한 첫 번째 의사 결정 트리를 보여줍니다. 두 번째 트리에서는 여성에게 2점을 할당했습니다.

두 점수를 모두 더하면 여성의 최종 점수는 5가 됩니다. 의사 결정 트리는 서로를 보완합니다. 이것은 GBT의 주요 원칙 중 하나입니다. 점수를 각 리프와 연결하면 GBT에 최적화에 대한 통합 접근 방식이 제공됩니다.x

랜덤 포레스트 대 그래디언트 부스트 트리

Gradient-Boosted Trees는 순차적으로 훈련되기 때문에 일반적으로 여러 트리를 별별로 훈련할 수 있는 Random Forest보다 느리고 확장성이 떨어지는 것으로 간주됩니다. 그러나 GBT는 종종 Random Forest에 비해 더 얇은 트리를 사용하므로 GBT가 더 빨리 훈련될 수 있습니다.

GBT에서 트리 수를 늘리면 과적합 가능성이 증가하고(GBT는 더 많은 트리를 활용하여 편향을 줄임), 랜덤 포레스트에서 트리 수를 늘리면 과적합 가능성이 감소합니다(랜덤 포레스트는 더 많은 트리를 활용하여 분산 감소). 일반적으로 더 많은 트리를 추가하면 성능이 향상됩니다.

3장 지도 학습

Random Forests, GBT의 성능은 트리 수가 너무 커지기 시작하면 저하되기 시작합니다. x_i 때문에 GBT는 Random Forest보다 조정하기가 더 어려울 수 있습니다.

Gradient-Boosted Trees는 매개변수가 올바르게 조정된 경우 일반적으로 Random Forest보다 더 강력한 것으로 간주됩니다. GBT는 이전에 구성된 결정 트리를 보완하는 새로운 결정 트리를 추가하여 Random Forest.xii에 비해 더 적은 수의 트리로 더 나은 예측 정확도를 제공합니다.

XGBoost 및 LightGBM과 같은 분류 및 회귀를 위해 최근에 개발된 대부분의 최신 알고리즘은 GBT의 개선된 변형입니다. 그들은 전통적인 GBT의 한계가 없습니다.

타사 분류 및 회귀 알고리즘

수많은 오픈 소스 기여자가 Spark용 타사 기계 학습 알고리즘을 개발하는 데 시간과 노력을 투자했습니다. 핵심 Spark MLlib 라이브러리의 일부는 아니지만 Databricks(XGBoost) 및 Microsoft(LightGBM)와 같은 회사는 이러한 프로젝트를 지원하고 전 세계적으로 광범위하게 사용됩니다.

XGBoost 및 LightGBM은 현재 분류 및 회귀를 위한 차세대 기계 학습 알고리즘으로 간주됩니다. 정확성과 속도가 중요한 상황에서 사용되는 알고리즘입니다. 이 장의 뒷부분에서 두 가지 모두에 대해 논의할 것입니다. 지금은 손을 더럽히고 몇 가지 예를 살펴보겠습니다.

로지스틱 회귀를 사용한 다중 클래스 분류

로지스틱 회귀는 확률을 예측하는 선형 분류기입니다. 사용 편의성과 빠른 훈련 속도 때문에 인기가 높으며 이진 분류와 다중 클래스 분류에 모두 자주 사용됩니다. 로지스틱 회귀와 같은 선형 분류기는 그림 3-5의 첫 번째 차트와 같이 데이터에 명확한 결정 경계가 있을 때 적합합니다. 클래스가 선형으로 분리되지 않는 경우(두 번째 차트 참조) 트리 기반 앙상블과 같은 비선형 분류기를 고려해야 합니다.

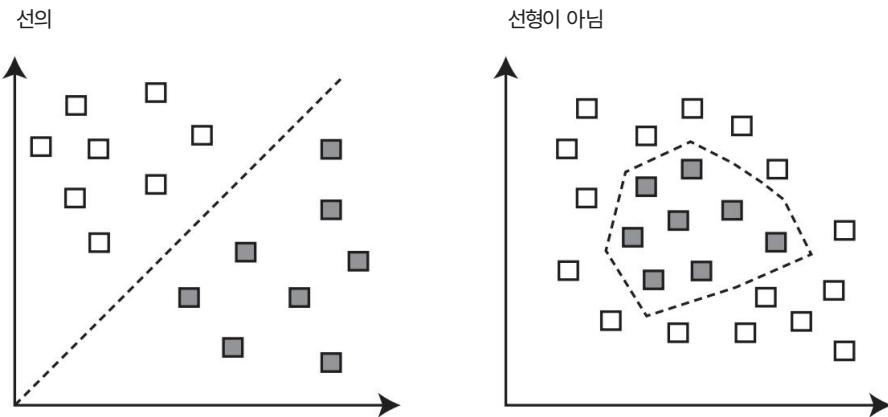


그림 3-5. 선형 대 비선형 분류 문제

예시

우리는 인기 있는 Iris 데이터 세트를 사용하여 첫 번째 예에서 다중 클래스 분류 문제를 다룰 것입니다(목록 3-1 참조). 데이터 세트에는 각각 50개의 인스턴스로 구성된 3개의 클래스가 포함되어 있으며 각 클래스는 다양한 붓꽃 식물(Iris Setosa, Iris Versicolor 및 Iris Virginica)을 나타냅니다. 그림 3-6에서 볼 수 있듯이 Iris Setosa는 Iris Versicolor 및 Iris Virginica와 선형으로 분리되지만 Iris Versicolor와 Iris Virginica는 선형으로 분리되지 않습니다. 로지스틱 회귀는 데이터 세트를 분류하는 데 여전히 적절한 작업을 수행해야 합니다.

3장 지도 학습

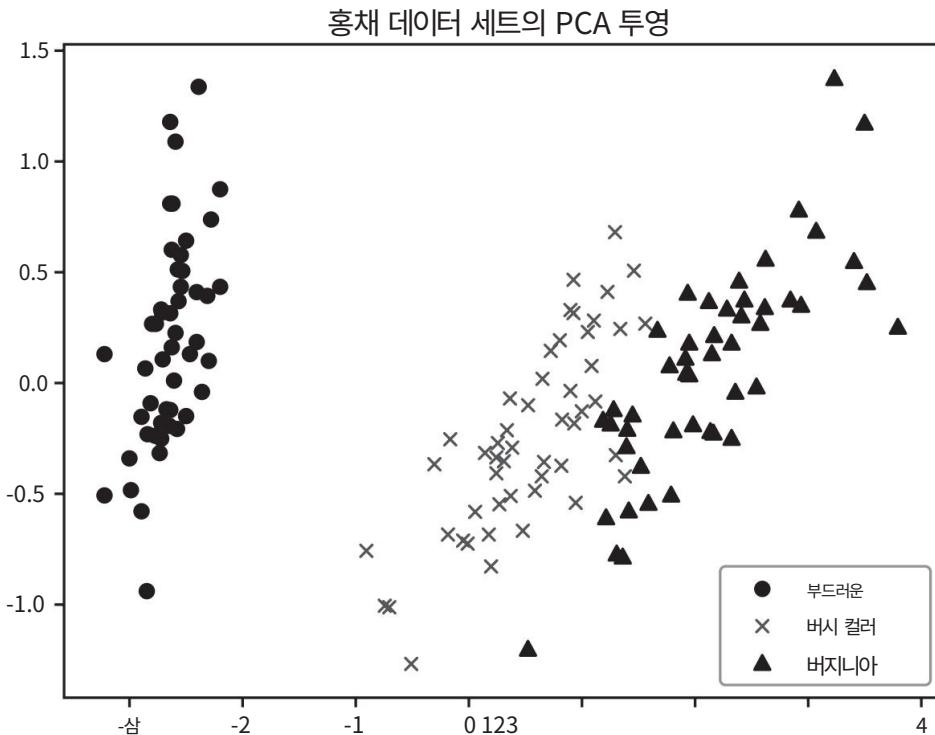


그림 3-6. Iris 데이터 세트의 주성분 분석 투영

우리의 목표는 일련의 기능이 주어지면 봇꽃 식물의 유형을 예측하는 것입니다. 데이터 세트에는 4개의 숫자 기능이 있습니다: sepal_length, sepal_width, feather_length, 그리고 feather_width(모두 센티미터 단위).

목록 3-1. 로지스틱 회귀를 사용한 분류

```
// 데이터에 대한 스키마를 생성합니다.

org.apache.spark.sql.types._ 가져오기

var irisSchema = StructType(배열(
    StructField("sepal_length", DoubleType, true),
    StructField("sepal_width", DoubleType, true),
    StructField("꽃잎 길이", DoubleType, true),
    StructField("꽃잎 너비", DoubleType, true),
    StructField("클래스", StringType, true)
))
```

// CSV 파일을 읽습니다. 방금 정의한 스키마를 사용합니다.

```
val dataDF = spark.read.format("csv")
    .option("헤더", "거짓")
    .schema(irisSchema)
    .load("/files/iris.data")
```

// 스키마를 확인합니다.

```
dataDF.printSchema
```

뿌리

```
|-- sepal_length: 이중(nullable = true)
|-- sepal_width: 이중(nullable = true)
|-- 꽃잎 길이: 이중(nullable = true)
|-- 꽃잎 너비: 이중(nullable = true)
|-- 클래스: 문자열(nullable = true)
```

// 데이터를 검사하여 올바른 형식인지 확인합니다.

```
dataDF.show
```

꽃잎 길이	꽃잎 너비	꽃잎 길이	꽃잎 너비	수업
5.1	3.5	1.4	0.2	아이리스-세토사
4.9	3.0	1.4	0.2	아이리스-세토사
4.7	3.2	1.3	0.2	아이리스-세토사
4.6	3.1	1.5	0.2	아이리스세토사
5.0	3.6	1.4	0.2	아이리스-세토사
5.4	3.9	1.7	0.4	아이리스세토사
4.6	3.4	1.4	0.3	아이리스 세토사
5.0	3.4	1.5	0.2	아이리스세토사
4.4	2.9	1.4	0.2	아이리스세토사
4.9	3.1	1.5	0.1	아이리스세토사
5.4	3.7	1.5	0.2	아이리스-세토사
4.8	3.4	1.6	0.2	아이리스-세토사
4.8	3.0	1.4	0.1	아이리스세토사
4.3	3.0	1.1	0.1	아이리스세토사

3장 지도 학습

5.8	4.0	1.2	0.2 아이리스-세토사
5.7	4.4	1.5	0.4 아이리스-세토사
5.4	3.9	1.3	0.4 아이리스세토사
5.1	3.5	1.4	0.3 아이리스세토사
5.7	3.8	1.7	0.3 아이리스 세토사
5.1	3.8	1.5	0.3 아이리스 세토사

상위 20개 행만 표시

// 데이터에 대한 요약 통계를 계산합니다. 이것은 할 수 있습니다

// 데이터 분포를 이해하는 데 도움이 됩니다.

dataDF.describe().show(5,15)

요약	sepal_length	sepal_width	꽃잎 길이	꽃잎 너비
카운트 150 150 150				
평균 5.843333333... 3.0540000000... 3.7586666666... 1.1986666666...				
stddev 0.8280661279 ... 0.4335943113 ... 1.7644204199 ... 0.7631607417 ...				
2.0 1.0 0.8 4.3				
4.4 6.9 2.5 7.9				

+-----+

| 수업|

+-----+

| 150|

| 널|

| 널|

| 아이리스 세토사 |

|아이리스-버지니카|

+-----+

// 입력 열 클래스는 현재 문자열입니다. 우리는 사용할 것입니다

// StringIndexer를 사용하여 double로 인코딩합니다. 새로운 가치

// 레이블이라는 새 출력 열에 저장됩니다.

org.apache.spark.ml.feature.StringIndexer 가져오기

```
val labelIndexer = 새로운 StringIndexer()
    .setInputCol("클래스")
    .setOutputCol("레이블")
```

값 데이터DF2 = 레이블 인덱서

```
.fit(dataDF)
.transform(dataDF)
```

// 새 DataFrame의 스키마를 확인합니다.

dataDF2.printSchema

뿌리

```
|-- sepal_length: 이중(nullable = true)
|-- sepal_width: 이중(nullable = true)
|-- 꽃잎 길이: 이중(nullable = true)
|-- 꽃잎 너비: 이중(nullable = true)
|-- 클래스: 문자열(nullable = true)
|-- 레이블: 이중(nullable = false)
```

// DataFrame에 추가된 새 열을 검사합니다.

dataDF2.show

꽃잎 길이	꽃잎 너비	꽃잎 길이	꽃잎 너비	클래스 라벨
5.1	3.5	1.4	0.2	아이리스세토사 0.0
4.9	3.0	1.4	0.2	아이리스세토사 0.0
4.7	3.2	1.3	0.2	아이리스세토사 0.0
4.6	3.1	1.5	0.2	아이리스-세토사 0.0
5.0	3.6	1.4	0.2	아이리스세토사 0.0
5.4	3.9	1.7	0.4	아이리스-세토사 0.0
4.6	3.4	1.4	0.3	아이리스 세토사 0.0
5.0	3.4	1.5	0.2	아이리스-세토사 0.0
4.4	2.9	1.4	0.2	아이리스-세토사 0.0
4.9	3.1	1.5	0.1	아이리스세토사 0.0
5.4	3.7	1.5	0.2	아이리스세토사 0.0

3장 지도 학습

4.8	3.4	1.6	0.2 아이리스-세토사 0.0
4.8	3.0	1.4	0.1 아이리스세토사 0.0
4.3	3.0	1.1	0.1 아이리스세토사 0.0
5.8	4.0	1.2	0.2 아이리스-세토사 0.0
5.7	4.4	1.5	0.4 아이리스-세토사 0.0
5.4	3.9	1.3	0.4 아이리스-세토사 0.0
5.1	3.5	1.4	0.3 아이리스 세토사 0.0
5.7	3.8	1.7	0.3 아이리스 세토사 0.0
5.1	3.8	1.5	0.3 아이리스세토사 0.0

상위 20개 행만 표시

```
// 특징을 단일 벡터로 결합
// VectorAssembler 변환기를 사용하는 열.

org.apache.spark.ml.feature.VectorAssembler 가져오기

val 기능 = Array("sepal_length", "sepal_width", "petal_length", "petal_width")
```

```
val 어셈블러 = 새로운 VectorAssembler()
    .setInputCols(기능)
    .setOutputCol("기능")

val dataDF3 = assembler.transform(dataDF2)
```

// DataFrame에 추가된 새 열을 검사합니다.

dataDF3.printSchema

뿌리

```
|-- sepal_length: 이중(nullable = true)
|-- sepal_width: 이중(nullable = true)
|-- 꽃잎 길이: 이중(nullable = true)
|-- 꽃잎 너비: 이중(nullable = true)
|-- 클래스: 문자열(nullable = true)
|-- 레이블: 이중(nullable = false)
|-- 기능: 벡터(nullable = true)
```

// DataFrame에 추가된 새 열을 검사합니다.

```
dataDF3.show
```

꽃잎 길이	꽃잎 너비	꽃잎 길이	꽃잎 너비	클래스 라벨
5.1	3.5	1.4	0.2	아이리스세토사 0.0
4.9	3.0	1.4	0.2	아이리스-세토사 0.0
4.7	3.2	1.3	0.2	아이리스세토사 0.0
4.6	3.1	1.5	0.2	아이리스세토사 0.0
5.0	3.6	1.4	0.2	아이리스세토사 0.0
5.4	3.9	1.7	0.4	아이리스세토사 0.0
4.6	3.4	1.4	0.3	아이리스 세토사 0.0
5.0	3.4	1.5	0.2	아이리스-세토사 0.0
4.4	2.9	1.4	0.2	아이리스세토사 0.0
4.9	3.1	1.5	0.1	아이리스세토사 0.0
5.4	3.7	1.5	0.2	아이리스-세토사 0.0
4.8	3.4	1.6	0.2	아이리스세토사 0.0
4.8	3.0	1.4	0.1	아이리스세토사 0.0
4.3	3.0	1.1	0.1	아이리스세토사 0.0
5.8	4.0	1.2	0.2	아이리스-세토사 0.0
5.7	4.4	1.5	0.4	아이리스-세토사 0.0
5.4	3.9	1.3	0.4	아이리스-세토사 0.0
5.1	3.5	1.4	0.3	아이리스 세토사 0.0
5.7	3.8	1.7	0.3	아이리스세토사 0.0
5.1	3.8	1.5	0.3	아이리스세토사 0.0
<hr/>				
<hr/>				
	기능			
<hr/>				
[[5.1,3.5,1.4,0.2]]				
[[4.9,3.0,1.4,0.2]]				
[[4.7,3.2,1.3,0.2]]				
[[4.6,3.1,1.5,0.2]]				
[[5.0,3.6,1.4,0.2]]				
[[5.4,3.9,1.7,0.4]]				
[[4.6,3.4,1.4,0.3]]				

3장 지도 학습

```

|[5.0,3.4,1.5,0.2]| |
|[4.4,2.9,1.4,0.2]| |
|[4.9,3.1,1.5,0.1]| |
|[5.4,3.7,1.5,0.2]| |
|[4.8,3.4,1.6,0.2]| |
|[4.8,3.0,1.4,0.1]| |
|[4.3,3.0,1.1,0.1]| |
|[5.8,4.0,1.2,0.2]| |
|[5.7,4.4,1.5,0.4]| |
|[5.4,3.9,1.3,0.4]| |
|[5.1,3.5,1.4,0.3]| |
|[5.7,3.8,1.7,0.3]| |
|[5.1,3.8,1.5,0.3]|
+-----+

```

상위 20개 행만 표시

```

// Pearson 상관 관계를 사용하여 특성과 클래스 사이의 통계적 의존성
을 // 측정해 보겠습니다. dataDF3.stat.corr("꽃잎 길이","레이블") res48: 더
블 = 0.9490425448523336

```

```

dataDF3.stat.corr("꽃잎 너비","레이블") res49: 더블
= 0.9564638238016178

```

```

dataDF3.stat.corr("sepal_length","label") res50:
Double = 0.7825612318100821

```

```

dataDF3.stat.corr("sepal_width","label") res51:
Double = -0.41944620026002677

```

// 꽃잎 길이와 꽃잎 너비는 클래스 상관 관계가 매우 높지만 // sepal_length 및 sepal_width는 클래스 상관 관계가 낮습니다.

// 2장에서 논의한 것처럼 상관 관계는 두 변수 간의 선형 관계가 // 얼마나 강한지를 평가합니다. 상관 관계를 사용하여 // 관련 기능(기능 클래스 상관 관계)을 선택하고 중복된 기능을 식별할 수 있습니다(기능 내 상관 관계).

```
// 데이터 세트를 훈련 데이터 세트와 테스트 데이터 세트로 나눕니다. 밸 시드 = 1234
```

```
val Array(trainingData, testData) = dataDF3.randomSplit(Array(0.8, 0.2), 시드)
```

```
// 이제 로지스틱 회귀를 사용하여 // 훈련 데이터 세트에 모델을 맞출 수 있습니다.
```

```
org.apache.spark.ml.classification.LogisticRegression 가져오기
```

```
val lr = 새로운 LogisticRegression()
```

```
// 훈련 데이터 세트를 사용하여 모델을 훈련합니다.
```

```
val 모델 = lr.fit(trainingData)
```

```
// 테스트 데이터 세트를 예측합니다.
```

```
val 예측 = 모델.transform(testData)
```

```
// DataFrame에 추가된 새 열을 확인합니다.
```

```
// rawPrediction, 확률, 예측.
```

```
예측.printSchema
```

뿌리

```
|-- sepal_length: double(nullable = true) |-- sepal_width:  
double(nullable = true) |-- feather_length: double(nullable =  
true) |-- feather_width: double(nullable = true) |-- 클래스: 문자  
열(nullable = true) |-- 레이블: double(nullable = false) |-- 가능:  
벡터(nullable = true) |-- rawPrediction: 벡터(nullable = true) |--  
확률: 벡터(nullable = true) | -- 예측: 이중(nullable = false)
```

```
// 예측을 검사합니다.
```

```
predicts.select("sepal_length", "sepal_width",  
"petal_length", "petal_width", "label", "prediction").show
```

3장 지도 학습

sepal_length	sepal_width	petal_length	petal_width	label	예측
4.3	3.0	1.1	0.1	0.0	0.2
4.4	2.9	1.4	0.0	0.2	0.0
4.4	3.0	1.3	0.2	0.0	0.2
4.8	3.1	1.6	0.0	0.2	0.0
5.0	3.3	1.4	0.2	0.0	0.2
5.0	3.4	1.5	0.0	1.4	1.0
5.0	3.6	1.4	0.1	0.0	0.2
5.1	3.4	1.5	0.0	1.3	1.0
5.2	2.7	3.9	2.4	2.0	1.0
5.2	4.1	1.5	1.0	1.5	1.0
5.3	3.7	1.5	1.6	1.0	1.8
5.6	2.9	3.6	2.0	1.3	1.0
5.8	2.8	5.1	1.5	2.0	2.4
6.0	2.2	4.0	2.0		1.0
6.0	2.9	4.5			1.0
6.0	3.4	4.5			1.0
6.2	2.8	4.8			2.0
6.2	2.9	4.3			1.0
6.3	2.8	5.1			1.0
6.7	3.1	5.6			2.0

상위 20개 행만 표시

// rawPrediction 및 확률 열을 검사합니다.

Prediction.select("rawPrediction","확률","예측")

.show(거짓)

원시 예측	확률	예측
[-27765.164694901094, 17727.78535517628, 10037.379339724806]		
[-24491.649758932126, 13931.526474094646, 10560.123284837473]		
[20141.806983153703, 1877.784589255676, -22019.591572409383]		

[-46255.06332259462,20994.503038678085,25260.560283916537] |
 [[25095.115980666546,110.99834659454791,-25206.114327261093] |
 [[-41011.14350152455,17036.32945903473,23974.814042489823] |
 [[20524.55747106708,1750.139974552606,-22274.697445619684] |
 [[29601.783587714817,-1697.1845083924927,-27904.599079322325]]|
 [[38919.06696252647,-5453.963471106039,-33465.10349142042] |
 [[-39965.27448934488,17725.41646382807,22239.85802551682] |
 [[-18994.667253235268,12074.709651218403,6919.957602016859] |
 [[-43236.84898013162,18023.80837865029,25213.040601481334] |
 [[-31543.179893646557,16452.928101990834,15090.251791655724] |
 [[-21666.087284218,13802.846783092147,7863.24050112584] |
 [[-24107.97243292983,14585.93668397567,9522.035748954155] |
 [[25629.52586174148,-192.40731255107312,-25437.11854919041] |
 [[-14271.522512385294,11041.861803401871,3229.660708983418] |
 [[-16548.06114507441,10139.917257827732,6408.143887246673] |
 [[22598.60355651257,938.4220993796007,-23537.025655892172] |
 [[-40984.78286289556,18297.704445848023,22687.078417047538] |

+-----+-----+

+-----+-----+
 |학률 |예측|
 +-----+-----+

[0.0,1.0,0.0]	1.0
[0.0,1.0,0.0]	1.0
[1.0,0.0,0.0]	0.0
[0.0,0.0,1.0]	2.0
[1.0,0.0,0.0]	0.0
[0.0,0.0,1.0]	2.0
[1.0,0.0,0.0]	0.0
[1.0,0.0,0.0]	0.0
[1.0,0.0,0.0]	0.0
[0.0,0.0,1.0]	2.0
[0.0,1.0,0.0]	1.0
[0.0,1.0,0.0]	1.0
[1.0,0.0,0.0]	0.0

3장 지도 학습

[[0.0,1.0,0.0]]1.0	
[0.0,1.0,0.0]]1.0	
[1.0,0.0,0.0]]0.0	
[0.0,0.0,1.0]]2.0	
+-----+-----+	

상위 20개 행만 표시

// 모델을 평가합니다. 여러 평가 메트릭을 사용할 수 있습니다.

// 다중 클래스 분류의 경우: f1(기본값), 정확도,

// weightedPrecision 및 weightedRecall.

// 평가 메트릭에 대해서는 2장에서 더 자세히 설명합니다.

org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator 가져오기

평가자 = 새로운 MulticlassificationEvaluator().setMetricName("f1")

val f1 = evaluator.evaluate(예측)

f1: 더블 = 0.958119658119658

val wp = evaluator.setMetricName("weightedPrecision").evaluate(예측)

wp: 더블 = 0.9635416666666667

val wr = evaluator.setMetricName("weightedRecall").evaluate(예측)

wr: 더블 = 0.9583333333333335

값 정확도 = evaluator.setMetricName("정확도").evaluate(예측)

정확도: 두 배 = 0.9583333333333334

로지스틱 회귀는 속도와 단순성으로 인해 첫 번째 기준 알고리즘으로 자주 사용되는 인기 있는 분류 알고리즘입니다. 프로덕션 용도의 경우, 데이터 세트에서 복잡한 비선형 관계를 캡처할 수 있는 뛰어난 정확도와 기능으로 인해 일반적으로 고급 트리 기반 앙상블이 선호됩니다.

랜덤 포레스트를 사용한 이탈 예측

Random Forest는 여러 의사 결정 트리를 기본 모델로 기반으로 구축된 강력한 앙상블 학습 알고리즘으로, 각각은 서로 다른 부트스트랩된 데이터 하위 집합에서 병렬로 훈련됩니다. 앞서 논의한 바와 같이 의사 결정나무는 과적합되는 경향이 있습니다. 랜덤 포레스트 주소

무작위로 선택된 데이터 하위 집합으로 각 의사 결정 트리를 훈련하기 위해 배깅(부트스트랩 집계)이라는 기술을 사용하여 과적합 합니다. 배깅은 모델의 분산을 줄여 과적합을 방지합니다. 랜덤 포레스트는 편향을 높이지 않고 모델의 분산을 줄입니다. 또한 각 의사 결정 트리에 대한 기능을 무작위로 선택하는 능 배깅을 수행합니다. 기능 배깅의 목표는 개별 트리 간의 상관 관계를 줄이는 것입니다.

분류의 경우 과반수 투표를 통해 최종 등급을 결정합니다. 모드(가장 자주 발생하는) 개별 의사 결정 트리에서 생성된 클래스가 최종 클래스가 됩니다. 회귀의 경우 개별 의사 결정 트리의 출력 평균이 모델의 최종 출력이 됩니다.

Random Forest는 결정 트리를 기본 모델로 사용하기 때문에 대부분의 자질, 연속 및 범주 기능을 모두 처리할 수 있으며 기능 확장 및 원-핫 인코딩이 필요하지 않습니다. Random Forest는 계층적 특성이 두 클래스를 모두 처리하도록 강제하기 때문에 불균형 데이터에서도 잘 수행됩니다. 마지막으로 Random Forest는 종속 변수와 독립 변수 간의 비선형 관계를 캡처할 수 있습니다.

Random Forest는 가장 널리 사용되는 트리 기반 양상을 알고리즘 중 하나입니다. 해석 가능성, 정확성 및 유연성으로 인한 분류 및 회귀. 그러나 Random Forest 모델 교육은 계산 집약적 일 수 있습니다(이는 Hadoop 또는 Spark와 같은 멀티코어 및 분산 환경에서 병렬화에 이상적임).

로지스틱 회귀 또는 나이브 베이즈와 같은 선형 모델에 비해 훨씬 더 많은 메모리와 컴퓨팅 리소스가 필요합니다. 또한 Random Forest는 텍스트나 게놈 데이터와 같은 고차원 데이터에 대해 성능이 떨어지는 경향이 있습니다.

참고 CSIRO Bioinformatics 팀은 원래 VariantSpark RF로 알려진 고차원 게놈 데이터 용으로 설계된 Random Forest의 확장성이 뛰어난 구현을 개발했습니다.^{xiii} VariantSpark RF는 수백만 개의 기능을 처리할 수 있으며 벤치마크^{xiv}에서 MLLib의 Random Forest 구현보다 훨씬 더 확장 가능한 것으로 나타났습니다. VariantSpark RF에 대한 자세한 내용은 CSIRO의 Bioinformatics 웹 사이트에서 확인할 수 있습니다. ReForSt는 DIBRIS – University of Genoa, Italy의 SmartLab 연구소에서 개발한 Random Forest의 확장성이 뛰어난 또 다른 구현입니다.^{xv} ReForSt는 수백만 개의 기능을 처리할 수 있으며 고전적인 Random Forest 알고리즘을 확장하는 새로운 양상을 방법인 Random Forest Rotation을 지원합니다.^{xvi}

3장 지도 학습

매개변수

Random Forest는 비교적 조정하기 쉽습니다. 몇 가지 중요한 매개변수를 적절하게 설정^{xvii} Random Forest를 성공적으로 사용하기에 충분합니다.

- **max_depth:** 트리의 최대 깊이를 지정합니다. `max_depth`에 높은 값을 설정하면 모델의 표현력이 향상될 수 있지만 너무 높게 설정하면 과적합 가능성이 증가하고 모델이 더 복잡해질 수 있습니다.
- **num_trees:** 맞출 나무의 수를 지정합니다. 증가 트리의 수는 분산을 줄여 일반적으로 정확도를 향상시킵니다. 트리 수를 늘리면 훈련 속도가 느려질 수 있습니다. 특정 지점을 넘어 더 많은 나무를 추가해도 정확도가 향상되지 않을 수 있습니다.
- **FeatureSubsetStrategy:** 각 노드에서 분할에 사용할 피처의 비율을 지정합니다. 이 매개변수를 설정하면 훈련 속도를 향상시킬 수 있습니다.
- **subsamplingRate:** 각 트리를 훈련하기 위해 선택할 데이터의 비율을 지정합니다. 이 매개변수를 설정하면 훈련 속도가 향상되고 과적합을 방지할 수 있습니다. 너무 낮게 설정하면 underfitting이 발생할 수 있습니다.

몇 가지 일반적인 지침을 제공했지만 항상 그렇듯이 이러한 매개변수에 대한 최적의 값을 결정하기 위해 매개변수 그리드 검색을 수행하는 것이 좋습니다.

Random Forest 매개변수의 전체 목록은 Spark MLlib의 온라인 설명서를 참조하십시오.

예시

이탈 예측은 은행, 보험 회사, 통신 회사, 케이블 TV 운영자 및 Netflix, Hulu, Spotify 및 Apple Music과 같은 스트리밍 서비스의 중요한 분류 사용 사례입니다. 서비스 구독을 취소할 가능성이 더 큰 고객을 예측할 수 있는 회사는 보다 효과적인 고객 유지 전략을 구현할 수 있습니다.

고객을 유지하는 것은 가치가 있습니다. 선도적인 고객 참여 분석 회사가 수행한 연구에 따르면 고객 이탈로 인해 미국 기업은 연간 1,360억 달러로 추산됩니다.^{xviii} Bain & Company에서 수행한 연구에 따르면 고객 유지율이 다음과 같이 증가하는 것으로 나타 났습니다.

단 5%만 수익을 25%에서 95%까지 증가시킵니다.^{xix} Lee Resource Inc.에서 제공한 또 다른 통계에 따르면 새로운 고객을 유치하는 데는 기존 고객을 유지하는 것보다 5배 더 많은 비용이 듭니다.^{xx} 우리의 예는 University of California Irvine Machine Learning Repository를 참조하십시오(목록 3-2 참조). 이것은 인기 있는 Kaggle dataset^{xxi}이며 온라인에서 광범위하게 사용됩니다.^{xxii}

책 전체에 있는 대부분의 예제에서 결과 DataFrame에 추가된 새 열을 볼 수 있도록 변환기와 추정기를 개별적으로 실행합니다(파이프라인에서 모두 지정하는 대신). 이렇게 하면 예제를 통해 작업할 때 "내부"에서 무슨 일이 일어나고 있는지 알 수 있습니다.

목록 3-2. 랜덤 포레스트를 사용한 이탈 예측

```
// CSV 파일을 DataFrame에 로드합니다.

val dataDF = spark.read.format("csv") .option("헤더",
    "true") .option("inferSchema",
    "true") .load("churn_data.txt")

// 스키마를 확인합니다.

dataDF.printSchema
뿌리
|-- state: string(nullable = true) |-- account_length:
double(nullable = true) |-- area_code: double(nullable = true)
|-- phone_number: string(nullable = true) |-- international_plan:
string (nullable = true) |-- voice_mail_plan: string(nullable =
true) |-- number_vmail_messages: double(nullable = true) |--_
total_day_minutes: double(nullable = true) |-- total_day_calls:
double(nullable = true) | -- total_day_charge: double(nullable = true) |--_
total_eve_minutes: double(nullable = true) |-- total_eve_calls:
double(nullable = true) |-- total_eve_charge: double(nullable = true) |--
total_night_minutes: double( nullable = true)
```

3장 지도 학습

```
|-- total_night_calls: 이중(nullable = true)  
|-- total_night_charge: 이중(nullable = true)  
|-- total_intl_minutes: 이중(nullable = true)  
|-- total_intl_calls: 이중(nullable = true)  
|-- total_intl_charge: 이중(nullable = true)  
|-- number_customer_service_calls: 이중(nullable = true)  
|-- churned: 문자열(nullable = true)
```

// 몇 개의 열을 선택합니다.

데이터DF

.select("주","전화 번호","국제 요금제","총 일_

분","변동").show

+-----+-----+-----+-----+

|주|전화 번호|국제 계획|총 일_분|이탈된|

+-----+-----+-----+-----+-----+

상위 20개 행만 표시

org.apache.spark.ml.feature.StringIndexer 가져오기

// 문자열 열 "churned"("True", "False")를 double(1,0)로 변환합니다.

val labelIndexer = 새로운

```
StringIndexer() .setInputCol("churned") .setOutputCol("label")
```

// 문자열 열 "international_plan"("yes", "no") //을 1,0을 두 배로 변환합니다.

발 intPlanIndexer = 새로운

```
StringIndexer() .setInputCol("international_plan") .setOutputCol("int_p_
```

// 우리의 기능을 선택합시다. 영역 지식은 특징 선택에 필수적입니다. // 나는 total_day_minutes
와 total_day_calls가 // 고객 이탈에 어느 정도 영향을 미친다고 생각합니다. 이 두 가지 측정항목
의 // 상당한 하락은 고객이 더 이상 서비스가 필요하지 않고 // 전화 요금제를 취소하기 직전임을 나
타낼 수 있습니다.

// 그러나 phone_number, area_code 및 state에는 // 예측 특성이 전혀 없다고 생각합니
다. // 이 장의 뒷부분에서 기능 선택에 대해 논의합니다.

val 기능 = Array("number_customer_service_calls", "total_day_
분", "total_eve_minutes", "account_length", "number_vmail_ 메시
지", "total_day_calls", "total_day_charge", "total_eve_calls", "total_eve_calls", "total_calls", "total_
", "total_intl_charge", "int_plan")

// ML 모델을 훈련하는 데 필요한 모든 기능을 포함하여 // 주어진 열 목록을 단일 벡터
열로 결합합니다.

org.apache.spark.ml.feature.VectorAssembler 가져오기

발 어셈블러 = 새로운

```
VectorAssembler() .setInputCols(기능) .setOutputCol("기능")
```

3장 지도 학습

```
// DataFrame에 레이블 열을 추가합니다.
```

```
값 데이터DF2 = 레이블 인덱서
```

```
.fit(dataDF) .transform(dataDF)
```

```
dataDF2.printSchema
```

뿌리

```
-- state: string(nullable = true) |-- account_length:  
double(nullable = true) |-- area_code: double(nullable = true)  
|-- phone_number: string(nullable = true) |-- international_plan:  
string (nullable = true) |-- voice_mail_plan: string(nullable =  
true) |-- number_vmail_messages: double(nullable = true) |--  
total_day_minutes: double(nullable = true) |-- total_day_calls:  
double(nullable = true) | -- total_day_charge: double(nullable = true) |--  
total_eve_minutes: double(nullable = true) |-- total_eve_calls:  
double(nullable = true) |-- total_eve_charge: double(nullable = true) |--  
total_night_minutes: double( nullable = true) |-- total_night_calls:  
double(nullable = true) |-- total_night_charge: double(nullable = true) |--  
total_intl_minutes: double(nullable = true) |-- total_intl_calls:  
double(nullable = true) |- - total_intl_charge: 이중(nullable = true) |--  
number_customer_service_calls: 이중(nullable = true) |-- churned: 문자열  
(nullable = true) |-- 레이블: 이중 (nullable = 거짓)
```

```
// "True"는 1로 변환되고 "False"는 0으로 변환됩니다.
```

```
dataDF2.select("churned","label").show
```

```
+-----+-----+
| 휴젓다라벨|
+-----+-----+
| 거짓| 0.0| | 거짓| 0.0|
| 참| 1.0| | 거짓| 0.0|
거짓| 0.0| | 거짓| 0.0|
거짓| 0.0| | 참| 1.0| | 거
짓| 0.0| | 거짓| 0.0| | 거
짓| 0.0| | 거짓| 0.0|
```

```
+-----+-----+
```

상위 20개 행만 표시

```
// DataFrame에 int_plan 열을 추가합니다.  
val dataDF3 = intPlanIndexer.fit(dataDF2).transform(dataDF2)
```

```
dataDF3.printSchema
```

뿌리

```
|-- state: string(nullable = true) |-- account_length:  
double(nullable = true) |-- area_code: double(nullable = true)  
|-- phone_number: string(nullable = true) |-- international_plan:  
string (nullable = true)
```

3장 지도 학습

```

|-- voice_mail_plan: 문자열(nullable = true)
|-- number_vmail_messages: 이중(nullable = true)
|-- total_day_minutes: 이중(nullable = true)
|-- total_day_calls: 이중(nullable = true)
|-- total_day_charge: 이중(nullable = true)
|-- total_eve_minutes: 이중(nullable = true)
|-- total_eve_calls: 이중(nullable = true)
|-- total_eve_charge: 이중(nullable = true)
|-- total_night_minutes: 이중(nullable = true)
|-- total_night_calls: 이중(nullable = true)
|-- total_night_charge: 이중(nullable = true)
|-- total_intl_minutes: 이중(nullable = true)
|-- total_intl_calls: 이중(nullable = true)
|-- total_intl_charge: 이중(nullable = true)
|-- number_customer_service_calls: 이중(nullable = true)
|-- churned: 문자열(nullable = true)
|-- 레이블: 이중(nullable = false)
|-- int_plan: 이중(nullable = false)

```

```
dataDF3.select("international_plan","int_plan").show
```

international_plan	int_plan
아	0.0
니	0.0
아	0.0
니 아	1.0
니	1.0
네	1.0
네	0.0
네 아	1.0
니	0.0
네 아	1.0
니	0.0
네	0.0
아니	0.0

	아니	0.0
+-----+-----+		

상위 20개 행만 표시

```
// DataFrame에 기능 벡터 열을 추가합니다.

val dataDF4 = assembler.transform(dataDF3)

dataDF4.printSchema

뿌리
|-- 상태: 문자열(nullable = true)
|-- account_length: 이중(nullable = true)
|-- area_code: 이중(nullable = true)
|-- phone_number: 문자열(nullable = true)
|-- international_plan: 문자열(nullable = true)
|-- voice_mail_plan: 문자열(nullable = true)
|-- number_vmail_messages: 이중(nullable = true)
|-- total_day_minutes: 이중(nullable = true)
|-- total_day_calls: 이중(nullable = true)
|-- total_day_charge: 이중(nullable = true)
|-- total_eve_minutes: 이중(nullable = true)
|-- total_eve_calls: 이중(nullable = true)
|-- total_eve_charge: 이중(nullable = true)
|-- total_night_minutes: 이중(nullable = true)
|-- total_night_calls: 이중(nullable = true)
|-- total_night_charge: 이중(nullable = true)
|-- total_intl_minutes: 이중(nullable = true)
|-- total_intl_calls: 이중(nullable = true)
|-- total_intl_charge: 이중(nullable = true)
|-- number_customer_service_calls: 이중(nullable = true)
|-- churned: 문자열(nullable = true)
```

3장 지도 학습

```
|-- 레이블: double(nullable = false) |-- int_plan:
double(nullable = false) |-- 기능: 벡터(nullable = true)
```

// 기능이 벡터화되었습니다.

```
dataDF4.select("기능").show(거짓)
```

```
+-----+
|특징
+-----+
[[1.0,265.1,197.4,128.0,25.0,110.0,45.07,99.0,16.78,91.0,3.0,2.7,0.0] ||
[1.0,161.6,195.5,107.0,26.0,123.0,27.47,103.0,16.62,103.0,3.0,3.7,0.0] ||
[0.0,243.4,121.2,137.0,0.0,114.0,41.38,110.0,10.3,104.0,5.0,3.29,0.0] ||
[2.0,299.4,61.9,84.0,0.0,71.0,50.9,88.0,5.26,89.0,7.0,1.78,1.0] ||
[3.0,166.7,148.3,75.0,0.0,113.0,28.34,122.0,12.61,121.0,3.0,2.73,1.0] ||
[0.0,223.4,220.6,118.0,0.0,98.0,37.98,101.0,18.75,118.0,6.0,1.7,1.0] ||
[3.0,218.2,348.5,121.0,24.0,88.0,37.09,108.0,29.62,118.0,7.0,2.03,0.0] ||
[0.0,157.0,103.1,147.0,0.0,79.0,26.69,94.0,8.76,96.0,6.0,1.92,1.0] ||
[1.0,184.5,351.6,117.0,0.0,97.0,31.37,80.0,29.89,90.0,4.0,2.35,0.0] ||
[0.0,258.6,222.0,141.0,37.0,84.0,43.96,111.0,18.87,97.0,5.0,3.02,1.0] ||
[4.0,129.1,228.5,65.0,0.0,137.0,21.95,83.0,19.42,111.0,6.0,3.43,0.0] ||
[0.0,187.7,163.4,74.0,0.0,127.0,31.91,148.0,13.89,94.0,5.0,2.46,0.0] ||
[1.0,128.8,104.9,168.0,0.0,96.0,21.9,71.0,8.92,128.0,2.0,3.02,0.0] ||
[3.0,156.6,247.6,95.0,0.0,88.0,26.62,75.0,21.05,115.0,5.0,3.32,0.0] ||
[4.0,120.7,307.2,62.0,0.0,70.0,20.52,76.0,26.11,99.0,6.0,3.54,0.0] ||
[4.0,332.9,317.8,161.0,0.0,67.0,56.59,97.0,27.01,128.0,9.0,1.46,0.0] ||
[1.0,196.4,280.9,85.0,0.27,0,139.0,33.39,90.0,23.88,75.0,4.0,3.73,0.0] ||
[3.0,190.7,218.2,93.0,0.0,114.0,32.42,111.0,18.55,121.0,3.0,2.19,0.0] ||
[1.0,189.7,212.8,76.0,0.33,0,66.0,32.25,65.0,18.09,108.0,5.0,2.7,0.0] ||
[1.0,224.4,159.5,73.0,0.0,90.0,38.15,88.0,13.56,74.0,2.0,3.51,0.0]
+-----+
```

상위 20개 행만 표시

// 데이터를 훈련 데이터와 테스트 데이터로 나눕니다.

발 시드 = 1234

```
val Array(trainingData, testData) = dataDF4.randomSplit(Array(0.8, 0.2), 시드)
```

```
trainingData.count res13:  
Long = 4009  
  
testData.count  
  
res14: 긴 = 991  
  
// 랜덤 포레스트 분류기를 생성합니다.  
  
org.apache.spark.ml.classification.RandomForestClassifier 가져오기  
  
val rf = 새로운  
    RandomForestClassifier() .setFeatureSubsetStrategy("자  
동") .setSeed(시드)  
  
// 이진 분류 평가자를 만들고 레이블 열을 // 평가에 사용하도록 설정합니다.  
  
org.apache.spark.ml.evaluation.BinaryClassificationEvaluator 가져오기  
  
값 평가자 = 새로운 BinaryClassificationEvaluator().setLabelCol("레이블")  
  
// 매개변수 그리드를 생성합니다.  
  
org.apache.spark.ml.tuning.ParamGridBuilder 가져오기  
  
val paramGrid = 새로운 ParamGridBuilder()  
    .addGrid(rf.maxBins, Array(10,  
    20,30)) .addGrid(rf.maxDepth, Array(5, 10,  
    15)) .addGrid(rf.numTrees, Array(3, 5, 100)) .addGrid  
    (rf.impurity, Array("gini", "entropy")) .build()  
  
// 파이프라인을 생성합니다.  
org.apache.spark.ml.Pipeline 가져오기  
  
val 파이프라인 = new Pipeline().setStages(Array(rf))  
  
// 교차 검증기를 생성합니다.  
  
org.apache.spark.ml.tuning.CrossValidator 가져오기  
  
val cv = 새로운 CrossValidator() .setEstimator(파  
이프라인) .setEvaluator(평가자)
```

3장 지도 학습

```

.setEstimatorParamMaps(paramGrid).setNumFolds(3)

// 이제 모델에 가장 적합한 매개변수 세트를 선택하여 // 훈련 데이터 세트를 사용하여 모델을 맞
출 수 있습니다.

val 모델 = cv.fit(trainingData)

// 이제 테스트 데이터에 대해 몇 가지 예측을 할 수 있습니다.

val 예측 = model.transform(testData)

// 모델을 평가합니다.

org.apache.spark.ml.param.ParamMap 가져오기

val pmap = ParamMap(evaluator.metricName -> "areaUnderROC")

val auc = evaluator.evaluate(예측, pmap)

auc: 더블 = 0.9270599683335483

// Random Forest 분류기는 AUC 점수가 높습니다. 테스트 // 데이터는 991개의 관찰
로 구성됩니다. 92명의 고객이 예측됩니다 // 서비스를 떠날 것입니다.

Predicts.count
res25: Long = 991

predicts.filter("prediction=1").count res26:
Long = 92

println(s"참 음수: ${predictions.select("*").where("예측 = 0 AND 레이블 = 0").count()} 참
양수: ${predictions.select("*").where("예측 = 1 AND 레이블 = 1").count()}")


참음성: 837 참양성: 81

// 우리의 테스트는 실제로 떠나는 81명의 고객을 예측하고 // 실제로 떠나지 않은 837명의 고객을 떠나지
않을 것으로 예측했습니다.

println(s"거짓음수: ${predictions.select("*").where("예측 = 0 AND 레이블 = 1").count()} 거짓
양성: ${predictions.select("*").where("예측 = 1 AND 레이블 = 0").count()}")


거짓 부정: 62 거짓 긍정: 11 128

```

// 우리의 테스트는 실제로 떠나지 않은 11명의 고객이 떠날 것으로 예측 했으며
// 또한 실제로 떠난 62명의 고객이 떠나지 않을 것이라고 예측했습니다.
// 타겟에 대한 RawPrediction 또는 Probability로 출력을 정렬할 수 있습니다.
// 가능성이 가장 높은 고객. 원시 예측 및 확률
// 각 예측에 대한 신뢰도 측정값을 제공합니다. 더 큰
// 값이 높을수록 모델이 예측에 더 확신을 갖게 됩니다.

```
Prediction.select("phone_number","RawPrediction","예측")
    .orderBy($"RawPrediction".asc)
    .show(거짓)
```

phone_number	원시 예측	예측
366-1084	[15.038138063913935,84.96186193608602] 1.0 334-6519	
[15.072688486480072,84.9273115135199]	1.0 359-5574	
[15.276260309388752,84.72373969061123]	1.0 399-7865	
[15.429722388653014,84.57027761134698]	1.0 335-2967	
[16.465107279664032,83.53489272033593]	1.0 345-9140	
[16.53288465159445,83.46711534840551]	1.0 342-6864	
[16.694165016887318,83.30583498311265]	1.0 419-1863	
[17.594670105674677,82.4053298943253]	1.0 384-7176	
[17.92764148018115,82.07235851981882]	1.0 357-1938	
[18.8550074623437,81.1449925376563]	1.0 355-6837	
[19.556608109022648,80.44339189097732]	1.0 417-1488	
[20.13305147603522,79.86694852396475]	1.0 394-5489	
[21.05074084178182,78.94925915821818]	1.0 394-7447	
[21.376663858426735,78.62333614157326]	1.0 339-6477	
[21.549262081786424,78.45073791821355]	1.0 406-7844	
[21.92209788389343,78.07790211610656]	1.0 372-4073	
[22.098599119168263,77.90140088083176]	1.0 404-4809	
[22.515513847987147,77.48448615201283]	1.0 347-8659	
[22.66840460762997,77.33159539237005]	1.0 335-1874	
[23.336632598761128,76.66336740123884]	1.0	

상위 20개 행만 표시

3장 지도 학습

기능 중요도

Random Forest(및 기타 트리 기반 양상블)에는 데이터세트에서 각 기능의 중요도를 측정하는 데 사용 할 수 있는 기본 제공 기능 선택 기능이 있습니다([목록 3-3 참조](#)).

랜덤 포레스트는 기능이 노드를 분할하기 위해 선택될 때마다 집계된 모든 트리에서 각 노드의 노드 불순율 감소의 합계를 포리스트의 트리 수로 나눈 값으로 기능 중요도를 계산합니다. Spark MLlib는 각 기능의 중요도 추정치를 반환하는 메서드를 제공합니다.

목록 3-3. 랜덤 포레스트로 기능 중요도 표시하기

```
org.apache.spark.ml.classification.RandomForestClassificationModel 가져오기
org.apache.spark.ml.PipelineModel 가져오기
```

```
val bestModel = model.bestModel
```

```
val 모델 = bestModel
```

```
.asInstanceOf[파이프라인 모
```

```
델] .stages .last .asInstanceOf[RandomForestClassificationModel]
```

```
model.feature 중요
```

```
기능 중요도: org.apache.spark.ml.linalg.Vector = (13,
[0,1,2,3,4,5,6,7,8,9,10,11,12], [ 0.20827010117447803, 0.1667170878866
0.06099491253318444, 0.008184141410796346, 0.06664053647245761,
0.007210608752126555, 0.21097011348
```

0.00644772968425685,
 0.04105403721675372,
 0.056954219262186724,
 0.09133501901837866])

기능의 수(이 예에서는 13개), 기능의 배열 인덱스 및 해당 가중치를 포함하는 벡터를 반환합니다. 표 3-1은 해당 가중치와 함께 표시된 실제 기능과 함께 더 읽기 쉬운 형식으로 출력을 보여줍니다.

보시다시피 total_day_charge, total_day_minutes 및 number_customer_service_통화가 가장 중요한 기능입니다. 말된다. 고객 서비스 전화가 많다는 것은 서비스가 여러 번 중단되거나 고객 불만이 많다는 의미일 수 있습니다. total_day_minutes 및 total_day_charge가 낮다는 것은 고객이 전화 요금제를 그렇게 많이 사용하지 않는다는 것을 나타낼 수 있습니다. 이는 곧 요금제를 취소할 준비가 되었음을 의미할 수 있습니다.

표 3-1. Telco 이탈 예측 예의 기능 중요성

색인	특징	기능 중요도
0	number_customer_service_calls	0.20827010117447803
1	total_day_minutes	0.1667170878866465
2	total_eve_minutes	0.06099491253318444
삼	계정 길이	0.008184141410796346
4	number_vmail_messages	0.06664053647245761
5	total_day_calls	0.0072108752126555
6	total_day_charge	0.21097011684691344
7	total_eve_calls	0.006902059667276019
8	total_eve_charge	0.06831916361401609
9	total_night_calls	0.00644772968425685
10	total_intl_calls	0.04105403721675372
11	total_intl_charge	0.056954219262186724
12	int_plan	0.09133501901837866

3장 지도 학습

참고 Random Forest에서 Spark MLlib의 기능 중요도 구현은 Gini 기반 중요도 또는 평균 불순률 감소(MDI)라고도 합니다.

랜덤 포레스트의 일부 구현은 정확도 기반 중요도 또는 평균 정확도 감소(MDA)로 알려진 피쳐 중요도를 계산하기 위해 다른 방법을 사용합니다.^{xxiii} 정확도 기반 중요도는 피쳐가 무작위로 치환될 때 예측 정확도의 감소를 기반으로 계산됩니다. Spark MLlib의 Random Forest 구현은 이 방법을 직접 지원하지 않지만 각 기능의 값을 한 번에 한 열씩 치환하면서 모델을 평가하여 수동으로 구현하는 것은 상당히 간단합니다.

때때로 최상의 모델이 사용하는 매개변수를 검사하는 것이 유용합니다([목록 3-4 참조](#)).

목록 3-4. 랜덤 포레스트 모델의 매개변수 추출

```
org.apache.spark.ml.classification.RandomForestClassificationModel 가져오기
org.apache.spark.ml.PipelineModel 가져오기
```

```
val bestModel = 모델
    .best모델
    .asInstanceOf[파이프라인 모델]
    .단계
    .마지막
    .asInstanceOf[RandomForest 분류 모델]

인쇄(bestModel.extractParamMap)
{
    rfc_81c4d3786152-cacheNodeIds: 거짓,
    rfc_81c4d3786152-checkpointInterval: 10,
    rfc_81c4d3786152-featureSubsetStrategy: 자동,
    rfc_81c4d3786152-featuresCol: 기능,
    rfc_81c4d3786152-불순률: 지니,
    rfc_81c4d3786152-labelCol: 레이블,
    rfc_81c4d3786152-maxBins: 10,
    rfc_81c4d3786152-maxDepth: 15,
    rfc_81c4d3786152-maxMemoryInMB: 256,
```

```

rfc_81c4d3786152-minInfoGain: 0.0,
rfc_81c4d3786152-minInstancesPerNode: 1,
rfc_81c4d3786152-num트리: 100,
rfc_81c4d3786152-predictionCol: 예측,
rfc_81c4d3786152-probabilityCol: 확률,
rfc_81c4d3786152-rawPredictionCol: rawPrediction,
rfc_81c4d3786152-시드: 1234,
rfc_81c4d3786152-subsamplingRate: 1.0
}

```

XGBoost4J-Spark를 사용한 eXtreme Gradient Boosting

그라디언트 부스팅 알고리즘은 분류 및 회귀를 위한 가장 강력한 기계 학습 알고리즘 중 일부입니다. 현재 다양한 그라디언트 부스팅 알고리즘 구현이 있습니다. 인기 있는 구현에는 AdaBoost 및 CatBoost(최근 Yandex의 오픈 소스 그라디언트 부스팅 라이브러리)가 있습니다.

Spark MLlib에는 자체 GBT(그라디언트 부스트 트리) 구현도 포함되어 있습니다.

XGBoost(eXtreme Gradient Boosting)는 현재 사용 가능한 최고의 그라디언트 부스트 트리 구현 중 하나입니다. 2014년 3월 27일에 Tianqi Chen이 연구 프로젝트로 출시한 XGBoost는 분류 및 회귀를 위한 지배적인 기계 학습 알고리즘이 되었습니다. 효율성과 확장성을 위해 설계된 병렬 트리 부스팅 기능은 다른 트리 기반 암상률 알고리즘보다 훨씬 빠릅니다. 높은 정확도로 인해 XGBoost는 여러 기계 학습 대회에서 우승하여 인기를 얻었습니다. 2015년에는 Kaggle에서 수상한 29개의 솔루션 중 17개가 XGBoost를 사용했습니다. KDD Cup [2015xxiv](#)의 상위 10개 솔루션은 모두 XGBoost를 사용했습니다.

XGBoost는 그라디언트 부스팅의 일반 원리를 사용하여 설계되었습니다. 약한 학습자를 강한 학습자로. 그러나 그라디언트 부스트 트리가 순차적으로 구축되는 동안 데이터에 서 천천히 학습하여 후속 반복에서 예측을 개선하는 반면 XGBoost는 트리를 병렬로 구축합니다. XGBoost는 내장된 정규화를 통해 모델 복잡성을 제어하고 과적합을 줄임으로써 더 나은 예측 성능을 생성합니다.

연속적인 특징에 대한 최상의 분할점을 찾을 때 분할점을 찾기 위해 근사 알고리즘을 사용합니다.[xxv](#)

근사 분할 방법은 이산 빈을 사용하여 연속적인 특징을 버킷화하고, 모델 학습 속도를 크게 향상시킵니다. XGBoost에는 히스토그램 기반 알고리즘을 사용하여 훨씬 더 효율적인 트리 성장 방법이 포함되어 있습니다.

3장 지도 학습

연속적인 특징을 개별 빈으로 베kt화하는 방법. 그러나 근사 방법은 반복당 새로운 빈 세트를 생성하지만 히스토그램 기반 접근 방식은 여러 반복에서 빈을 재사용합니다.

이 접근 방식을 사용하면 빈과 상위 및 형제 히스토그램 빼기를 캐시하는 기능과 같이 근사 방법으로는 달성할 수 없는 추가 최적화가 가능합니다 .xxvi 정렬 작업을 최적화하기 위해 XGBoost는 정렬된 데이터를 인메모리 블록 단위로 저장합니다. 정렬 블록은 병렬 CPU 코어에 의해 효율적으로 분산되고 수행될 수 있습니다. XGBoost는 가중 분위수 스케치 알고리즘을 통해 가중 데이터를 효과적으로 처리할 수 있고, 희소 데이터를 효율적으로 처리할 수 있으며, 캐시를 인식하고, 데이터가 메모리에 맞지 않도록 대용량 데이터 세트에 디스크 공간을 활용하여 코어 외 컴퓨팅을 지원합니다.

XGBoost4J-Spark 프로젝트는 XGBoost를 Spark로 이식하기 위해 2016년 말에 시작되었습니다. XGBoost4J-Spark는 Spark의 확장성이 뛰어난 분산 처리 엔진을 활용하며 Spark MLlib의 DataFrame/Dataset 추상화와 완벽하게 호환됩니다. XGBoost4J-Spark는 Spark MLlib 파이프라인에 원활하게 포함되고 Spark MLlib의 변환기 및 추정기와 통합될 수 있습니다.

참고 XGBoost4J-Spark에는 Apache Spark 2.4 이상이 필요합니다. <http://spark.apache.org> 에서 직접 Spark를 설치하는 것이 좋습니다 . XGBoost4J-Spark는 Cloudera, Hortonworks 또는 MapR과 같은 다른 공급업체의 타사 Spark 배포판과의 작동을 보장하지 않습니다. 자세한 내용은 공급업체 설명서를 참조하십시오 .xxvii

매개변수

XGBoost는 Random Forest보다 훨씬 많은 매개변수를 가지고 있으며 일반적으로 더 많은 조정이 필요합니다. 처음에 가장 중요한 매개변수에 초점을 맞추면 XGBoost를 시작할 수 있습니다. 나머지는 알고리즘에 익숙해지면 배울 수 있습니다.

- max_depth: 트리의 최대 깊이를 지정합니다. max_depth에 높은 값을 설정하면 과적합 가능성이 증가하고 모델이 더 복잡해질 수 있습니다.

- n_estimators: 맞출 나무의 수를 지정합니다. 일반적으로

말하자면 값이 클수록 좋습니다. 이 매개변수를 너무 높게 설정하면 훈련 속도에 영향을 미칠 수 있습니다. 특정 지점을 넘어 더 많은 나무를 추가해도 정확도가 향상되지 않을 수 있습니다. 기본값은 100 .xxviii 로 설정됩니다.

- **sub_sample:** 각 트리에 대해 선택할 데이터의 비율을 지정합니다. 이 매개변수를 설정하면 훈련 속도를 높이고 과적합을 방지하는 데 도움이 됩니다. 너무 낮게 설정하면 underfitting이 발생할 수 있습니다.
- **colsample_bytree:** 열의 비율을 지정합니다.
각 나무에 대해 무작위로 선택됩니다. 이 매개변수를 설정하면 훈련 속도를 높이고 과적합을 방지하는 데 도움이 됩니다. 관련 매개변수에는 colsample_bylevel 및 colsample_bynode가 있습니다.
- **목표:** 학습 과제와 학습 목표를 지정합니다. 그것 예측할 수 없는 결과나 낮은 정확도를 피하기 위해 이 매개변수에 올바른 값을 설정하는 것이 중요합니다. XGBClassifier는 바이너리 분류를 위해 기본적으로 binary:logistic 을 사용하는 반면 XGBRegressor는 reg:squarederror로 기본 설정되어 있습니다. 다른 값에는 multi:softmax 가 포함됩니다. 및 다중 클래스 분류를 위한 multi:softprob ; rank:pairwise, rank:ndcg 및 rank:map 은 순위 지정용입니다. 몇 가지를 언급하자면 Cox 비례 위험 모델을 사용하는 생존 회귀에 대한 생존 :cox 입니다.
- **learning_rate(eta):** learning_rate는 축소 요인으로 사용됩니다.
학습 속도를 늦추는 것을 목표로 각 부스팅 단계 후에 기능 가중치를 줄입니다. 이 매개변수는 과적합을 제어하는 데 사용됩니다. 값이 낮을수록 더 많은 나무가 필요합니다.
- **n_jobs:** XGBoost에서 사용하는 병렬 스레드 수를 지정합니다(n_thread가 더 이상 사용되지 않는 경우 이 매개변수를 대신 사용).
이는 매개변수를 사용하는 방법에 대한 일반적인 지침일 뿐입니다. 이러한 매개변수에 대한 최적의 값을 결정하기 위해 매개변수 그리드 검색을 수행하는 것이 좋습니다. XGBoost 매개변수의 전체 목록은 XGBoost의 온라인 설명서를 참조하십시오.

참고 스칼라의 변수 명명 규칙과 일관성을 유지하기 위해 XGBoost4J Spark는 기본 매개변수 세트와 이러한 매개변수의 카멜 케이스 변형(예: max_depth 및 maxDepth)을 모두 지원합니다.

3장 지도 학습

예시

우리는 동일한 telco churn 데이터 세트와 이전 Random Forest 예제의 대부분의 코드를 재사용할 것입니다(목록 3-5 [참조](#)). 이번에는 파이프라인을 사용하여 변환기와 추정기를 함께 묶습니다.

목록 3-5. XGBoost4J-Spark를 사용한 이탈 예측

```
// XGBoost4J-Spark는 외부 패키지로 제공됩니다.  
// 스파크 쉘을 시작합니다. XGBoost4J-Spark 패키지를 지정합니다.
```

```
스파크 쉘 --패키지 ml.dmlc:xgboost4j-spark:0.81
```

```
// CSV 파일을 DataFrame에 로드합니다.
```

```
val dataDF = spark.read.format("csv") .option("헤더",  
    "true") .option("inferSchema",  
    "true") .load("churn_data.txt")
```

```
// 스키마를 확인합니다.
```

```
dataDF.printSchema
```

뿌리

```
|-- state: string(nullable = true) |-- account_length:  
double(nullable = true) |-- area_code: double(nullable = true)  
|-- phone_number: string(nullable = true) |-- international_plan:  
string (nullable = true) |-- voice_mail_plan: string(nullable =  
true) |-- number_vmail_messages: double(nullable = true) |--  
total_day_minutes: double(nullable = true) |-- total_day_calls:  
double(nullable = true) |-- total_day_charge: double(nullable = true) |--  
total_eve_minutes: double(nullable = true) |-- total_eve_calls:  
double(nullable = true) |-- total_eve_charge: double(nullable = true) |--  
total_night_minutes: double( nullable = true)
```

3장 지도 학습

예시

우리는 동일한 telco churn 데이터 세트와 이전 Random Forest 예제의 대부분의 코드를 재사용할 것입니다(목록 3-5 [참조](#)). 이번에는 파이프라인을 사용하여 변환기와 추정기를 함께 묶습니다.

목록 3-5. XGBoost4J-Spark를 사용한 이탈 예측

```
// XGBoost4J-Spark는 외부 패키지로 제공됩니다.  
// 스파크 쉘을 시작합니다. XGBoost4J-Spark 패키지를 지정합니다.
```

```
스파크 쉘 --패키지 ml.dmlc:xgboost4j-spark:0.81
```

```
// CSV 파일을 DataFrame에 로드합니다.
```

```
val dataDF = spark.read.format("csv") .option("헤더",  
    "true") .option("inferSchema",  
    "true") .load("churn_data.txt")
```

```
// 스키마를 확인합니다.
```

```
dataDF.printSchema
```

뿌리

```
|-- state: string(nullable = true) |-- account_length:  
double(nullable = true) |-- area_code: double(nullable = true)  
|-- phone_number: string(nullable = true) |-- international_plan:  
string (nullable = true) |-- voice_mail_plan: string(nullable =  
true) |-- number_vmail_messages: double(nullable = true) |--  
total_day_minutes: double(nullable = true) |-- total_day_calls:  
double(nullable = true) |-- total_day_charge: double(nullable = true) |--  
total_eve_minutes: double(nullable = true) |-- total_eve_calls:  
double(nullable = true) |-- total_eve_charge: double(nullable = true) |--  
total_night_minutes: double( nullable = true)
```

```

|-- total_night_calls: 이중(nullable = true)
|-- total_night_charge: 이중(nullable = true)
|-- total_intl_minutes: 이중(nullable = true)
|-- total_intl_calls: 이중(nullable = true)
|-- total_intl_charge: 이중(nullable = true)
|-- number_customer_service_calls: 이중(nullable = true)
|-- churned: 문자열(nullable = true)

```

// 몇 개의 열을 선택합니다.

```
dataDF.select("state","phone_number","international_plan","churned").show
```

 주 전화번호 국제계획 변경됨 			
KS	382-4657	아니	거짓
오	371-7191	아니	거짓
뉴저지	358-1921	아니	거짓
오	375-9999	네	거짓
확인	330-6626	네	거짓
알	391-8027	네	거짓
MA	355-9993	아니	거짓
모	329-9001	네	거짓
LA	335-4719	아니	거짓
WV	330-8173	네	거짓
인	329-6603	아니	참
리	344-9403	아니	거짓
이아	363-1107	아니	거짓
MT	394-8006	아니	거짓
이아	366-9238	아니	거짓
뉴욕	351-7269	아니	참
아이디	350-8884	아니	거짓
VT	386-2923	아니	거짓
버지니아	356-2992	아니	거짓
텍사스	373-2782	아니	거짓

상위 20개 행만 표시

3장 지도 학습

```
org.apache.spark.ml.feature.StringIndexer 가져오기
```

```
// String "churned" 열("True", "False")을 double(1,0)로 변환합니다.
```

```
val labelIndexer = 새로운
```

```
StringIndexer() .setInputCol("churned") .setOutputCol("label")
```

```
// 문자열 "international_plan"("no", "yes") 열을 double(1,0)로 변환합니다.
```

```
발 intPlanIndexer = 새로운
```

```
StringIndexer() .setInputCol("international_plan") .setOutputCol("int_plan")
```

```
// 모델 피팅을 위해 선택할 피처를 지정합니다.
```

```
val 기능 = Array("number_customer_service_calls", "total_day_분", "total_eve_minutes", "account_length", "number_vmail_메시지", "total_day_calls", "total_day_charge", "total_eve_calls", "total_eve_calls", "total_calls", "total_night", "total_intl_charge", "int_plan")
```

```
// 기능을 단일 벡터 열로 결합합니다.
```

```
org.apache.spark.ml.feature.VectorAssembler 가져오기
```

```
발 어셈블러 = 새로운 VectorAssembler() .setInputCols(기능) .setOutputCol("기능")
```

```
// 데이터를 훈련 데이터와 테스트 데이터로 나눕니다.
```

```
발 시드 = 1234
```

```
val Array(trainingData, testData) = dataDF.randomSplit(Array(0.8, 0.2), 시드)
```

```
// XGBoost 분류기를 생성합니다.
```

```
ml.dmlc.xgboost4j.scala.spark.XGBoostClassifier 가져오기
```

```
ml.dmlc.xgboost4j.scala.spark.XGBoostClassificationModel 가져오기
```

```

val xgb = 새로운
    XGBoostClassifier() .setFeaturesCol("기
    능") .setLabelCol("레이블")

// XGBClassifier의 목표 매개변수는 기본적으로 바이너리:로지스틱 으로 설정 되며 // 이는 이 예제에서 원하
는 학습 작업 및 목표입니다. // (이진 분류). 작업에 따라 // 올바른 학습 작업과 목표를 설정하는 것을 잊지 마십
시오.

org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator 가져오기

val 평가자 = 새로운 MulticlassClassificationEvaluator(). setLabelCol("레이블")

org.apache.spark.ml.tuning.ParamGridBuilder 가져오기

val paramGrid = 새로운 ParamGridBuilder()
    .addGrid(xgb.maxDepth, Array(3,
    8)) .addGrid(xgb.eta, Array(0.2, 0.6)) .build()

// 이번에는 파이프라인의 모든 단계를 지정합니다.

org.apache.spark.ml 가져오기.{ 파이프라인, 파이프라인스테이지 }

val 파이프라인 = new
    Pipeline() .setStages(Array(labelIndexer, intPlanIndexer, 어셈블러, xgb))

// 교차 검증기를 생성합니다.

org.apache.spark.ml.tuning.CrossValidator 가져오기

val cv = 새로운
    CrossValidator() .setEstimator(파
    이프라인) .setEvaluator(평가
    자) .setEstimatorParamMaps(paramGrid) .setNumFolds(3)

// 이제 훈련 데이터를 사용하여 모델을 맞출 수 있습니다. // 교차 검증을 실행하여 최상의 매개변수 세
트를 선택합니다.

val 모델 = cv.fit(trainingData)

```

3장 지도 학습

// 이제 테스트 데이터에 대해 몇 가지 예측을 할 수 있습니다.

```
val 예측 = model.transform(testData)
```

```
예측.printSchema
```

뿌리

```
|-- state: string(nullable = true) |-- account_length:  
double(nullable = true) |-- area_code: double(nullable = true)  
|-- phone_number: string(nullable = true) |-- international_plan:  
string (nullable = true) |-- voice_mail_plan: string(nullable =  
true) |-- number_vmail_messages: double(nullable = true) |--  
total_day_minutes: double(nullable = true) |-- total_day_calls:  
double(nullable = true) | -- total_day_charge: double(nullable = true) |--  
total_eve_minutes: double(nullable = true) |-- total_eve_calls:  
double(nullable = true) |-- total_eve_charge: double(nullable = true) |--  
total_night_minutes: double( nullable = true) |-- total_night_calls:  
double(nullable = true) |-- total_night_charge: double(nullable = true)  
|-- total_intl_minutes: double(nullable = true) |-- total_intl_calls:  
double(nullable = true) | - total_intl_charge: 이중(nullable = true) |--  
number_customer_service_calls: 이중(nullable = true) |-- churned: 문자열  
(nullable = true) |-- 레이블: 이중 (nullable = false) |-- int_plan:  
double(nullable = false) |-- 기능: 벡터(nullable = true) |-- rawPrediction: 벡  
터(nullable = true) |-- 확률: 벡터(nullable = true) | -- 예측: 이중(nullable =  
false)
```

// 모델을 평가해 봅시다.

```
val auc = evaluator.evaluate(예측) auc: Double =  
0.9328044307445879
```

// XGBoost4J-Spark에 의해 생성된 AUC 점수는 이전 Random Forest 예제와 비교하여 // 약간 더 좋습니다.
XGBoost4J-Spark는 또한 // 이 데이터 세트를 훈련하는 데 Random Forest보다 빠릅니다.

// Random Forest와 마찬가지로 XGBoost를 사용하면 기능 중요도를 추출할 수 있습니다.

```
ml.dmlc.xgboost4j.scala.spark.XGBoostClassificationModel 가져오기
```

```
org.apache.spark.ml.PipelineModel 가져오기
```

```
val bestModel = model.bestModel
```

```
val 모델 = bestModel
```

```
.asInstanceOf[파이프라인 모델] .stages .last
```

```
.asInstanceOf[XGBoost 분류 모델]
```

// 특징 중요도를 추출하기 위해 getFeatureScore 메소드를 실행합니다.

```
model.nativeBooster.getFeatureScore()
```

```
res9: scala.collection.mutable.Map[문자열, 정수] = Map(f7 -> 4, f9 -> 7, f10 -> 2, f12 -> 4, f11 -> 8, f0 -> 5,  
f1 -> 19, f2 -> 17, f3 -> 10, f4 -> 2, f5 -> 3)
```

// 이 메서드는 기능에 대한 키 매핑과 // 배열 인덱스 및 기능 중요도 점수에 해당하는 값이 있는 맵을 반환합니다.

3장 지도 학습

표 3-2. XGBoost4J-Spark를 사용한 기능 중요도

색인	특징	기능 중요도
0	number_customer_service_calls	2
1	total_day_minutes	15
2	total_eve_minutes	10
삼	계정 길이	삼
4	number_vmail_messages	2
5	total_day_calls	삼
6	total_day_charge	생략
7	total_eve_calls	2
8	total_eve_charge	생략
9	total_night_calls	2
10	total_intl_calls	2
11	total_intl_charge	1
12	int_plan	5

출력에 두 개의 열, 특히 total_day_charge(f6) 및 total_eve_charge(f8)가 누락되었음을 알 수 있습니다. 이는 XGBoost가 모델의 예측 정확도를 향상시키는 데 효과가 없다고 간주되는 기능입니다(표 3-2 참조). 최소한 하나의 분할에서 사용된 기능만 XGBoost 기능 중요도 출력으로 만들습니다. 몇 가지 가능한 설명이 있습니다. 삭제된 기능의 분산이 매우 낮거나 0임을 의미할 수 있습니다. 또한 이 두 가지 기능이 다른 기능과 높은 상관 관계가 있음을 의미할 수도 있습니다.

XGBoost의 기능 중요도 출력을 이전 Random Forest 예제와 비교할 때 주의해야 할 몇 가지 흥미로운 사항이 있습니다. 이전 Random Forest 모델은 number_customer_service_calls를 가장 중요한 기능 중 하나로 간주했지만 XGBoost는 가장 덜 중요한 기능 중 하나로 순위를 매겼습니다. 마찬가지로 이전 Random Forest 모델은 total_day_charge를 가장 중요한 기능으로 간주하지만 XGBoost는 중요도가 낮아 출력에서 완전히 생략했습니다(Listing 3-6 참조).

목록 3-6. XGBoost4J-Spark 모델의 매개변수 추출

```

val bestModel = 모델
    .best모델
    .asInstanceOf[파이프라인 모
델] .stages .last

    .asInstanceOf[XGBoost 분류 모델]

인쇄(bestModel.extractParamMap) {

    xgbc_9b95e70ab140 알파 : 0.0
    xgbc_9b95e70ab140-baseScore : 0.5
    xgbc_9b95e70ab140-checkpointInterval : -1
    xgbc_9b95e70ab140-checkpointPath :
    xgbc_9b95e70ab140-colsampleBylevel : 1.0
    xgbc_9b95e70ab140-colsampleBytree : 1.0
    xgbc_9b95e70ab140-customEval : NULL,
    xgbc_9b95e70ab140-customObj : NULL,
    xgbc_9b95e70ab140-ETA : 0.2 xgbc_9b95e70ab140-
evalMetric : 오류 xgbc_9b95e70ab140-featuresCol : 기
능 xgbc_9b95e70ab140 감마 0.0, xgbc_9b95e70ab140-
growPolicy : 깊이 방향, xgbc_9b95e70ab140-labelCol :
레이블 xgbc_9b95e70ab140 람다 : 1.0 xgbc_9b95e70ab140-
lambdaBias : 0.0 xgbc_9b95e70ab140-maxBin : 16-
xgbc_9b95e70ab140 maxDeltaStep : 0.0
xgbc_9b95e70ab140 MAXDEPTH-8, xgbc_9b95e70ab140-
minChildWeight : 1.0 xgbc_9b95e70ab140 소실 :
xgbc_9b95e70ab140-nthread 트리 : NaN의
xgbc_9b95e70ab140 normalizeType-1-
xgbc_9b95e70ab140 numEarlyStoppingRounds : 0,
}

```

3장 지도 학습

```

xgbc_9b95e70ab140-num라운드: 1,
xgbc_9b95e70ab140-num일꾼: 1,
xgbc_9b95e70ab140-목표: reg:선형,
xgbc_9b95e70ab140-predictionCol: 예측,
xgbc_9b95e70ab140-probabilityCol: 확률,
xgbc_9b95e70ab140-rateDrop: 0.0,
xgbc_9b95e70ab140-rawPredictionCol: rawPrediction,
xgbc_9b95e70ab140-sampleType: 균일,
xgbc_9b95e70ab140-scalePosWeight: 1.0,
xgbc_9b95e70ab140-시드: 0,
xgbc_9b95e70ab140-무음: 0,
xgbc_9b95e70ab140-sketchEps: 0.03,
xgbc_9b95e70ab140-skipDrop: 0.0,
xgbc_9b95e70ab140-서브샘플: 1.0,
xgbc_9b95e70ab140-timeoutRequestWorkers: 1800000,
xgbc_9b95e70ab140-trackerConf: TrackerConf(0, 파이썬),
xgbc_9b95e70ab140-trainTestRatio: 1.0,
xgbc_9b95e70ab140-treeLimit: 0,
xgbc_9b95e70ab140-treeMethod: 자동,
xgbc_9b95e70ab140-useExternalMemory: 거짓
}

```

LightGBM: Microsoft의 빠른 그라디언트 부스팅

수년 동안 XGBoost는 분류 및 회귀에 대해 모두가 가장 좋아하는 알고리즘이었습니다. 최근 LightGBM이 왕좌를 향한 새로운 도전자로 떠올랐습니다.

XGBoost와 유사한 비교적 새로운 트리 기반 그라디언트 부스팅 변형입니다.

LightGBM은 Microsoft의 DMTK(Distributed Machine Learning Toolkit) 프로젝트의 일부로 2016년 10월 17일에 출시되었습니다. 빠르고 분산되도록 설계되어 훈련 속도가 빨라지고 메모리 사용량이 적습니다. GPU 및 병렬 학습과 대규모 데이터 세트를 처리하는 기능을 지원합니다. LightGBM은 공개 데이터 세트에 대한 여러 벤치마크 및 실험에서 XGBoost보다 훨씬 빠르고 정확도가 높은 것으로 나타났습니다.

참고 LightGBM은 MMLSpark(Machine Learning for Apache Spark) 에코 시스템의 일부로 Spark에 이식되었습니다. Microsoft는 Microsoft Cognitive Toolkit, OpenCV 및 LightGBM과 같은 Apache Spark 에코시스템과의 원활한 통합을 통해 데이터 과학 및 딥 러닝 도구를 적극적으로 개발해 왔습니다. MMLSpark에는 Python 2.7 또는 3.5 이상, Scala 2.11 및 Spark 2.3 이상이 필요합니다.

LightGBM은 XGBoost에 비해 몇 가지 장점이 있습니다. 히스토그램을 사용하여 연속적인 기능을 개별 빈으로 묶습니다. 이를 통해 LightGBM은 메모리 사용량 감소, 각 분할에 대한 이득 계산 비용 감소, 병렬 학습을 위한 통신 비용 감소와 같은 XGBoost(기본적으로 트리 학습에 사전 정렬 기반 알고리즘을 사용함)에 비해 몇 가지 성능 이점을 제공합니다. LightGBM은 노드의 히스토그램을 계산하기 위해 형제 및 부모에 대해 히스토그램 빼기를 수행하여 추가 성능 향상을 달성합니다. 벤치마크 온라인 쇼 LightGBM은 일부 작업에서 XGBoost(비닝 제외)보다 11배 ~ 15배 빠릅니다.^{xxix}

LightGBM은 일반적으로 나무를 잎사귀 방향(최상 우선)으로 성장시켜 정확도 측면에서 XGBoost를 능가합니다. 의사 결정 트리를 훈련하기 위한 두 가지 주요 전략, 레벨별 및 잎별 전략이 있습니다(그림 3-7 참조). 레벨별 트리 성장은 대부분의 트리 기반 앙상블(XGBoost 포함)에 대한 의사 결정 트리를 성장시키는 전통적인 방법입니다. LightGBM은 잎사귀 성장 전략을 도입했습니다. 레벨별 성장과 대조적으로 잎사귀 성장은 일반적으로 더 빨리 수렴되고 xxx 는 더 낮은 손실을 달성합니다.^{xxxii}

참고 리프 단위 성장은 작은 데이터 세트에 과적합되는 경향이 있습니다. LightGBM에서 max_depth 매개변수를 설정하여 트리 깊이를 제한하는 것이 좋습니다. max_depth가 설정되어 어도 나무는 여전히 잎사귀로 자랍니다.^{xxxii} 이 장의 뒷부분에서 LightGBM 매개변수 조정에 대해 설명합니다.

3장 지도 학습

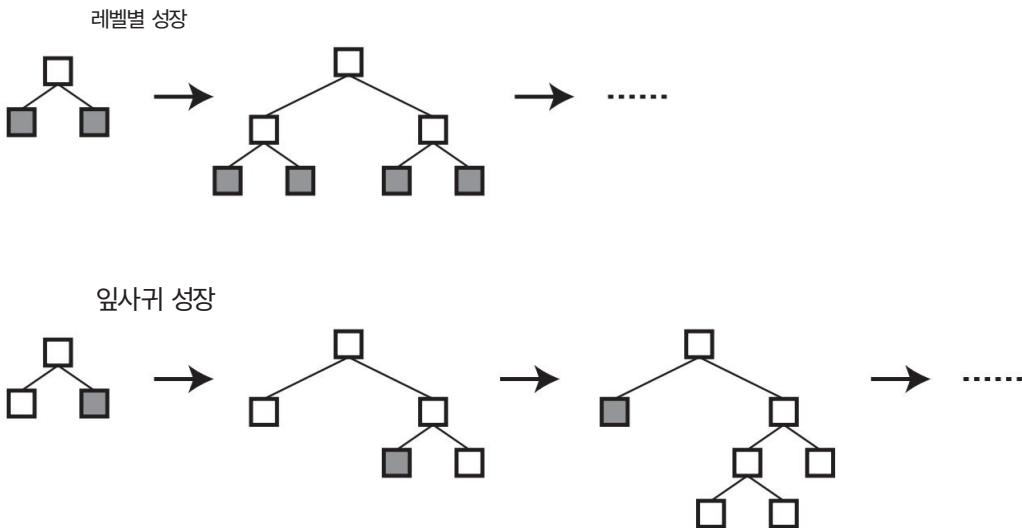


그림 3-7. 수준별 성장 대 잎별 성장

참고 XGBoost는 이후로 잎별 트리 성장 전략과 히스토그램을 사용하여 연속 기능을 개별 빈으로 버킷화하는 등 LightGBM이 개척한 많은 최적화를 구현했습니다. 최신 벤치마크는 XGBoost가 LightGBM.xxxxiii와 경쟁적인 성능에 도달함을 보여줍니다.

매개변수

LightGBM 조정은 Random Forest와 같은 다른 알고리즘에 비해 약간 더 복잡합니다. LightGBM은 매개변수가 제대로 구성되지 않은 경우 과적합에 취약할 수 있는 리프 방식(최상 우선) 트리 성장 알고리즘을 사용합니다. 또한 LightGBM에는 100개 이상의 매개변수가 있습니다. 가장 중요한 매개변수에 초점을 맞추는 것만으로도 LightGBM을 시작하는 데 도움이 됩니다. 나머지는 알고리즘에 익숙해지면 배울 수 있습니다.

- `max_depth`: 나무가 너무 깊게 자라는 것을 방지하려면 이 매개변수를 설정합니다. 얕은 나무는 과적합될 가능성이 적습니다. 이 매개변수를 설정하는 것은 데이터세트가 작은 경우 특히 중요합니다.

- num_leaves: 트리 모델의 복잡성을 제어합니다. 과적합을 방지하려면 값이 2^{max_depth} 보다 작아야 합니다. num_leaves를 큰 값으로 설정하면 과적합 가능성이 높아지면서 정확도가 증가할 수 있습니다. num_leaves를 작은 값으로 설정하면 과적합을 방지할 수 있습니다.
 - min_data_in_leaf: 이 매개변수를 큰 값으로 설정하면 나무가 너무 깊게 자라는 것을 방지 할 수 있습니다. 이것은 과적합을 제어하는 데 도움이 되도록 설정할 수 있는 또 다른 매개 변수입니다. 값을 너무 크게 설정하면 underfitting이 발생할 수 있습니다.
 - max_bin: LightGBM은 연속 기능의 값을 다음으로 그룹화합니다 .
히스토그램을 사용하여 이산 버킷. 값이 그룹화될 빈 수를 지정하려면 max_bin을 설정하십시오. 작은 값은 과적합을 제어하고 훈련 속도를 향상시키는 데 도움이 되는 반면 큰 값은 정확도를 향상시킵니다.
 - feature_fraction: 이 매개변수는 기능 서브샘플링을 활성화합니다. 이 매개변수는 각 반복에서 무작위로 선택되는 기능의 비율을 지정합니다. 예를 들어, feature_fraction을 0.75로 설정하면 각 반복에서 75%의 기능이 무작위로 선택됩니다.
- 이 매개변수를 설정하면 훈련 속도를 높이고 과적합을 방지하는 데 도움이 됩니다.
- bagging_fraction: 각 반복에서 선택될 데이터의 비율을 지정합니다. 예를 들어 bagging_fraction을 0.75로 설정하면 각 반복에서 데이터의 75%가 무작위로 선택 됩니다. 이 매개변수를 설정하면 훈련 속도를 높이고 과적합을 방지하는 데 도움이 됩니다.
 - num_iteration: 부스팅 반복 횟수를 설정합니다. 기본값은 100. 다중 클래스 분류를 위해 LightGBM은 num_class * num_을 빌드합니다. 반복 트리. 이 매개변수를 설정하면 훈련 속도에 영향을 줍니다.
 - 목표: XGBoost와 마찬가지로 LightGBM은 여러 목표를 지원합니다. 기본 목표는 회귀로 설정됩니다. 모델이 수행하려는 작업 유형을 지정하려면 이 매개변수를 설정하십시오 . 회귀 작업의 경우 옵션은 regression_l2, regression_l1, poisson, quantile, mape, gamma, huber, fair 또는 tweedie입니다. 분류 작업의 경우 옵션은 바이너리, 멀티클래스 또는 멀티클래스소바입니다. 예측할 수 없는 결과나 낮은 정확도를 피하기 위해 목표를 올바르게 설정하는 것이 중요합니다.

3장 지도 학습

항상 그렇듯이 매개변수 그리드 검색을 수행하여 최적의 값을 결정합니다.
이러한 매개변수에 대해 매우 권장됩니다. LightGBM 매개변수의 전체 목록은 LightGBM 온라인 설명서를 참조하십시오.

참고 이 글을 쓰는 시점에서 Spark용 LightGBM은 Python용 LightGBM과 기능 패리티에 아직 도달하지 않았습니다. Spark용 LightGBM에는 가장 중요한 매개변수가 포함되어 있지만 여전히 몇 가지가 빠져 있습니다. <https://bit.ly/2OqHI2M> 을 방문하여 LightGBM for Spark에서 사용 가능한 모든 매개변수 목록을 얻을 수 있습니다. <https://bit.ly/30YGyaO>.

예시

Listing 3-7 에 나와 있는 동일한 telco churn 데이터 세트와 이전 Random Forest 및 XGBoost 예제의 대부분의 코드를 재사용합니다.

목록 3-7. LightGBM을 통한 이탈 예측

```
spark-shell --packages Azure:mmlspark:0.15
// CSV 파일을 DataFrame에 로드합니다.

val dataDF = spark.read.format("csv")
    .option("헤더", "참")
    .option("inferSchema", "true")
    .load("churn_data.txt")
```

// 스키마를 확인합니다.

```
dataDF.printSchema
```

뿌리

```
|-- 상태: 문자열(nullable = true)
|-- account_length: 이중(nullable = true)
|-- area_code: 이중(nullable = true)
|-- phone_number: 문자열(nullable = true)
|-- international_plan: 문자열(nullable = true)
```

```

|-- voice_mail_plan: 문자열(nullable = true)
|-- number_vmail_messages: 이중(nullable = true)
|-- total_day_minutes: 이중(nullable = true)
|-- total_day_calls: 이중(nullable = true)
|-- total_day_charge: 이중(nullable = true)
|-- total_eve_minutes: 이중(nullable = true)
|-- total_eve_calls: 이중(nullable = true)
|-- total_eve_charge: 이중(nullable = true)
|-- total_night_minutes: 이중(nullable = true)
|-- total_night_calls: 이중(nullable = true)
|-- total_night_charge: 이중(nullable = true)
|-- total_intl_minutes: 이중(nullable = true)
|-- total_intl_calls: 이중(nullable = true)
|-- total_intl_charge: 이중(nullable = true)
|-- number_customer_service_calls: 이중(nullable = true)
|-- churned: 문자열(nullable = true)

```

// 몇 개의 열을 선택합니다.

```
dataDF.select("state","phone_number","international_plan","churned").show
```

```
+----+-----+-----+-----+
```

|주|전화번호|국제계획|변경됨|

```
+----+-----+-----+-----+
```

KS	382-4657	아니 거짓
오	371-7191	아니 거짓
뉴저지	358-1921	아니 거짓
외	375-9999	네 거짓
확인	330-6626	네 거짓
알	391-8027	네 거짓
MA	355-9993	아니 거짓
모	329-9001	네 거짓
LA	335-4719	아니 거짓
WV	330-8173	네 거짓
인	329-6603	아니 참
리	344-9403	아니 거짓
이아	363-1107	아니 거짓

3장 지도 학습

MT	394-8006	아니 거짓
이아	366-9238	아니 거짓
뉴욕	351-7269	아니 참
아이디	350-8884	아니 거짓
VT	386-2923	아니 거짓
비지니스	356-2992	아니 거짓
텍사스	373-2782	아니 거짓

상위 20개 행만 표시

org.apache.spark.ml.feature.StringIndexer 가져오기

```
val labelIndexer = new StringIndexer().setInputCol("변경된").
setOutputCol("레이블")
```

```
val intPlanIndexer = 새로운 StringIndexer().setInputCol("국제 계획").
setOutputCol("int_plan")
```

```
val 기능 = Array("number_customer_service_calls", "total_day_분",
"total_eve_minutes", "account_length", "number_vmail_메시지",
"total_day_calls", "total_day_charge", "total_eve_calls", "total_eve_charge",
"total_night_calls", "total_intl_calls", "total_intl_청구",
"int_plan")
```

org.apache.spark.ml.feature.VectorAssembler 가져오기

```
val 어셈블러 = 새로운 VectorAssembler()
.setInputCols(기능)
.setOutputCol("기능")
```

발 시드 = 1234

val Array(trainingData, testData) = dataDF.randomSplit(Array(0.9, 0.1), 시드)

// LightGBM 분류기를 생성합니다.

com.microsoft.ml.spark.LightGBMClassifier 가져오기

```
val lightgbm = 새로운 LightGBMClassifier()
.setFeaturesCol("기능")
.setLabelCol("레이블")
```

```
.setRawPredictionCol("rawPrediction") .setObjective("바
이너리")

// setObjective 메서드를 사용하여 올바른 목표를 설정하는 것을 잊지 마십시오.
// 잘못된 목표를 지정하면 정확도에 영향을 미치거나 // 예측할 수 없는 결과가 생성될 수 있습니다.
LightGBM에서 기본 목표는 // 회귀로 설정됩니다. 이 예에서는 이진 분류를 수행하므로 // 목적을 이진으
로 설정합니다.
```

참고 Spark는 버전 2.4부터 Barrier Execution Mode를 지원합니다. LightGBM은 버전 0.18부터 setUseBarrierExecutionMode 메소드로 Barrier Execution Mode를 지원합니다.

org.apache.spark.ml.evaluation.BinaryClassificationEvaluator 가져오기

값 평가자 = 새로운

```
BinaryClassificationEvaluator() .setLabelCol("label") .setMetricName("areaUnderROC")
```

org.apache.spark.ml.tuning.ParamGridBuilder 가져오기

```
val paramGrid = new
    ParamGridBuilder() .addGrid(lightgbm.maxDepth,
        Array(2, 3, 4)) .addGrid(lightgbm.numLeaves, Array(4,
        6, 8)) .addGrid(lightgbm.numIterations, Array(600) ) .진
    다()
```

org.apache.spark.ml 가져오기.{ 파이프라인, 파이프라인스테이지 }

val 파이프라인 = new

```
Pipeline() .setStages(Array(labelIndexer, intPlanIndexer, 어셈블러,
    lightgbm))
```

org.apache.spark.ml.tuning.CrossValidator 가져오기

val cv = 새로운

```
CrossValidator() .setEstimator(파
이프라인) .setEvaluator(평가자)
```

3장 지도 학습

```
.setEstimatorParamMaps(paramGrid) .setNumFolds(3)

val 모델 = cv.fit(trainingData)
// 이제 테스트 데이터에 대해 몇 가지 예측을 할 수 있습니다.

val 예측 = model.transform(testData)
예측.printSchema

뿌리
|-- state: string(nullable = true) |-- account_length:
double(nullable = true) |-- area_code: double(nullable = true)
|-- phone_number: string(nullable = true) |-- international_plan:
string (nullable = true) |-- voice_mail_plan: string(nullable =
true) |-- number_vmail_messages: double(nullable = true) |--_
total_day_minutes: double(nullable = true) |-- total_day_calls:
double(nullable = true) | -- total_day_charge: double(nullable = true) |--_
total_eve_minutes: double(nullable = true) |-- total_eve_calls:
double(nullable = true) |-- total_eve_charge: double(nullable = true) |--_
total_night_minutes: double( nullable = true) |-- total_night_calls:
double(nullable = true) |-- total_night_charge: double(nullable = true)
|-- total_intl_minutes: double(nullable = true) |-- total_intl_calls:
double(nullable = true) | - total_intl_charge: 이중(nullable = true) |--_
number_customer_service_calls: 이중(nullable = true) |-- churned: 문자열(
nullable = true) |-- 레이블: 이중 (nullable = false) |-- int_plan:
double(nullable = false) |-- 기능: 벡터(nullable = true) |-- rawPrediction: 벡
터(nullable = true) |-- 확률: 벡터(nullable = true) | -- 예측: 이중(nullable =
false)
```

// 모델을 평가합니다. AUC 점수가 Random Forest보다 높습니다.

// 및 이전 예제의 XGBoost.

```
val auc = evaluator.evaluate(예측)
```

auc: 더블 = 0.940366124260358

//LightGBM을 사용하면 기능 중요도를 추출할 수도 있습니다.

```
com.microsoft.ml.spark.LightGBMClassificationModel 가져오기
```

```
org.apache.spark.ml.PipelineModel 가져오기
```

```
val bestModel = model.bestModel
```

```
val 모델 = bestModel.asInstanceOf[파이프라인 모델]
```

.단계

.마지막

```
.asInstanceOf[LightGBM 분류 모델]
```

LightGBM에는 "split"(총 분할 수) 및 "gain"(총 정보 이득)의 두 가지 유형의 기능 중요도가 있습니다. "이득"을 사용하는 것이 일반적으로 권장되며 기능 중요도를 계산하는 Random Forest의 방법과 대략 유사하지만, LightGBM은 이진 분류 예제에서 지니 불순물을 사용하는 대신 교차 엔트로피(로그 손실)를 사용합니다([표 3-3](#) 및 [3-4](#) 참조).). 최소화할 손실은 지정된 목표에 따라 다릅니다.^{xxxiv}

```
val gainFeatureImportances = model.getFeatureImportances("이득")
```

```
gainFeatureImportances: Array[Double] =
```

```
어레이(2648.0893859118223, 5339.0795262902975, 2191.832309693098
```

```
, 564.6461282968521, 1180.4672759771347, 656.8244850635529 0.0,
```

```
533.6638155579567, 579.7435692846775, 651.5408382415771, 1179.492751300335,
```

```
2186.5995585918427, 1773.7864662855864)
```

3장 지도 학습

표 3-3. Information Gain을 사용하는 LightGBM의 기능 중요성

색인	특징	기능 중요도
0	number_customer_service_calls	2648.0893859118223
1	total_day_minutes	5339.0795262902975
2	total_eve_minutes	2191.832309693098
3	계정 길이	564.6461282968521
4	number_vmail_messages	1180.4672759771347
5	total_day_calls	656.8244850635529
6	total_day_charge	0.0
7	total_eve_calls	533.6638155579567
8	total_eve_charge	579.7435692846775
9	total_night_calls	651.5408382415771
10	total_intl_calls	1179.492751300335
11	total_intl_charge	2186.5995585918427
12	int_plan	1773.7864662855864

"split"을 사용할 때 출력을 비교하십시오.

```
val gainFeatureImportances = model.getFeatureImportances("분할")
개인 기능 중요: Array[Double] = Array(159.0, 583.0, 421.0, 259.0, 133.0, 264.0, 0.0, 214.0, 92.0, 279.0, 279.0),
5686
```

표 3-4. 분할 수를 사용한 LightGBM의 기능 중요성

인덱스	기능	기능 중요도
0	number_customer_service_calls	159.0
1	total_day_minutes	583.0
2	total_eve_minutes	421.0
삼	계정 길이	259.0
4	number_vmail_messages	133.0
5	total_day_calls	264.0
6	total_day_charge	0.0
7	total_eve_calls	214.0
8	total_eve_charge	92.0
9	total_night_calls	279.0
10	total_intl_calls	279.0
11	total_intl_charge	366.0
12	int_plan	58.0

```
println(s"참 음수: ${predictions.select("*").where("예측 = 0 AND 레이블 = 0").count()} 참 양수: \$ {predictions.select("*").where("예측 = 1 AND 레이블 = 1").count()}"")
```

참음성: 407 참양성: 58

```
println(s"거짓음수: ${predictions.select("*").where("예측 = 0 AND 레이블 = 1").count()} 거짓양성: \$ {predictions.select("*").where("예측 = 1 AND 레이블 = 0").count()}"")
```

거짓 부정: 20 거짓 긍정: 9

Naive Bayes를 사용한 감정 분석

Naive Bayes는 Bayes의 정리를 기반으로 하는 간단한 다중 클래스 선형 분류 알고리즘입니다. Naive Bayes는 데이터 세트의 기능이 독립적이라고 순진하게 가정하고 기능 간의 가능한 상관 관계를 무시하기 때문에 그 이름을 얻었습니다. 이것은 실제 시나리오의 경우가 아니지만 여전히 Naive Bayes는 특히 작은 데이터 세트 또는 높은 차원의 데이터 세트에서 잘 수행되는 경향이 있습니다. 선형 분류기와 마찬가지로 비선형 분류 문제에서는 성능이 좋지 않습니다. Naive Bayes는 데이터 세트에 대한 단일 패스만 필요로 하는 계산 효율적이고 확장성이 뛰어난 알고리즘입니다. 대규모 데이터 세트를 사용하는 분류 작업에 대한 좋은 기준 모델입니다. 특정 기능 집합이 주어진 클래스에 속할 확률을 찾는 방식으로 작동합니다. Bayes의 정리 방정식은 다음과 같이 나타낼 수 있습니다.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$P(A|B)$ 는 다음과 같이 해석될 수 있는 사후 확률입니다. "사건 B가 주어졌을 때 사건 A가 일어날 확률은 얼마입니까?" B는 특징 벡터를 나타냅니다. 분자는 조건부 확률에 사전 확률을 곱한 값을 나타냅니다. 분모는 증거를 나타냅니다. 방정식은 다음과 같이 더 정확하게 작성할 수 있습니다.

$$\text{피 } y|x_{1:n}) \frac{1}{4} = \frac{\text{피}(y_1:n | x_1:n)}{\text{피}(x_1:n) \frac{1}{4}}$$

Naive Bayes는 텍스트 분류에 자주 사용됩니다. 텍스트 분류를 위한 인기 있는 응용 프로그램에는 스팸 탐지 및 문서 분류가 있습니다. 또 다른 텍스트 분류 사용 사례는 감정 분석입니다. 회사는 제품이나 서비스에 대한 여론이 긍정적인지 부정적인지 판단하기 위해 정기적으로 소셜 미디어의 댓글을 확인합니다. 헤지 펀드는 감정 분석을 사용하여 주식 시장의 움직임을 예측합니다.

Spark MLlib는 Bernoulli naïve Bayes 및 다항 naïve Bayes를 지원합니다.

Bernoulli naïve Bayes는 부울 또는 이진 기능(예: 문서에 단어의 존재 여부)에서만 작동하는 반면, 다항 naïve Bayes는 이산 기능(예: 단어 수)을 위해 설계되었습니다. MLlib의 순진한 Bayes 구현을 위한 기본 모델 유형은 다항식으로 설정됩니다. 평활화를 위해 다른 매개변수인 람다를 설정할 수 있습니다(기본값은 1.0).

예시

감정 분석을 위해 순진한 Bayes를 사용하는 방법을 보여주는 예제를 살펴보겠습니다. 캘리포니아 대학교 어바인 머신 러닝 리포지토리의 인기 있는 데이터 세트를 사용합니다. 데이터 세트는 "깊은 기능을 사용하여 그룹에서 개별 레이블로"라는 논문을 위해 만들어졌습니다. Kotzias et. al., KDD 2015. 데이터 세트는 IMDB, Amazon 및 Yelp의 세 회사에서 제공합니다. 각 회사에 대해 500개의 긍정적인 리뷰와 500개의 부정적인 리뷰가 있습니다. Amazon의 데이터 세트를 사용하여 Amazon 제품 리뷰를 기반으로 특정 제품에 대한 감정이 긍정적(1) 또는 부정적(0)일 확률을 결정합니다.

데이터 세트의 각 문장을 특징 벡터로 변환해야 합니다. 스파크 MLlib 정확히 이 목적을 위한 변압기를 제공합니다. TF-IDF(Term Frequency-Inverse Document Frequency)는 일반적으로 텍스트에서 특징 벡터를 생성하는 데 사용됩니다. TF-IDF는 해당 단어가 문서에서 발생하는 횟수(TF)와 단어가 전체 말뭉치(IDF)에서 발생하는 빈도를 계산하여 말뭉치의 문서에 대한 단어의 관련성을 결정하는 데 사용됩니다. Spark MLlib에서는 TF와 IDF가 별도로 구현됩니다(HashingTF 및 IDF).

TF-IDF를 사용하여 단어를 특징 벡터로 변환하기 전에 다른 변환기인 토크나이저를 사용하여 문장을 개별 단어로 분할해야 합니다. 단계는 Listing 3-8에 표시된 코드와 함께 그림 3-8과 같아야 합니다 .

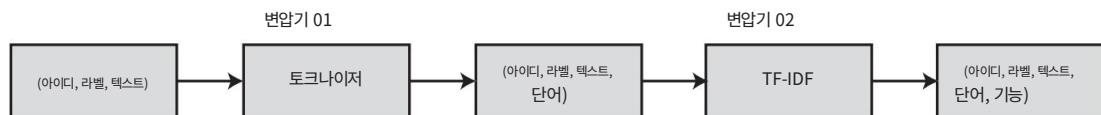


그림 3-8. 감정 분석 예제의 기능 변환

목록 3-8. Naive Bayes를 사용한 감정 분석

```
// 데이터세트에 대한 스키마를 생성하여 시작합니다.
org.apache.spark.sql.types._ 가져오기
```

```
var reviewSchema = StructType(배열(
    StructField("텍스트", StringType, true),
    StructField("레이블", IntegerType, true)
))
```

3장 지도 학습

```
// 탭으로 구분된 텍스트 파일에서 DataFrame을 만듭니다.  
// 탭이나 쉼표로 구분되는지 여부에 관계없이 "csv" 형식을 사용합니다.  
// 파일에 헤더가 없으므로 헤더를 설정합니다.  
// 옵션을 false로 설정합니다. 구분 기호를 탭으로 설정하고 스키마를 사용합니다.  
// 방금 만든 것입니다.
```

```
val reviewDF = spark.read.format("csv")  
    .option("헤더", "거짓")  
    .option("구분자", "\t")  
    .schema(reviewsSchema)  
    .load("/files/amazon_cells_labelled.txt")
```

// 스키마를 검토합니다.

reviewDF.printSchema

뿌리

```
|-- 텍스트: 문자열(nullable = true)  
|-- 레이블: 정수(nullable = true)
```

// 데이터를 확인합니다.

리뷰DF.show

	텍스트 라벨
그래서 와가 없어... 잘했어, 엑셀...	0
텍에 좋은... 충전기에 연결... 마이	1
크가 좋아요. 흔들어야 하는데 ...	1
단절이 있다면... 당신이 Razr라	0
면... 말할 필요도 없이 ... 정말 낭비	1
야... 그리고 사운드는... 그는 매우	0
인상 깊었습니다...	0
	1
	0
	0
	1
	1

들이라면... 아주 좋은 품질... 다자	0
인은 ver... 강력 추천합니다... 나	1
는 모두에게 조언한다... 지금까지	0
너무 좋아!. 잘 작동합니다!. 그것	1
은 클릭으로 pl...	0
	1
	1
	0

+-----+-----+

상위 20개 행만 표시

// 행 수를 계산해 보겠습니다.

```
reviewDF.createOrReplaceTempView("리뷰")
spark.sql("레이블별로 리뷰 그룹에서 레이블, 개수(*) 선택").show
```

+-----+-----+

라벨	수(1)
1	500
0	500

+-----+-----+

// 데이터 세트를 학습 데이터 세트와 테스트 데이터 세트로 무작위로 나눕니다.

발 시드 = 1234

```
val Array(trainingData, testData) = reviewDF.randomSplit(Array(0.8, 0.2), 시드)
```

트레이닝 데이터.카운트

res5: 긴 = 827

testData.count

res6: 긴 = 173

// 문장을 단어로 나눕니다.

org.apache.spark.ml.feature.TOKENIZER 가져오기

```
val 토크나이저 = new Tokenizer().setInputCol("텍스트")
.setOutputCol("단어")
```

3장 지도 학습

// 토큰화된 데이터를 확인합니다.

```
val tokenizedDF = tokenizer.transform(trainingData)
```

토큰화된DF.show

텍스트 라벨	단어
(작동합니다!))설정할 수 없	1 [(그것, 작동합니다!)]
습니다... /* 성대가 함께 제공됩	1 [)설정할 수 없습니다...
니다... 물품이 도착했습니	1 [* , 와, ...
다... 1. 오래 지속되는 b... 좋	1 [...., 아이템, 도착...
아요 2개... :-)오, 요금이...	0 [1., 오래 지속되는...
실망입니다. 정크 티 조각...	1 [2, 엄지손가락, 위로, t...
좋은 품질의 바... 반드시 공부	1 [:-)오, 그, 차...
해야 하는 ... 꽤 좋은 프로...	0 [아, 실망.]
사용 가능한 키보드... 일주일	0 [a, piece, of, ju...
후... 재치있게 논쟁한 후...	1 [좋은, 좋은 품질...
첫 번째 c 이후... 아마존 짜증	0 [아, 꼭, 공부, ...
나. 절대 쓰레기. 정말 좋아	1 [아, 예쁘다, 좋다,...
요. 어댑터는 ...	1 [a, 사용 가능, 키보...
	0 [a, 주, 나중에, ...
	0 [말다툼 끝에 ...
	0 [이후, 전나무...
	0 [아마존, 짜증나.]
	0 [절대, 정크.]
	1 [물론입니다. 훌륭합니다.]
	0 [어댑터, n...

상위 20개 행만 표시

// 다음으로 HashingTF를 사용하여 토큰화된 단어를 변환합니다.

// 고정 길이 특징 벡터로.

org.apache.spark.ml.feature.HashingTF 가져오기

```
val htf = 새로운 HashingTF().setNumFeatures(1000)
    .setInputCol("단어")
```

```
.setOutputCol("기능")

// 벡터화된 기능을 확인합니다.

val hashedDF = htf.transform(tokenizedDF)

hashedDF.show

+-----+-----+-----+
| 텍스트|라벨| 단어| 기능|
+-----+-----+-----+
| (작동합니다!)| 설정할 수 없습니다| 1|[그것, 작동합니다!])(1000,[369,504],...|
다...|* 성...|.... 물품 도착...| 1|. 오| 1|[]설정, 할 수 없습니다...(1000,[299,520,53...|
래 지속되는 b...| 좋아요 2개...|:-)| 1|*, 와, ...|(1000,[34,51,67,1...|
오, 요금이...| 실망입니다.| 정크 티| 1|[...., 항목, 도착...|(1000,[98,133,245...|
조각...| 좋은 품질의 바...| 반드시 공| 0|[1., 긴, 지속...|(1000,[138,258,29...|
부해야 하는 ...| 꽤 좋은 프로...| 사| 1|[2, 엄지손가락, 위로, t...|(1000,[92,128,373...|
용 가능한 키보드...| 일주일 후...| 재| 1|[:-)오, 그, 차...|(1000,[388,497,52...|
치있게 논쟁한 후...| 첫 번째 c 이| 0|[a, 실망.](1000,[170,386],...|
후...| 아마존 짜증나.| 절대 쓰레| 0|[a, piece, of, ju...|(1000,[34,36,47,7...|
기.| 정말 좋아요.| 어댑터는 ...| 1|[a, 좋은, 품질...|(1000,[77,82,168,...|
| 0|[a, 필수, 공부, ...|(1000,[23,36,104,...|
| 1|[아, 예쁘다, 좋다,...|(1000,[168,170,27...|
| 1|[a, 사용 가능, keybo...|(1000,[2,116,170,...|
| 0|[a, 주, 나중, ...|(1000,[77,122,156...|
| 0|[후, 논쟁, ...|(1000,[77,166,202...|
| 0|[뒤에, the, 전나무...|(1000,[63,77,183,...|
| 0|[아마존, 짜증나.](1000,[828,966],...|
| 0|[절대, 정크.](1000,[607,888],...|
| 1|[물론입니다.](1000,[589,903],...|
| 0|[어댑터, n...|(1000,[0,18,51,28...|
+-----+-----+-----+
```

상위 20개 행만 표시

// 우리는 MLlib에서 제공하는 순진한 Bayes 분류기를 사용할 것입니다.

org.apache.spark.ml.classification.NaiveBayes 가져오기

val nb = 새로운 NaiveBayes()

3장 지도 학습

```
// 이제 조립에 필요한 모든 부품이 있습니다.  
// 머신 러닝 파이프라인.  
  
org.apache.spark.ml.Pipeline 가져오기  
  
val 파이프라인 = new Pipeline().setStages(Array(토큰나이저, htf, nb))  
  
// 훈련 데이터 세트를 사용하여 모델을 훈련합니다.  
  
val 모델 = pipeline.fit(trainingData)  
  
// 테스트 데이터 세트를 사용하여 예측합니다.  
  
val 예측 = model.transform(testData)  
  
// 각 리뷰에 대한 예측을 표시합니다.  
  
predicts.select("텍스트", "예측").show  
+-----+-----+  
|      텍스트|예측|  
+-----+-----+  
||확실히 기억하고 있어...| #1 작      1.0| | |
|동합니다 - #2 ...| | 배수로 $50.|    1.0|  
|많은 웹사이트...| |충전 후...| |전      0.0|  
|화가 끝나면...| |전반적으로 나는    1.0|  
|날씬하다...| |그것은 그저...| |또      0.0|  
|한, 당신의 사진이...| |그리고 난      0.0|  
|그냥 사랑해...| |그리고 그 어떤    1.0|  
|것도...| |잘못된 선택입니다.| |최      0.0|  
|고의 헤드셋...| |큰 실망...| |블루      0.0|  
|투스 범위 나는...| |하지만 이것에    1.0|  
|도 불구하고...| |구매자-매우      1.0|  
|Ca...| |아무것도|매장할 수 없습      0.0|  
                                         1.0|  
                                         0.0|  
                                         0.0|  
                                         0.0|  
                                         1.0|  
                                         0.0|
```

중국 위조... 다음 경우에는 구	0.0
매하지 마십시오.	0.0

상위 20개 행만 표시

// 이진 분류기 평가자를 사용하여 모델을 평가합니다.

```
org.apache.spark.ml.evaluation.BinaryClassificationEvaluator 가져오기
```

```
val 평가자 = 새로운 BinaryClassificationEvaluator()
```

```
org.apache.spark.ml.param.ParamMap 가져오기
```

```
val paramMap = ParamMap(evaluator.metricName -> "areaUnderROC")
```

```
val auc = evaluator.evaluate(예측, paramMap)
```

```
auc: 더블 = 0.5407085561497325
```

// 긍정적인 예를 테스트합니다.

```
val 예측 = 모
```

```
델 .transform(sc.parallelize(Seq("이 제품은 좋습니다")).toDF("텍스트"))
```

```
predicts.select("텍스트", "예측").show
```

텍스트	예측
-----	----

이 제품은 좋다	1.0
----------	-----

// 부정적인 예를 테스트합니다.

```
val 예측 = 모
```

```
델 .transform(sc.parallelize(Seq("이 제품은 불량입니다")).toDF("텍스트"))
```

```
predicts.select("텍스트", "예측").show
```

텍스트	예측
-----	----

이 제품은 불량입니다	0.0
-------------	-----

3장 지도 학습

모델을 개선하기 위해 수행할 수 있는 몇 가지 작업이 있습니다. 대부분의 NLP(자연어 처리) 작업에서는 n-gram, 표제어 표기법 및 불용어 제거와 같은 추가 텍스트 전처리를 수행하는 것이 일반적입니다. 4 장에서 Stanford CoreNLP와 Spark NLP를 다룹니다.

회귀

회귀는 연속 숫자 값을 예측하기 위한 지도 머신 러닝 작업입니다. 인기 있는 사용 사례에는 판매 및 수요 예측, 주식, 주택 또는 상품 가격 예측, 일기 예보 등이 있습니다. 회귀에 대해서는 1장에서 더 자세히 설명 합니다.

단순 선형 회귀

선형 회귀는 하나 이상의 독립 변수와 종속 변수 간의 선형 관계를 조사하는 데 사용됩니다. 단일 독립 변수와 단일 연속 종속 변수 간의 관계 분석을 단순 선형 회귀라고 합니다.

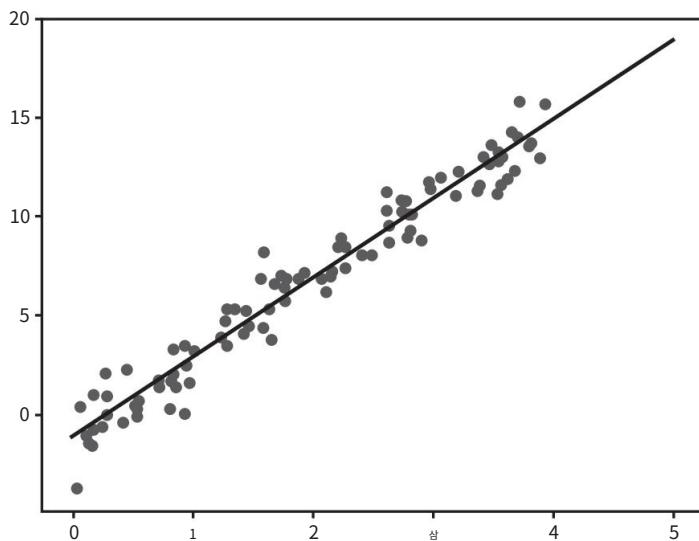


그림 3-9. 단순 선형 회귀 플롯

그림 3-9에서 볼 수 있듯이 플롯은 직선을 그리려는 선형 공격을 보여줍니다.
관찰된 반응과 예측된 값 사이의 잔차 제곱합을 가장 잘 줄이는 선.^{xxxv}

예시

이 예에서는 단순 선형 회귀를 사용하여 해당 지역의 평균 가구 소득(독립 변수)에 따라 주택 가격(종속 변수)이 어떻게 변하는지 보여줍니다. 목록 3-9는 코드를 자세히 설명합니다.

목록 3-9. 선형 회귀 예제

가져오기 org.apache.spark.ml.regression.LinearRegression 가져오기 spark.implicits._

```
val dataDF =  
  Seq(  
    (50000,  
     302200), (75200,  
     550000), (90000,  
     680000), (32800,  
     225000), (32800,  
     225000), (0,50,0),  
    (41000, 275000),  
    (540000, 0) 700000),  
    (88500, 673100),  
    (92000, 695000),  
    (53000, 320900),  
    (85200, 652800),  
    (85200, 652800),
```

```
(0,1,0,3,0,0,3,0,3,0,0,3,0,1,57000,890000), (128000), (178000), (33400, 265900), (143000, 846000), (  
dataDF.show
```

3장 지도 학습

```
+-----+-----+
|avg_area_income| 가격|
+-----+-----+
| 50000|302200|
| 75200|550000|
| 90000|680000|
| 32800|225000|
| 41000|275000|
| 54000|300500|
| 72000|525000|
| 105000|700000|
| 88500|673100|
| 92000|695000|
| 53000|320900|
| 85200|652800|
| 157000|890000|
| 128000|735000|
| 71500|523000|
| 114000|720300|
| 33400|265900|
| 143000|846000|
| 68700|492000|
| 46100|285000|
+-----+-----+
```

org.apache.spark.ml.feature.VectorAssembler 가져오기

```
val 어셈블러 = 새로운 VectorAssembler()
    .setInputCols(Array("avg_area_income"))
    .setOutputCol("기능")

val dataDF2 = assembler.transform(dataDF)

dataDF2.show
```

```
+-----+-----+
|avg_area_income| 가격| 기능|
+-----+-----+
      50000|302200| [50000.0]|
      75200|550000| [75200.0]|
      90000|680000| [90000.0]|
      32800|225000| [32800.0]|
      41000|275000| [41000.0]|
      54000|300500| [54000.0]|
      72000|525000| [72000.0]|
      105000|700000|[105000.0]|
      88500|673100| [88500.0]|
      92000|695000| [92000.0]|
      53000|320900| [53000.0]|
      85200|652800| [85200.0]|
      157000|890000|[157000.0]|
      128000|735000|[128000.0]|
      71500|523000| [71500.0]|
      114000|720300|[114000.0]|
      33400|265900| [33400.0]|
      143000|846000|[143000.0]|
      68700|492000| [68700.0]|
      46100|285000| [46100.0]|
+-----+-----+
```

```
val lr = 새로운 LinearRegression()
```

```
  .setMaxIter(10)
  .setFeaturesCol("기능")
  .setLabelCol("가격")
```

```
발 모델 = lr.fit(dataDF2)
```

```
org.apache.spark.ml.linalg.Vectors 가져오기
```

```
val testData = 스파크
```

```
  .createDataFrame(Seq(Vectors.dense(75000)))
  .map(Tuple1.apply))
  .toDF("기능")
```

3장 지도 학습

```
val 예측 = model.transform(testData)
```

```
예측.쇼
+-----+
| 기능| 예측|
+-----+
|[75000.0]|504090.35842779215|
+-----+
```

XGBoost4J-Spark를 사용한 다중 회귀

다중 회귀는 두 개 이상의 독립 변수와 단일 연속 종속 변수가 있는 보다 현실적인 시나리오에서 사용됩니다. 실제 사용 사례에서 선형 및 비선형 기능을 모두 갖는 것이 일반적입니다. 트리 기반 양상을

XGBoost와 같은 알고리즘은 선형 및 비선형 기능을 모두 처리할 수 있어 대부분의 프로덕션 환경에 이상적입니다. 다중 회귀에 XGBoost와 같은 트리 기반 양상을 사용하는 대부분의 상황에서 예측 정확도가 훨씬 더 높아야 합니다. xxxvi

이 장의 앞부분에서 분류 문제를 해결하기 위해 XGBoost를 사용했습니다. XGBoost는 분류와 회귀를 모두 지원하므로 회귀에 XGBoost를 사용하는 것은 분류와 매우 유사합니다.

예시

Listing 3-10 과 같이 다중 회귀 예제에 대해 약간 더 복잡한 데이터 세트를 사용할 것 입니다. 데이터세트는 Kaggle.xxxxvii에서 다운로드할 수 있습니다. 우리의 목표는 데이터세트에 제공된 속성을 기반으로 주택 가격을 예측하는 것입니다. 데이터세트에는 7개의 열이 있습니다. Avg. 지역 소득, 평균 면적 주택 연령, 평균 방의 면적, 평균 면적 침실 수, 면적 인구, 가격 및 주소. 단순함을 위해 주소 필드를 사용하지 않습니다(유용한 정보는 집 주소에서 가까운 학교의 위치를 찾을 수 있음). 가격은 우리의 종속 변수입니다.

목록 3-10. XGBoost4J-Spark를 사용한 다중 회귀

```
스파크 웹 --패키지 ml.dmlc:xgboost4j-spark:0.81
```

```
org.apache.spark.sql.types._ 가져오기
```

```
// 데이터세트에 대한 스키마를 정의합니다.
```

```

var priceSchema = StructType(배열(
    StructField("avg_area_income", DoubleType, true),
    StructField("avg_area_house_age", DoubleType, true),
    StructField("avg_area_num_rooms", DoubleType, true),
    StructField("avg_area_num_bedrooms", DoubleType, true),
    StructField("area_population", DoubleType, true),
    StructField("가격", DoubleType, true) ))

val dataDF = spark.read.format("csv") .option("해
더","true") .schema(pricesSchema) .load("USA_Housing.csv").na.drop()

// 데이터세트를 검사합니다.

dataDF.printSchema 루트

 |-- avg_area_income: double(nullable = true)
 |-- avg_area_house_age: double(nullable = true)
 |-- avg_area_num_rooms: double(nullable = true)
 |-- avg_area_num_bedrooms: double(nullable = true) |-- area_population:
더블 (nullable = true) |-- price: double(nullable = true)

dataDF.select("avg_area_income","avg_area_house_age","avg_area_num_rooms").보여 주다

+-----+-----+-----+
| avg_area_income|avg_area_house_age|avg_area_num_rooms|
+-----+-----+-----+
| 79545.45857431678| 5.682861321615587| 7.009188142792237| |
79248.64245482568| 6.0028998082752425| 6.730821019094919| |
61287.067178656784| 5.865889840310001| 8.512727430375099| |
63345.24004622798| 7.1882360945186425| 5.586728664827653| |
59982.197225708034| 5.040554523106283| 7.8393877785120487| |
80175.7541594853| 4.9884077575337145| 6.104512439428879| 64698.46342788773|
6.025335906887153| 8.147759585023431|

```

3장 지도 학습

```
| 78394.33927753085|6.9897797477182815| 6.620477995185026|
| 59927.66081334963| 5.36212556960358|6.3931209805509015|
| 81885.92718409566| 4.423671789897876| 8.167688003472351|
| 80527.47208292288| 8.09351268063935| 5.042746799645982|
| 50593.69549704281| 4.496512793097035| 7.467627404008019|
| 39033.809236982364| 7.671755372854428| 7.250029317273495|
| 73163.6634410467| 6.919534825456555|5.9931879009455695|
| 69391.3801843616| 5.344776176735725| 8.406417714534253|
| 73091.86674582321| 5.443156466535474| 8.517512711137975|
| 79706.96305765743| 5.067889591058972| 8.219771123286257|
| 61929.07701808926| 4.788550241805888|5.0970095543775615|
| 63508.19429942997| 5.947165139552473| 7.187773835329727|
| 62085.27640340488| 5.739410843630574| 7.09180810424997|
+-----+-----+-----+

```

상위 20개 행만 표시

```
dataDF.select("avg_area_num_bedrooms","area_population","price").show
```

```
+-----+-----+-----+
|avg_area_num_bedrooms| area_population| 가격|
+-----+-----+-----+
| 4.09|23086.800502686456|1059033.5578701235|
| 3.09| 40173.07217364482| 1505890.91484695|
| 5.13| 36882.15939970458|1058987.9878760849|
| 3.26| 34310.24283090706|1260616.8066294468|
| 4.23|26354.109472103148| 630943.4893385402|
| 4.04|26748.428424689715|1068138.0743935304|
| 3.41| 60828.24908540716|1502055.8173744078|
| 2.42|36516.358972493836|1573936.5644777215|
| 2.3| 29387.39600281585| 798869.5328331633|
| 6.1| 40149.96574921337|1545154.8126419624|
| 4.1| 47224.35984022191| 1707045.722158058|
| 4.49|34343.991885578806| 663732.3968963273|
| 3.1| 39220.36146737246|1042814.0978200927|
| 2.27|32326.123139488096|1291331.5184858206|
| 4.37|35521.294033173246|1402818.2101658515|
| 4.01|23929.524053267953|1306674.6599511993|
+-----+
```

```

3.12|39717.81357630952|1556786.6001947748|4.3|
24595.90149782299|528485.2467305964|5.12|
35719.653052030866|1019425.9367578316|5.49|
44922.106702293066|1030591.4292116085|
+-----+-----+-----+

```

상위 20개 행만 표시

```
값 기능 = Array("avg_area_income", "avg_area_house_age",
"avg_area_num_rooms", "avg_area_num_bedrooms", "area_population")
```

// 특징을 단일 특징 벡터로 결합합니다.

```
org.apache.spark.ml.feature.VectorAssembler 가져오기
```

```
발 어셈블러 = 새로운 VectorAssembler() .setInputCols(기
능) .setOutputCol("기능")
```

```
val dataDF2 = assembler.transform(dataDF)
```

```
dataDF2.select("가격", "기능").show(20,50)
```

```

+-----+-----+-----+
|      가격|          기능|
+-----+-----+-----+
|1059033.5578701235|[79545.45857431678,5.682861321615587,7.00918814...| | 1505890.91484695|
[79248.64245482568,6.0028998082752425,6.7308210...| |1058987.9878760849|
[61287.067178656784,5.865889840310001,8.5127274...| |1260616.8066294468|
[63345.24004622798,7.1882360945186425,5.5867286...| | 630943.4893385402|
[59982.197225708034,5.040554523106283,7.8393877...| |1068138.0743935304|
[80175.7541594853,4.9884077575337145,6.10451243...| |1502055.8173744078|
[64698.46342788773,6.025335906887153,8.14775958...| |1573936.5644777215|
[78394.33927753085,6.9897797477182815,6.6204779...| | 798869.5328331633|
[59927.66081334963,5.36212556960358,6.393120980...| |1545154.8126419624|
[81885.92718409566,4.423671789897876,8.16768800...| | 1707045.722158058|
[80527.47208292288,8.09351268063935,5.042746799...| | 663732.3968963273|
[50593.69549704281,4.496512793097035,7.46762740...| |1042814.0978200927|
[39033.809236982364,7.671755372854428,7.2500293...| |1291331.5184858206|
[73163.6634410467,6.919534825456555,5.993187900...|

```

3장 지도 학습

```
|1402818.2101658515|[69391.3801843616,5.344776176735725,8.406417714...| |1306674.6599511993|
[73091.86674582321,5.443156466535474,8.51751271...| |1556786.6001947748|
[79706.96305765743,5.067889591058972,8.21977112...| | 528485.2467305964|
[61929.07701808926,4.788550241805888,5.09700955...| |1019425.9367578316|
[63508.19429942997,5.947165139552473,7.18777383...| |1030591.4292116085|
[62085.27640340488,5.739410843630574,7.09180810...|
+-----+-----+
```

상위 20개 행만 표시

// 데이터 세트를 훈련 데이터와 테스트 데이터로 나눕니다.

발 시드 = 1234

```
val Array(trainingData, testData) = dataDF2.randomSplit(Array(0.8, 0.2), 시드)
```

// 화구에 XGBoost를 사용합니다.

```
가져오기 ml.dmlc.xgboost4j.scala.spark.{XGBoostRegressionModel,XGBoostRegressor}
```

val xgb = 새로운

```
XGBoostRegressor() .setFeaturesCol("기
능") .setLabelCol("가격")
```

// 매개변수 그리드를 생성합니다.

```
org.apache.spark.ml.tuning.ParamGridBuilder 가져오기
```

val paramGrid = 새로운 ParamGridBuilder()

```
.addGrid(xgb.maxDepth, Array(6,
9)) .addGrid(xgb.eta, Array(0.3, 0.7)).build()
```

```
paramGrid: 배열[org.apache.spark.ml.param.ParamMap] = 배열
```

```
({ xgbr_bacf108db722-eta: 0.3, xgbr_bacf108db722-maxDepth: 6
```

```
}, {
```

```
  xgbr_bacf108db722-eta: 0.3,
  xgbr_bacf108db722-maxDepth: 9
```

```
}, {
```

```
  xgbr_bacf108db722-eta: 0.7,
  xgbr_bacf108db722-maxDepth: 6
```

```

}, {
    xgbr_bacf108db722-eta: 0.7,
    xgbr_bacf108db722-maxDepth: 9
})
// 평가자를 생성합니다.

import org.apache.spark.ml.evaluation.RegressionEvaluator

값 평가자 = 새로운 RegressionEvaluator() .setLabelCol("가
격") .setPredictionCol("예
측") .setMetricName("rmse")

// 교차 검증기를 생성합니다.

import org.apache.spark.ml.tuning.CrossValidator
val cv = 새로운

CrossValidator() .setEstimator(xgb) .setEvaluator(평가자) .setEstimatorParamMaps

val 모델 = cv.fit(trainingData)

val 예측 = model.transform(testData)

predicts.select("기능", "가격", "예측").show
+-----+-----+-----+-----+
|      기능|          가격| 예측|
+-----+-----+-----+-----+
|[17796.6311895433...|302355.83597895555| 591896.9375| | |
|[35454.7146594754...| 1077805.577726322| 440094.75|| |
|[35608.9862370775...| 449331.5835333807| 672114.75|| |
|[38868.2503114142...| 759044.6879907805| 672114.75|| |
|[40752.7142433209...| 560598.5384309639| 591896.9375| |
|[41007.4586732745...| 494742.5435776913| 421605.28125| |
|[41533.0129597444...| 682200.3005599922| 505685.96875| |
|[42258.7745410484...| 852703.2636757497| 591896.9375| |

```

3장 지도 학습

```
[42940.1389392421...| 680418.7240122693| 591896.9375| [43192.1144092488...|
1054606.9845532854| 505685.96875| [43241.9824225005...|
629657.6132544072| 505685.96875| [44328.2562966742...| 601007.3511604669|
141361.53125| [45347.1506816944...| 541953.9056802422| 441908.40625| |
[45546.6434075757...| 923830.33486809| 591896.9375| [45610.9384142094...|
961354.287727855| 849175.75| [45685.2499205068...| 867714.3838490517|
441908.40625| [45990.1237417814...| 1043968.3994445396| 849175.75| |
[46062.7542664558...| 675919.6815570832| 505685.96875| [46367.2058588838...|
268050.81474351394| 379889.625| [47467.4239151893...| 762144.9261238109|
591896.9375|
```

```
+-----+-----+-----+-----+
```

상위 20개 행만 표시

RMSE(평균 제곱근 오차)를 사용하여 모델을 평가해 보겠습니다. 잔차는 회귀선에서 데이터 점 까지의 거리를 측정한 것입니다. RMSE는 잔차의 표준 편차이며 예측 오차를 측정하는 데 사용됩니다 .xxxviii

```
val rmse = evaluator.evaluate(예측)
```

```
rmse: 더블 = 438499.82356536255
```

// 매개변수를 추출합니다.

```
model.bestModel.extractParamMap
```

```
res11: org.apache.spark.ml.param.ParamMap = {
```

```
xgbr_8da6032c61a9 알파 : 0.0
xgbr_8da6032c61a9-baseScore : 0.5
xgbr_8da6032c61a9-checkpointInterval : -1
xgbr_8da6032c61a9-checkpointPath : xgbr_8da6032c61a9-
colsampleBylevel : 1.0 xgbr_8da6032c61a9-
colsampleBytree : 1.0 xgbr_8da6032c61a9-customEval :
NULL, xgbr_8da6032c61a9-customObj : NULL,
xgbr_8da6032c61a9- eta: 0.7, xgbr_8da6032c61a9-
evalMetric: rmse,
```

```
xgbr_8da6032c61a9-featuresCol : 가능
xgbr_8da6032c61a9 감마 0.0, xgbr_8da6032c61a9-
growPolicy : 깊이 방향, xgbr_8da6032c61a9-
labelCol : 가격 xgbr_8da6032c61a9 람다 : 1.0
xgbr_8da6032c61a9-lambdaBias : 0.0
xgbr_8da6032c61a9-maxBin : 16 xgbr_8da6032c61a9-
maxDeltaStep : 0.0, xgbr_8da6032c61a9-
MAXDEPTH : 9 xgbr_8da6032c61a9-minChildWeight :
1.0 xgbr_8da6032c61a9는 소실 : NaN의
xgbr_8da6032c61a9-normalizeType : 트리
xgbr_8da6032c61a9-nthread : 1 xgbr_8da6032c61a9-
numEarlyStoppingRounds : 0 xgbr_8da6032c61a9-
numRound : 1 xgbr_8da6032c61a9-numWorkers :
1 xgbr_8da6032c61a9 - 목적 : 등록 : 선형
xgbr_8da6032c61a9-predictionCol : 예측
xgbr_8da6032c61a9-rateDrop : 0.0 xgbr_8da6032c61a9-
sampleType : 균일 xgbr_8da6032c61a9-scalePosWeight :
1.0 xgbr_8da6032c61a9 씨 : 0 xgbr_8da6032c61a9 유성 :
0 xgbr_8da6032c61a9-sketchEps : 0.03
xgbr_8da6032c61a9-skipDrop : 0.0, xgbr_8da6032c61a9-
subsample: 1.0, xgbr_8da6032c61a9-
timeoutRequestWorkers: 1800000, xgbr_8da6032c61a9-
trackerConf: Trac kerConf(0,python), xgbr_8da6032c61a9-
trainTestRatio: 1.0, xgbr_8da6032c61a9-treeLimit: 0,
xgbr_8da6032c61a9-treeMethod: 자동, xgbr_8da6032c61
}

}
```

3장 지도 학습

LightGBM을 사용한 다중 회귀

Listing 3-11 에서는 LightGBM을 사용할 것입니다. LightGBM은 회귀 작업을 위해 특별히 LightGBMRegressor 클래스와 함께 제공됩니다. 주택 데이터 세트와 이전 XGBoost 예제의 대부분의 코드를 재사용합니다.

목록 3-11. LightGBM을 사용한 다중 회귀

```
spark-shell --packages Azure:mmlspark:0.15
```

```
var priceSchema = StructType(배열(
    StructField("avg_area_income", DoubleType, true),
    StructField("avg_area_house_age", DoubleType, true),
    StructField("avg_area_num_rooms", DoubleType, true),
    StructField("avg_area_num_bedrooms", DoubleType, true),
    StructField("area_population", DoubleType, true),
    StructField("가격", DoubleType, true) ))
```

```
val dataDF = spark.read.format("csv") .option("해
```

```
더","true") .schema(pricesSchema) .load("USA_Housing.csv") .na.drop()
```

```
dataDF.printSchema
```

뿌리

```
|-- avg_area_income: double(nullable = true) |--  
avg_area_house_age: double(nullable = true) |--  
avg_area_num_rooms: double(nullable = true) |--  
avg_area_num_bedrooms: double(nullable = true) |-- area_population:  
(nullable = true) |-- price: double(nullable = true)
```

```
dataDF.select("avg_area_income", "avg_area_house_age",
"avg_area_num_rooms") .show
```

avg_area_income	avg_area_house_age	avg_area_num_rooms
79545.45857431678	5.682861321615587	7.009188142792237
79248.64245482568	6.0028998082752425	6.730821019094919
61287.067178656784	5.865889840310001	8.512727430375099
63345.24004622798	7.1882360945186425	5.586728664827653
59982.197225708034	5.040554523106283	7.8393877785120487
80175.7541594853	4.9884077575337145	6.104512439428879
6.025335906887153	8.147759585023431	78394.33927753085
6.9897797477182815	6.620477995185026	59927.66081334963
6.3931209805509015	81885.92718409566	4.423671789897876
8.167688003472351	80527.47208292288	8.09351268063935
5.042746799645982		
50593.69549704281 4.496512793097035 7.467627404008019		
39033.809236982364	7.671755372854428	7.250029317273495
73163.6634410467	6.919534825456555	5.9931879009455695
5.344776176735725	8.406417714534253	73091.86674582321
8.517512711137975	79706.96305765743	5.067889591058972
61929.07701808926 4.788550241805888	5.0970095543775615	8.219771123286257
63508.19429942997	5.947165139552473	7.187773835329727
5.739410843630574	7.09180810424997	62085.27640340488

avg_area_num_bedrooms	area_population	가격
4.09	23086.800502686456	1059033.5578701235 3.09
40173.07217364482	1505890.91484695	5.13
36882.15939970458	1058987.9878760849	

3장 지도 학습

```

+-----+
| 3.26| 34310.24283090706|1260616.8066294468|
| 4.23|26354.109472103148| 630943.4893385402|
| 4.04|26748.428424689715|1068138.0743935304|
| 3.41| 60828.24908540716|1502055.8173744078|
| 2.42|36516.358972493836|1573936.5644777215|
| 2.3| 29387.39600281585| 798869.5328331633|
| 6.1| 40149.96574921337|1545154.8126419624|
| 4.1| 47224.35984022191| 1707045.722158058|
| 4.49|34343.991885578806| 663732.3968963273|
| 3.1| 39220.36146737246|1042814.0978200927|
| 2.27|32326.123139488096|1291331.5184858206|
| 4.37|35521.294033173246|1402818.2101658515|
| 4.01|23929.524053267953|1306674.6599511993|
| 3.12| 39717.81357630952|1556786.6001947748|
| 4.3| 24595.90149782299| 528485.2467305964|
| 5.12|35719.653052030866|1019425.9367578316|
| 5.49|44922.106702293066|1030591.4292116085|
+-----+

```

상위 20개 행만 표시

```
값 기능 = Array("avg_area_income","avg_area_house_age",
"avg_area_num_rooms","avg_area_num_bedrooms","area_population")
```

```
org.apache.spark.ml.feature.VectorAssembler 가져오기
```

```
val 아셈블러 = 새로운 VectorAssembler()
    .setInputCols(기능)
    .setOutputCol("기능")
```

```
val dataDF2 = assembler.transform(dataDF)
```

```
dataDF2.select("가격","기능").show(20,50)
```

```

+-----+
|           가격          |          기능          |
+-----+-----+
|1059033.5578701235|[79545.45857431678,5.682861321615587,7.00918814...|
| 1505890.91484695|[79248.64245482568,6.0028998082752425,6.7308210...|
|1058987.9878760849|[61287.067178656784,5.865889840310001,8.5127274...|

```

3장 지도 학습

```
|1260616.8066294468|[63345.24004622798,7.1882360945186425,5.5867286...| | 630943.4893385402|
[59982.197225708034,5.040554523106283,7.8393877...| |1068138.0743935304|
[80175.7541594853,4.9884077575337145,6.10451243...| |1502055.8173744078|
[64698.46342788773,6.025335906887153,8.14775958...| |1573936.5644777215|
[78394.33927753085,6.9897797477182815,6.6204779...| | 798869.5328331633|
[59927.66081334963,5.36212556960358,6.393120980...| |1545154.8126419624|
[81885.92718409566,4.423671789897876,8.16768800...| | 1707045.722158058|
[80527.47208292288,8.09351268063935,5.042746799...| | 663732.3968963273|
[50593.69549704281,4.496512793097035,7.46762740...| |1042814.0978200927|
[39033.809236982364,7.671755372854428,7.2500293...| |1291331.5184858206|
[73163.6634410467,6.919534825456555,5.993187900...| |1402818.2101658515|
[69391.3801843616,5.344776176735725,8.406417714...| |1306674.6599511993|
[73091.86674582321,5.443156466535474,8.51751271...| |1556786.6001947748|
[79706.96305765743,5.067889591058972,8.21977112...| | 528485.2467305964|
[61929.07701808926,4.788550241805888,5.09700955...| |1019425.9367578316|
[63508.19429942997,5.947165139552473,7.18777383...| |1030591.4292116085|
[62085.27640340488,5.739410843630574,7.09180810...| |
+-----+-----+
```

상위 20개 행만 표시

발 시드 = 1234

```
val Array(trainingData, testData) = dataDF2.randomSplit(Array(0.8, 0.2), 시드)

com.microsoft.ml.spark.{LightGBMRegressionModel,LightGBMRegressor} 가져오기

발 lightgbm = 새로운 LightGBMRegressor() .setFeaturesCol("기
능") .setLabelCol("가격") .setObjective("회
귀")
```

org.apache.spark.ml.tuning.ParamGridBuilder 가져오기

```
val paramGrid = new
ParamGridBuilder() .addGrid(lightgbm.numLeaves,
Array(6, 9)) .addGrid(lightgbm.numIterations, Array(10,
15)) .addGrid(lightgbm.maxDepth, Array(2, 3, 4)) .짓다()
```

3장 지도 학습

```
paramGrid: 배열[org.apache.spark.ml.param.ParamMap] = 배열({
    LightGBMRegressor_f969f7c475b5-maxDepth: 2,
    LightGBMRegressor_f969f7c475b5-num반복: 10,
    LightGBMRegressor_f969f7c475b5-numLeaves: 6
}, {
    LightGBMRegressor_f969f7c475b5-maxDepth: 3,
    LightGBMRegressor_f969f7c475b5-num반복: 10,
    LightGBMRegressor_f969f7c475b5-numLeaves: 6
}, {
    LightGBMRegressor_f969f7c475b5-maxDepth: 4,
    LightGBMRegressor_f969f7c475b5-num반복: 10,
    LightGBMRegressor_f969f7c475b5-numLeaves: 6
}, {
    LightGBMRegressor_f969f7c475b5-maxDepth: 2,
    LightGBMRegressor_f969f7c475b5-num반복: 10,
    LightGBMRegressor_f969f7c475b5-numLeaves: 9
}, {
    LightGBMRegressor_f969f7c475b5-maxDepth: 3,
    LightGBMRegressor_f969f7c475b5-num반복: 10,
    LightGBMRegressor_f969f7c475b5-numLeaves: 9
}, {
    라이...
})
```

org.apache.spark.ml.evaluation.RegressionEvaluator 가져오기

```
값 평가자 = 새로운 RegressionEvaluator() .setLabelCol("가
격") .setPredictionCol("예
측") .setMetricName("rmse")
```

org.apache.spark.ml.tuning.CrossValidator 가져오기

```
val cv = 새로운
```

```
CrossValidator() .setEstimator(lightgbm) .setEvaluator(평가자) .setEstimatorParamMaps(pa
```

```
val 모델 = cv.fit(trainingData)
```

```
val 예측 = model.transform(testData)
```

```
predicts.select("기능","가격","예측").show  
+-----+-----+-----+
```

기능	가격	예측
[17796.6311895433... 302355.83597895555 965317.3181705693		
[35454.7146594754... 1077805.577726322 1093159.8506664087		
[35608.9862370775... 449331.5835333807 1061505.7131801855		
[38868.2503114142... 759044.6879907805 1061505.7131801855		
[40752.7142433209... 560598.5384309639 974582.8481703462		
[41007.4586732745... 494742.5435776913 881891.5646432829		
[41533.0129597444... 682200.3005599922 966417.0064436384		
[42258.7745410484... 852703.2636757497 1070641.7611960804		
[42940.1389392421... 680418.7240122693 1028986.6314725328		
[43192.1144092488... 1054606.9845532854 1087808.2361520242		
[43241.9824225005... 629657.6132544072 889012.3734817103		
[44328.2562966742... 601007.3511604669 828175.3829271109		
[45347.1506816944... 541953.9056802422 860754.7467075661		
[45546.6434075757... 923830.33486809 950407.7970842035		
[45610.9384142094... 961354.287727855 1175429.1179985087		
[45685.2499205068... 867714.3838490517 828812.007346283		
[45990.1237417814... 1043968.3994445396 1204501.1530193759		
[46062.7542664558... 675919.6815570832 973273.6042265462		
[46367.2058588838... 268050.81474351394 761576.9192149616		
[47467.4239151893... 762144.9261238109 951908.0117790927		

상위 20개 행만 표시

```
val rmse = evaluator.evaluate(예측)
```

```
rmse: 더블 = 198601.74726198777
```

3장 지도 학습

각 기능에 대한 기능 중요도 점수를 추출해 보겠습니다.

```
val 모델 = lightgbm.fit(trainingData)
```

```
model.getFeatureImportances("이득")
```

```
res7: 배열[더블] = 배열(1.110789482705408E15, 5.69355224816896E14, 3.25231517467648E14,  
1.16104381056.146, 70)
```

목록의 출력 순서를 기능 벡터(avg_area_income, avg_area_house_age, avg_area_num_rooms, avg_area_num_bedrooms, area_population), avg_area_income가 가장 중요한 기능이고 avg_area_house_age, area_population 및 avg_area_num_rooms가 그 뒤를 잇는 것 같습니다.
가장 덜 중요한 기능은 avg_area_num_bedrooms입니다.

요약

Spark MLlib에 포함된 가장 인기 있는 지도 학습 알고리즘과 XGBoost 및 LightGBM과 같이 외부에서 사용할 수 있는 최신 알고리즘에 대해 논의했습니다. Python용 XGBoost 및 LightGBM에 대한 문서는 온라인에서 풍부하게 제공되지만 Spark에 대한 정보와 예제는 제한적입니다. 이 장은 격차를 해소하는 데 도움이 되는 것을 목표로 합니다.

<https://xgboost.readthedocs.io/en/latest>를 참조하십시오. 예 대해 자세히 알아보기 XGBoost, LightGBM의 경우 <https://lightgbm.readthedocs.io/en/latest> 최신 정보를 가지고 있습니다. Spark MLlib에 포함된 분류 및 회귀 알고리즘 이면의 이론과 수학에 대한 보다 심층적인 내용은 Gareth James, Daniela Witten, Trevor Hastie 및 Robert Tibshirani 의 통계 학습 소개 (Springer, 2017)를 참조하십시오. Trevor Hastie, Robert Tibshirani, Jerome Friedman 의 통계적 학습 요소 (Springer, 2016). Spark MLlib에 대한 자세한 내용은 Apache Spark의 기계 학습 라이브러리(Mllib) 가이드 온라인 (<https://spark.apache.org/docs/latest/ml-guide.html>)을 참조하세요.

참고문헌

- 나. 유대 진주; "E PUR SI MUOVE (AND YET IT MOVES)," 2018, The Book Of Why: 인과의 새로운 과학
- ii. 아파치 스파크; "다항 로지스틱 회귀", spark.apache.org, 2019, https://spark.apache.org/docs/latest/ml_classification-regression.html#multinomial-logistic_회귀
- iii. 게오르기オス 드라코스; "서포트 벡터 머신 대 로지스틱 회귀", forwarddatascience.com, 2018, https://towardsdatascience.com/support-vector-machine_vs-logistic-regression-94cc2975433f
- iv. 아파치 스파크; "다층 퍼셉트론 분류기", spark.apache.org, 2019, https://spark.apache.org/docs/latest/ml_classification-regression.html#multilayer-perceptron_분류기
- v. Analytics Vidhya 콘텐츠 팀; "나무에 대한 완전한 튜토리얼 스크래치 기반 모델링(R 및 Python)," AnalyticsVidhya.com, 2016년, www.analyticsvidhya.com/blog/2016/04/complete_tutorial-tree-based-modeling-scratch-in-python/#one
- vi. 라이트GBM; "범주별 기능을 위한 최적의 분할", lightgbm.readthedocs.io, 2019, <https://lightgbm.readthedocs.io/en/latest/Features.html>
- vii. Joseph Bradley와 Manish Amde; "랜덤 포레스트와 MLLib에서 부스팅," Databricks, 2015, <https://databricks.com/blog/2015/01/21/random-forests-and-boosting-in-mllib.HTML>
- viii. 분석 Vidhya 콘텐츠 팀; "에 대한 엔드 투 엔드 가이드 XGBoost 이면의 수학 이해", analyticsvidhya.com, 2018, www.analyticsvidhya.com/blog/2018/09/an-end-to-end_guide-to-understand-the-math-behind-xgboost/

3장 지도 학습

ix. 벤 고먼; "A Kaggle Master가 Gradient Boosting에 대해 설명합니다."

Kaggle.com, 2017, <http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/>

엑스. XGBoost; "부스트된 트리 소개", xgboost.readthedocs.

아이오, 2019, <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

xi. 아파치 스파크; "앙상블 - RDD 기반 API", spark.apache.

org, 2019, https://spark.apache.org/docs/latest/mllib_ensembles.html#gradient-boosted-trees-gbts

xii. 텐치 천; "언제 GBM(Gradient Boosted Machines)보다 랜덤 포레스트를 사용할까

요?", quora.com, 2015년, www.quora.com/When-would-one-use-Random-Forests-over-Gradient-Boosted-Machines-GBMs

xiii. 에이단 오브라이언 등 알.; "VariantSpark 기계 학습을 위한

Genomics Variants", CSIRO, 2018, <https://bioinformatics.csiro.au/variantspark>

xiv. Denis C. Bauer, et. 알.; "넓은 랜덤 포레스트를 사용하여 유전체학에서 차원의 저주 깨기", Databricks, 2017, <https://databricks.com/blog/2017/07/26/break-of-dimensionality-in-genomics-using-wide-random-forests.html>

xv. 알레산드로 룰리 외 알.; "ReForeSt", github.com, 2017, <https://github.com/alessandrolulli/reforest>

16. 재조림; "임의의 숲 분류 모델을 학습하는 방법

ReForeSt', sites.google.com, 2019, <https://sites.google.com/view/reforest/example?authuser=0>

xvii. 아파치 스파크; "앙상블 - RDD 기반 API", spark.apache.

org, 2019, https://spark.apache.org/docs/latest/mllib_ensembles.html#random-forests

xviii. 콜마이너; CallMiner, 2018년, www.globenewswire.com/news-release/2018/09/27/1577343/0/ "고객을 중요시하지 않는다는 새로운 연구 결과에 따르면 1,360억 달러의 전환 전염병이 발생합니다."

ko/New-research-finds-not-valuing-customers-leads-to-136-billion-switching-epidemic.html

xix. 레드 라이언헬드; "비용 절감을 위한 처방", Bain & Company, 2016, www2.bain.com/Images/BB_Prescription_cutting_costs.PDF

더블 엑스. 알렉스 로렌스; "고객 유지를 위한 5가지 팁
기업가", Forbes, 2012, www.forbes.com/sites/alexlawrence/2012/11/01/five-customer-retention-tips-for-entrepreneurs/

xxi. 데이비드 벅스; "통신 데이터 세트의 이탈", Kaggle, 2017,
www.kaggle.com/becksddf/churn-in-telecoms-dataset

xxii. 제프리 슈메인; "Apache Spark MLlib를 사용하여 Telco 변동을 예측하는 방법", DZone, 2016, <https://dzone.com/articles/how-to-predict-telco-churn-with-apache-spark-mllib>

xxiii. 제이크 호어; "Random Forest에 대한 변수 중요도는 어떻게 계산됩니까?" DisplayR, 2018,
www.displayr.com/how-is-variable-important-calculated-for-a-random-forest/

xxiv. 디드릭 닐슨; "XGBoost로 트리 부스팅", 노르웨이어
과학 대학, 2016, https://brage.bibsys.no/xmlui/bitstream/handle/11250/2433761/16128_전체_텍스트.pdf

xxv. 리나 쇼; "XGBoost: 간결한 기술 개요,"
KDNNuggets, 2017, www.kdnuggets.com/2017/10/xgboost-concrete-technical-overview.html

xxvi. 조현수; "Fast Histogram Optimized Grower, 8배에서 10배까지의 속도 향상", DMLC, 2017, <https://github.com/dmlc/xgboost/tree/1950>

3장 지도 학습

xxvii. XGBoost; "XGBoost4J-Spark로 ML 애플리케이션 구축"

xgboost.readthedocs.io, 2019, https://xgboost.readthedocs.io/en/latest/jvm/xgboost4j_spark_tutorial.html#pipeline-with-hyper-parameter-tunning

xxviii. 제이슨 브라운리; "결정의 수와 크기를 조정하는 방법"

Python에서 XGBoost를 사용하는 나무" machinelearningmastery.com, 2016,
<https://machinelearningmastery.com/tune-number-size-decision-trees-xgboost-python/>

xxix. 로라; "LightGBM 벤치마킹: LightGBM 대 xgboost는 얼마나 빠릅니까?",

medium.com, 2017, <https://medium.com/implodinggradients/benchmarking-lightgbm-how-fast-is-lightgbm-vs-xgboost-15d224568031>

트리플 엑스. 라이트GBM; "속도 및 메모리 사용 최적화", lightgbm.

readthedocs.io, 2019, <https://lightgbm.readthedocs.io/en/cross/Features.html>

xxx. 데이비드 막스; "의사결정 트리: 리프 방식(최상 우선) 및 레벨 방식 트리 탐색",

stackexchange.com, 2018년, <https://datascience.stackexchange.com/questions/26699/decision-trees-leaf-wise-best-first-and-level-wise-tree-traverse>

xxxii. 라이트GBM; "LightGBM 기능", lightgbm.readthedocs.io, 2019, <https://lightgbm.readthedocs.io/en/latest/Features.html>

xxxiii. 실라드 파프카; "다양한 오픈소스 GBM의 성능"

구현," github.com, 2019, <https://github.com/szilard/GBM> 성능

xxxiv. 헤리오 안토니오 소토; "'gain'이 반환하는 기능 중요도는 무엇입니까?", github.com, 2018,

<https://github.com/Microsoft/LightGBM/문제/1842>

xxxv. 사이킷런; "선형 회귀 예제", scikit-learn.org, 2019, https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html

3장 지도 학습

- xxxvi. Hongjian Li, et. al.; "다중을 랜덤 포레스트로 대체
선형 회귀는 채점 기능의 결합 친화도 예측을 개선합니다: 사례 연구로서의 Cyscore,"
nih.gov, 2014, www.ncbi.nlm.nih.gov/pmc/articles/PMC4153907/
- xxxvii. 아리안 판찰; "USA Housing.csv," Kaggle, 2018, www.kaggle.com/aariyan101/usa-housingcsv
- xxxviii. 데이터사이언스센트럴; "RMSE: 제곱 평균 제곱근 오차"
DataSciencecentral.com, 2016, www.statisticshowto.datasciencecentral.com/rmse/

4장

비지도 학습

새로운 지식은 지구상에서 가장 가치 있는 상품입니다. 우리가 더 많은 진실을
다룰수록 우리는 더 부자가 됩니다.

—커트 보네구티

비지도 학습은 레이블이 지정된 응답의 도움 없이 데이터 세트에서 숨겨진 패턴과 구조를 찾는 기계 학습 작업입니다. 비지도 학습은 입력 데이터에만 액세스할 수 있고 훈련 데이터를 사용할 수 없거나 얻기 어려울 때 이상적입니다. 일반적인 방법에는 클러스터링, 주제 모델링, 이상 감지 및 주성분 분석이 포함됩니다.

K-평균을 사용한 클러스터링

클러스터링은 몇 가지 유사점이 있는 레이블이 지정되지 않은 관찰을 그룹화하기 위한 감독되지 않은 기계 학습 작업입니다. 인기 있는 클러스터링 사용 사례에는 고객 세분화, 사기 분석 및 이상 감지가 포함됩니다. 클러스터링은 훈련 데이터가 부족하거나 사용할 수 없는 경우 분류기에 대한 훈련 데이터를 생성하는 데에도 자주 사용됩니다.

K-Means는 클러스터링을 위한 가장 인기 있는 비지도 학습 알고리즘 중 하나입니다.

Spark MLlib에는 K-means⁴로 알려진 K-means의 확장 가능한 구현이 포함되어 있습니다.

그림 4-1은 Iris 데이터 세트의 관측치를 3개의 개별 클러스터로 그룹화하는 K-평균을 보여줍니다.

4장 비지도 학습

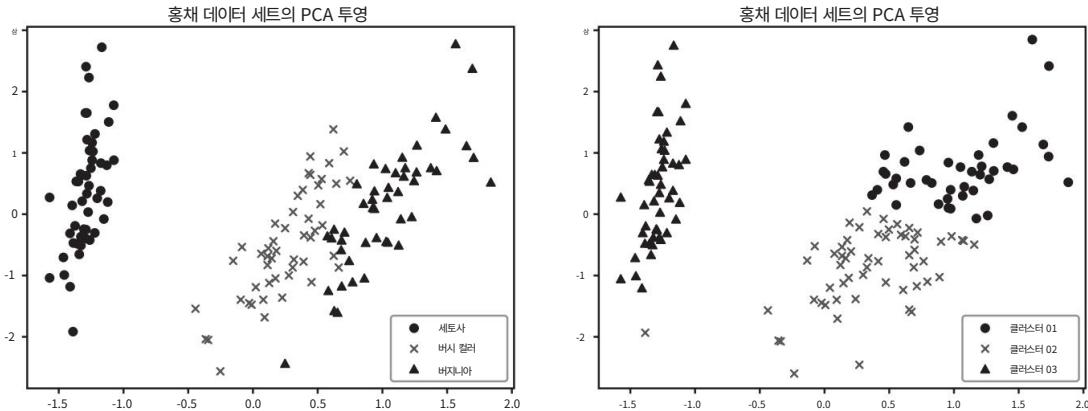


그림 4-1. K-평균을 사용하여 홍채 데이터 세트 클러스터링

그림 4-2 는 작동 중인 K-평균 알고리즘을 보여줍니다. 관측치는 정사각형으로 표시되고 군집 중심은 삼각형으로 표시됩니다. 그림 4-2 (a)는 원본 데이터셋을 보여줍니다. K-Means는 각 클러스터의 시작점으로 사용되는 중심을 무작위로 할당하여 작동합니다(그림 4-2 (b) 및 (c)). 알고리즘은 유clidean 거리를 기반으로 각 데이터 포인트를 가장 가까운 중심에 반복적으로 할당합니다. 그런 다음 해당 클러스터의 일부인 모든 포인트의 평균을 계산하여 각 클러스터에 대한 새로운 중심을 계산합니다(그림 4-2 (d) 및 (e)). 알고리즘은 미리 정의된 반복 횟수에 도달하거나 모든 데이터 포인트가 가장 가까운 중심에 할당되고 더 이상 수행할 수 있는 재할당이 없을 때 반복을 중지합니다(그림 4-2 (f)).

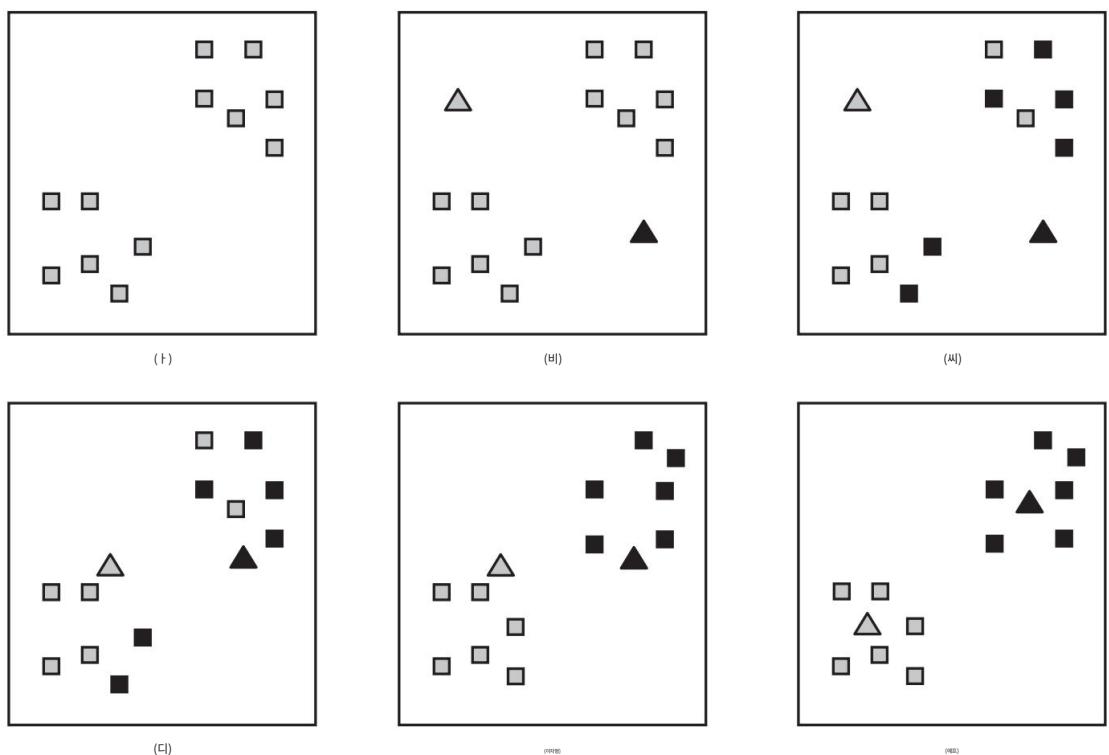


그림 4-2. 실행 중인 K-Means 알고리즘ii

K-Means는 사용자가 알고리즘에 클러스터 k 의 수를 제공하도록 요구합니다. 데이터 세트에 대한 최적의 클러스터 수를 찾는 방법이 있습니다. 팔꿈치와 실루엣 방법은 이 장의 뒷부분에서 논의할 것입니다.

예시

간단한 고객 세분화 예제를 살펴보겠습니다. 7개의 관측값과 3개의 범주형 및 2개의 연속 특성이 혼합된 작은 데이터 세트를 사용합니다. 시작하기 전에 K-평균의 또 다른 한계를 해결해야 합니다. K-Means는 성별 ("M", "F"), 결혼 상태 ("M", "S") 및 주 ("CA", "NY") 와 같은 범주형 기능을 직접 사용할 수 없으며 모든 기능이 연속적이다. 그러나 실제 데이터 세트에는 범주형 및 연속형 기능의 조합이 포함되는 경우가 많습니다. 다행스럽게도 범주형 기능을 숫자 형식으로 변환하여 K-평균을 계속 사용할 수 있습니다.

4장 비지도 학습

이것은 들리는 것처럼 간단하지 않습니다. 예를 들어 결혼 상태를 문자열 표현 "M" 및 "S"에서 숫자로 변환하려면 0을 "M"으로, 1을 "S"로 매핑하는 것이 K-평균에 적합하다고 생각할 수 있습니다. 2 장에서 배웠듯이 이것을 정수 또는 레이블 인코딩이라고 합니다. 그러나 이것은 또 다른 주름을 가져옵니다. 정수에는 K-means와 같은 일부 기계 학습 알고리즘이 잘못 해석할 수 있는 자연스러운 순서 ($0 < 1 < 2$)가 있습니다. 실제로는 정수로 인코딩되기 때문에 한 범주 값이 다른 범주 값보다 "크다"고 생각합니다. 이러한 순서 관계가 데이터에 존재합니다. 이로 인해 예기치 않은 결과가 발생할 수 있습니다. 이 문제를 해결하기 위해 원-핫 인코딩이라는 다른 유형의 인코딩을 사용합니다.ⁱⁱⁱ

범주형 기능을 정수로 변환한 후(StringIndexer 사용), 원-핫 인코딩(OneHotEncoderEstimator 사용)을 사용하여 범주형 기능을 이진 벡터로 나타냅니다. 예를 들어 상태 특성 ("CA", "NY", "MA", "AZ")은 표 4-1에서 원-핫 인코딩됩니다.

표 4-1. 원-핫 인코딩 상태 기능

결국이나	뉴욕	엄마	AZ
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Feature scaling은 K-means의 또 다른 중요한 전처리 단계입니다. 논의

2 장에서 피쳐 스케일링은 거리 계산을 포함하는 많은 기계 학습 알고리즘의 모범 사례이다 요구 사항으로 간주됩니다. 피쳐 스케일링은 데이터가 다른 스케일로 측정되는 경우 특히 중요합니다. 특정 기능은 값의 범위가 상당히 넓어 다른 기능을 지배할 수 있습니다. 피쳐 스케일링은 각 피쳐가 최종 거리에 비례적으로 기여하도록 합니다. 우리의 예에서는 Listing 4-1과 같이 StandardScaler 추정기를 사용하여 평균이 0이고 단위 분산(표준 편차가 1)이 되도록 기능의 크기를 조정합니다.

목록 4-1. K-Means를 사용한 고객 세분화 예

// 샘플 데이터를 생성하여 예제부터 시작하겠습니다.

```
발 custDF = Seq((100,
29000,"M","F","CA",25), (101,
36000,"M","M","CA",46), (102, 5000,
"S","F","NY",18), (103,
68000,"S","M","AZ",39), (104,
2000,"S","F","CA ",16), (105,
75000,"S","F","CA",41), (106,
90000,"M","M","MA",47), (107, 87000,
"S","M","NY",38) ).toDF("고객 ID", "소
득","결혼 상태","성별","주","나이")
```

// 일부 전처리 단계를 수행합니다.

org.apache.spark.ml.feature.StringIndexer 가져오기

```
발 젠더 인덱서 = 새로운 StringIndexer() .setInputCol("성
별") .setOutputCol("성별
Idx")
```

```
val stateIndexer = 새로운
StringIndexer() .setInputCol("상
태") .setOutputCol("state_idx")
```

발 mstatusIndexer = 새로운

```
StringIndexer() .setInputCol("maritalstatus") .setOutputCol("maritalstatus_idx")
```

org.apache.spark.ml.feature.OneHotEncoderEstimator 가져오기

```
val 인코더 = new OneHotEncoderEstimator()
.setInputCols(Array("gender_idx","state_idx",
"maritalstatus_idx")) .setOutputCols(Array("gender_enc","state_enc",
"maritalstatus_enc"))
```

4장 비지도 학습

```
val custDF2 = 성별 인덱서.fit(custDF).transform(custDF)

발 custDF3 = stateIndexer.fit(custDF2).transform(custDF2)

발 custDF4 = mstatusIndexer.fit(custDF3).transform(custDF3)
```

```
custDF4.select("gender_idx","state_idx","maritalstatus_idx").show
```

	gender_idx	state_idx	maritalstatus_idx
1	0.0	0.0	1.0
1	1.0	0.0	1.0
1	0.0	1.0	0.0
1	1.0	3.0	0.0
1	0.0	0.0	0.0
1	0.0	0.0	0.0
1	1.0	2.0	1.0
1	1.0	1.0	0.0

```
val custDF5 = encoder.fit(custDF4).transform(custDF4)
```

```
custDF5.printSchema
```

뿌리

```
|-- customerid: 정수(nullable = false)
|-- 수입: 정수(nullable = false)
|-- 결혼 상태: 문자열(nullable = true)
|-- 성별: 문자열(nullable = true)
|-- 상태: 문자열(nullable = true)
|-- 나이: 정수(nullable = false)
|-- gender_idx: 이중(nullable = false)
|-- state_idx: 이중(nullable = false)
|-- maritalstatus_idx: 이중(nullable = false)
|-- gender_enc: 벡터(nullable = true)
|-- state_enc: 벡터(nullable = true)
|-- maritalstatus_enc: 벡터(nullable = true)
```

```
custDF5.select("gender_enc","state_enc","maritalstatus_enc").show
```

성별	state_enc maritalstatus_enc
(1,[0],[1.0]) (3,[0],[1.0]) (1,[],[])(3,[0],	(1,[],[])
[1.0]) (1,[0],[1.00])(3,[0],[1.00])(1,[0]),[],(3,	(1,[],[])
[0],[1.0]) (3,[0],[1.0]) (1,[],[])(3,[2],[1.0])	(1,[0],[1.0])
(1,[],[])(3,[1],[1.0])	(1,[0],[1.0])
	(1,[0],[1.0])
	(1,[],[])
	(1,[0],[1.0])
	(1,[0],[1.0])
	+-----+-----+-----+

org.apache.spark.ml.feature.VectorAssembler 가져오기

```
val 어셈블러 = 새로운 VectorAssembler()
    .setInputCols(Array("소득", "gender_enc", "state_enc",
    "maritalstatus_enc", "나이"))
    .setOutputCol("기능")
```

발 custDF6 = assembler.transform(custDF5)

custDF6.printSchema

뿌리

```
-- customerid: 정수(nullable = false)
-- 수입: 정수(nullable = false)
-- 결혼 상태: 문자열(nullable = true)
-- 성별: 문자열(nullable = true)
-- 상태: 문자열(nullable = true)
-- 나이: 정수(nullable = false)
-- gender_idx: 이중(nullable = false)
-- state_idx: 이중(nullable = false)
-- maritalstatus_idx: 이중(nullable = false)
-- gender_enc: 벡터(nullable = true)
-- state_enc: 벡터(nullable = true)
-- maritalstatus_enc: 벡터(nullable = true)
-- 기능: 벡터(nullable = true)
```

4장 비지도 학습

```
custDF6.select("기능").show(거짓)

+-----+
|특징          |
+-----+
|[29000.0,1.0,1.0,0.0,0.0,0.0,25.0]||(7,[0,2,6],
[36000.0,1.0,46.0])||[5000.0,1.0,0.0,1.0,0.0,1.0,18.0]
||(7,[0,5,6],[68000.0,1.0,39.0])|||
|[2000.0,1.0,1.0,0.0,0.0,1.0,16.0]|||
|[75000.0,1.0,1.0,0.0,0.0,1.0,41.0]||(7,[0,4,6],
[90000.0,1.0,47.0])||[87000.0,0.0,0.0,1.0,0.0,1.0,38.0]|
```

```
+-----+
```

org.apache.spark.ml.feature.StandardScaler 가져오기

발 스케일러 = 새로운 StandardScaler() .setInputCol("기

```
능") .setOutputCol("scaledFeatures") .setWithStd(true) .setWithMean(false)
```

```
val custDF7 = scaler.fit(custDF6).transform(custDF6)
```

`custDF7.printSchema`

뿌리

```
-- customerid: 정수(nullable = false) -- 수입: 정수(nullable =
false) -- maritalstatus: 문자열(nullable = true) -- 성별: 문자열
(nullable = true) -- state: string (nullable = true) -- age: 정수
(nullable = false) -- gender_idx: double(nullable = false) --
state_idx: double(nullable = false) -- maritalstatus_idx:
double(nullable = false) -- gender_enc: 벡터(nullable = true)
```

```
|-- state_enc: 벡터(nullable = true) |-- maritalstatus_enc:
  벡터(nullable = true) |-- 기능: 벡터(nullable = true) |-- scaledFeatures:
    벡터(nullable = true)
```

```
custDF7.select("scaledFeatures").show(8,65)
```

```
+-----+
| 확장된 기능
+-----+
|[0.8144011366375091,1.8708286933869707,1.8708286933869707,0.0,...|(7,[0,2,6],
[1.0109807213431148,1.8708286933869707,3.7319696616...||(0.1404139890754326,1.870828693
1.9096302514258834,1.9321835661585918,3.1640612348 ... ||
[0.05616559563017304,1.8708286933869707,1.8708286933869707,0.0 ... ||
[2.106209836131489,1.8708286933869707,1.8708286933869707,0.0,0 ... || (7, [0,4,6],
[2.5274518033577866,2.82842712474619,3.813099436871...|
[2.443203409912527,0.0,0.0,2.16024689.90269028

+-----+
```

// 두 개의 클러스터를 생성합니다.

```
org.apache.spark.ml.clustering.KMeans 가져오기
```

```
val kmeans = 새로운
```

```
KMeans() .setFeaturesCol("scaledFeatures") .setPredictionCol("예측") .setK(2)
```

```
org.apache.spark.ml.Pipeline 가져오기
```

```
val 파이프라인 = new
```

```
Pipeline() .setStages(Array(genderIndexer, stateIndexer,
mstatusIndexer, 인코더, 어셈블러, 스케일러, kmeans))
```

```
val 모델 = 파이프라인.fit(custDF)
```

```
val 클러스터 = model.transform(custDF)
```

4장 비지도 학습

```
clusters.select("고객 ID", "수입", "결혼 상태",
    "성별", "주", "나이", "예측")
    .보여 주다
```

고객 ID	수입	결혼 상태	성별	주	나이	예측
100	29000	남	남	캘리포니아	25	1
101	36000	남	남	캘리포니아	46	0
102	5000	103	여	뉴욕	18	1
68000	104	여	남	아조	39	0
2000년	105	여	여	캘리포니아	16	1
75000	106	여	여	캘리포니아	41	0
90000	107	남	남	MA	47	0
87000		여	남	뉴욕	38	0

org.apache.spark.ml.clustering.KMeansModel 가져오기

```
val 모델 = 파이프라인.stages.last.asInstanceOf[KMeansModel]
```

```
model.clusterCenters.foreach(println)
[1.9994952044341603,0.37416573867739417,0.7483314773547883,0.4320493798938574,
0.565685424949238,1.159310139695155,3.4236765156588613]
[0.3369935737810382,1.8708286933869707,1.247219128924647,0.7200822998230956,
0.0,1.288122377439061,1.5955522466340666]
```

WSSSE(집합 오차의 집합 내)를 계산하여 클러스터를 평가합니다.

"팔꿈치 방법"을 사용하여 WSSSE를 검사하는 것은 최적의 클러스터 수를 결정하는 데 도움이 되는 경우가 많습니다. 엘보 우 방법은 모델을 k 값의 범위로 맞추고 WSSSE에 대해 플로팅하여 작동합니다. 꺾은선형 차트를 육안으로 검사하고 구부러 진 팔과 비슷하면 곡선에서 구부러지는 지점("팔꿈치")이 k에 대한 최적의 값을 나타냅니다.

```
val wssse = model.computeCost(custDF)
wssse: 더블 = 32.09801038868844
```

4장 비지도 학습

클러스터 품질을 평가하는 또 다른 방법은 실루엣 계수 점수를 계산하는 것입니다. 실루엣 점수는 한 군집의 각 점이 다른 군집의 점과 얼마나 가까운지에 대한 메트릭을 제공합니다. 실루엣 점수가 클수록 클러스터의 품질이 좋습니다. 점수가 1에 가까울수록 포인트가 클러스터의 중심에 더 가깝다는 것을 나타냅니다. 0에 가까운 점수는 포인트가 다른 클러스터에 더 가깝다는 것을 나타내고 음수 값은 포인트가 잘못된 클러스터에 지정되었을 수 있음을 나타냅니다.

`org.apache.spark.ml.evaluation.ClusteringEvaluator` 가져오기

가치 평가자 = 새로운 `ClusteringEvaluator()`

`val 실루엣 = evaluator.evaluate(클러스터)`

실루엣: 더블 = 0.6722088068201866

LDA(Latent Dirichlet Allocation)를 사용한 주제 모델링

LDA(Latent Dirichlet Allocation)는 2003년 David M. Blei, Andrew Ng 및 Michael Jordan에 의해 개발되었지만 인구 유전학에 사용된 유사한 알고리즘은 2000년 Jonathan K. Pritchard, Matthew Stephens 및 Peter Donnelly도 제안했습니다. LDA, 머신 러닝에 적용되는 그래픽 모델을 기반으로 하며 GraphX를 기반으로 구축된 Spark MLlib에 포함된 첫 번째 알고리즘입니다. Latent Dirichlet Allocation은 토픽 모델링에 널리 사용됩니다. 주제 모델은 문서 그룹에서 주제(또는 주제)를 자동으로 파생합니다(그림 4-3). 이러한 주제는 콘텐츠 기반 권장 사항, 문서 분류, 차원 축소 및 기능화에 사용할 수 있습니다.

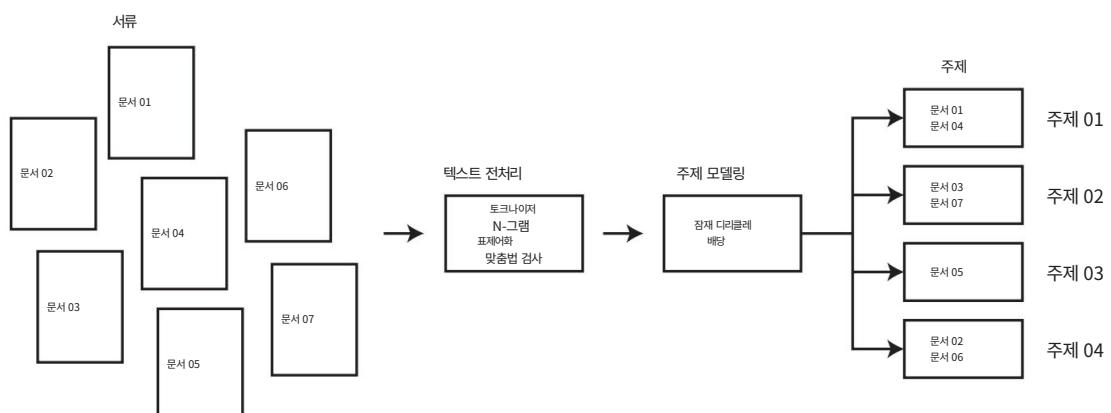


그림 4-3. Latent Dirichlet Allocation을 사용하여 주제별로 문서 그룹화

4장 비지도 학습

Spark MLlib의 광범위한 텍스트 마이닝 및 사전 처리 기능에도 불구하고 몇 가지만 언급하면 대부분의 엔터프라이즈급 NLP 라이브러리에서 찾을 수 있는 표제어 추출, 형태소 분석 및 감정 분석과 같은 몇 가지 기능이 부족합니다. 이 장의 뒷부분에 나오는 주제 모델링 예제를 위해 이러한 기능 중 일부가 필요합니다. Stanford CoreNLP for Spark 및 Spark NLP를 John Snow Labs에서 소개할 좋은 시간입니다.

Spark용 스탠퍼드 CoreNLP

Stanford CoreNLP는 Stanford University의 NLP 연구 그룹에서 개발한 전문가 수준의 NLP 라이브러리입니다. CoreNLP는 아랍어, 중국어, 영어, 프랑스어 및 독일어와 같은 여러 언어를 지원합니다. 네이티브 Java API와 웹 API 및 명령줄 인터페이스를 제공합니다. R, Python, Ruby 및 Lua와 같은 주요 프로그래밍 언어를 위한 타사 API도 있습니다. Databricks의 소프트웨어 엔지니어인 Xiangrui Meng은 Spark용 Stanford CoreNLP 래퍼를 개발했습니다(목록 4-2 참조).

목록 4-2. Spark용 Stanford CoreNLP에 대한 간략한 소개

```
spark-shell --packages databricks:spark-corenlp:0.4.0-spark2.4-scala2.11 --jars stanford-corenlp-3.9.1-models.jar
```

```
가져오기 spark.implicits._  
org.apache.spark.sql.types._ 가져오기
```

```
값 데이터DF = Seq(  
(1, "케빈 듀란트는 2019년 올스타 NBA MVP였습니다."),  
(2, "스테판 커리는 NBA 최고의 클러치 3점 슈터다."),  
(3, "내 게임은 20년 전만큼 좋지 않다."),  
(4, "마이클 조던은 역대 최고의 NBA 선수입니다."),  
(5, "레이커스는 현재 NBA에서 최악의 성적을 기록하고 있다."))  
.toDF("아이디", "텍스트")
```

```
dataDF.show(거짓)
```

```
+-----+
|아이디 |텍스트
+-----+
```

|1 |Kevin Durant는 2019년 NBA 올스타 MVP였습니다. | 2 |Stephen Curry는 NBA 최고의 클러치 3점 슈터입니다. ||
 3 |내 게임은 20년 전만큼 좋지 않다. | 4 |마이클 조던은 NBA 역사상 가장 위대한 선수입니다. | 5 |레이커스는 현재 NBA에
 서 최악의 성적을 기록하고 있습니다.|

```
+-----+
```

// Stanford CoreNLP를 사용하면 텍스트 처리 기능을 연결할 수 있습니다. 문서를 // 분할하여 문장으로 만든 다음 문장을 단어로 토큰화합니다.

가져오기 com.databricks.spark.corenlp.functions._

값 데이터DF2 = 데이터DF

```
.select(explode(ssplit('text)).as('sen)) .select('sen,
 tokenize('sen).as('words))
```

dataDF2.show(5,30)

```
+-----+-----+-----+
| | |센| | | |단어|
+-----+-----+-----+
```

|Kevin Durant는 2019 A...|[Kevin, Durant, was, 2...| |Stephen Curry는 최고의 c...|[Stephen,
 Curry, is, b...] |내 게임은 나만큼 좋지 않아...|[내, 게임, ~이다, ~처럼, ~처럼, 좋지 않다...| |마이클 조던
 은 위대하다...|[마이클 조던은, ...| |레이커스는 현재 o...|[레이커스, 현재, 하...|

```
+-----+-----+-----+
```

// 문장에 대한 감정 분석을 수행합니다. // 스케일의 범위는 강한 부정의 경우 0에서 강한 긍정의
 경우 4입니다.

val dataDF3 =

```
dataDF .select(explode(ssplit('text)).as('sen)) .select('sen,
 tokenize('sen).as('words), 감정('sen).as('sentiment) ))
```

4장 비지도 학습

```
dataDF3.show(5,30)
```

	센	단어감정	
Kevin Durant는 2019 A... [Kevin, Durant, was, 2,...]	Stephen Curry는 최고의 c... [Stephen,	1	
Curry, is, b... 내 게임은 나만큼 좋지 않아... [내, 게임, ~이다, ~처럼, ~처럼, 좋지 않다...]	마이클 조던	3	
은 위대하다... [마이클 조던은, ...]	레이커스는 현재 o... [레이커스, 현재, 하...]	1	
		3	
		1	

사용 가능한 전체 기능 목록을 보려면 Databricks의 CoreNLP GitHub 페이지를 방문하세요.
Spark용 스탠포드 CoreNLP.

John Snow Labs의 Spark NLP

John Snow Labs의 Spark NLP 라이브러리는 기본적으로 Spark ML Pipeline API를 지원합니다. Scala로 작성되었으며 Scala와 Python API를 모두 포함합니다. 여기에는 토크나이저, 표제어, 형태소 분석기, 엔터티 및 날짜 추출기, 품사 태거, 문장 경계 감지, 맞춤법 검사기 및 명명된 엔터티 인식과 같은 몇 가지 고급 기능이 포함됩니다.

주석자는 Spark NLP에서 NLP 기능을 제공합니다. 주석은 결과입니다

Spark NLP 작업. Annotator에는 Annotator 접근 방식과 Annotator 모델의 두 가지 유형이 있습니다.

Annotator Approaches는 Spark MLlib 추정기를 나타냅니다. 데이터가 있는 모델을 피팅하여 주석 모델 또는 변환기를 생성합니다. Annotator Model은 데이터세트를 가져와서 주석의 결과로 열을 추가하는 변환기입니다.

Spark 추정기 및 변환기로 표시되기 때문에 주석을 Spark Pipeline API와 쉽게 통합할 수 있습니다. Spark NLP는 사용자가 해당 기능에 액세스할 수 있는 여러 방법을 제공합니다.vi

사전 훈련된 파이프라인

Spark NLP에는 빠른 텍스트 주석을 위해 사전 훈련된 파이프라인이 포함되어 있습니다. Spark NLP는 텍스트를 입력으로 받아들이는 Explain_document_ml이라는 사전 훈련된 파이프라인을 제공합니다 (목록 4-3 참조). 사전 훈련된 파이프라인에는 널리 사용되는 텍스트 처리 기능이 포함되어 있으며 많은 번 거로움 없이 Spark NLP를 사용하는 빠르고 더러운 방법을 제공합니다.

목록 4-3. Spark NLP 사전 훈련된 파이프라인 예

```
spark-shell --package com.johnsnowlabs.nlp.pretrained.PretrainedPipeline 가져오기

val annotations = PretrainedPipeline("explain_document_ml").annotate("지
난 여름에 그리스를 방문했습니다. 멋진 여행이었습니다. 미코노스에서 수영하러 갔습니다.")

annotations("sentence")
res7: Seq[String] = List(저는 지난 여름에 그리스를 방문했습니다., 멋
진 여행이었습니다., 저는 미코노스에서 수영을 했습니다.)

annotations("token") res8:
Seq[String] = List(I, Visited, Greece, last, summer, ., It, was, a, great, trip, ., I, 갔다,
Swimming, in, Mykonos, .)

annotations("lemma") res9:
Seq[String] = List(I, Visit, Greece, last, summer, ., It, be, a, great, trip, ., I, go, swim,
in, Mykonos, .)
```

Spark DataFrame으로 사전 훈련된 파이프라인

사전 훈련된 파이프라인은 Listing 4-4 와 같이 Spark DataFrames에서도 작동 합니다.

목록 4-4. Spark DataFrames를 사용한 Spark NLP 사전 훈련된 파이프라인

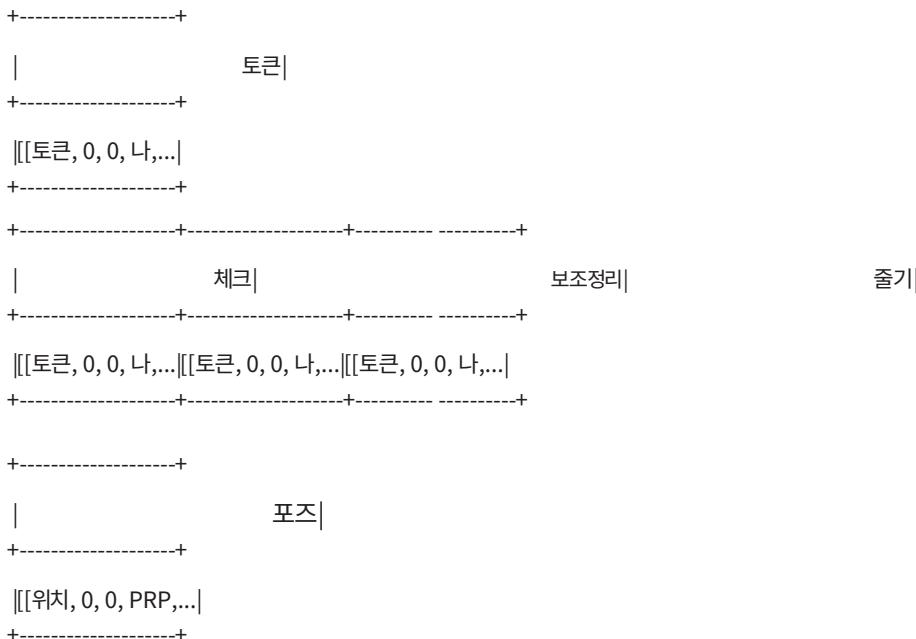
```
val data = Seq("지난 여름에 그리스를 방문했습니다. 멋진 여행이었습니다. 미코노스에서 수영하러 갔습니
다.").toDF("text")

val 주석 = PretrainedPipeline("explain_document_ml").transform(data)

주석.show()

+-----+-----+-----+
|      텍스트|      문서|      문장|
+-----+-----+-----+
|그리스를 방문했습니다 ...|[문서, 0, 77...|[문서, 0, 28...|
+-----+-----+-----+
```

4장 비지도 학습



Spark MLlib 파이프라인으로 사전 훈련된 파이프라인

Spark MLlib 파이프라인과 함께 사전 훈련된 파이프라인을 사용할 수 있습니다(목록 4-5 참조). 사람이 읽을 수 있는 형식으로 토큰을 표시하려면 Finisher라는 특수 변환기 가 필요합니다.

목록 4-5. Spark MLlib 파이프라인 예제를 사용하여 사전 훈련된 파이프라인

```

import com.johnsnowlabs.nlp.Finisher
org.apache.spark.ml.Pipeline 가져오기

val data = Seq("지난 여름에 그리스를 방문했습니다. 멋진 여행이었습니다. 미코노스에서 수영하러 갔습니다.").toDF("text")

val 피니셔 = new Finisher()
.setInputCols("문장", "토큰", "정리 보조표")

val ExplainPipeline = PretrainedPipeline("explain_document_ml").model

val 파이프라인 = new Pipeline()
.setStages(배열(파이프라인 설명, 완료자))

```

```
pipeline.fit(data).transform(data).show(false)
```

```
+-----+  
| 텍스트 |  
+-----+
```

```
| 나는 지난 여름에 그리스를 방문했다. 멋진 여행이었습니다.|  
+-----+
```

```
+-----+  
| 텍스트 |  
+-----+
```

```
| 미코노스에 수영하러 갔어요.|  
+-----+
```

```
+-----+  
| 완료_문장 |  
+-----+
```

```
| [작년 여름에 그리스를 다녀왔습니다., 좋은 여행이었습니다.|  
+-----+
```

```
+-----+  
| 완성된 문장 |  
+-----+
```

```
| [,미코노스에서 수영하러 갔어요.]|  
+-----+
```

```
+-----+  
| finished_token |  
+-----+
```

```
| [나는, 방문했다, 그리스, 지난, 여름, ., 그것은, 이었다, 대단하다, 여행, .,]|  
+-----+
```

```
+-----+  
| finished_lemma |  
+-----+
```

```
| [나, 방문, 그리스, 지난, 여름, ., It, be, Great, 여행, .,]|  
+-----+
```

4장 비지도 학습

+-----+	
완성된_정리	
+-----+	
[, 나, 가고, 해엄치고, 인, 미코노스, .]	
+-----+	

나만의 Spark MLlib 파이프라인 만들기

Listing 4-6 과 같이 자신의 Spark MLlib 파이프라인에서 직접 주석자를 사용할 수 있습니다.

목록 4-6. 나만의 Spark MLlib 파이프라인 예제 만들기

```
com.johnsnowlabs.nlp.base._ 가져오기
com.johnsnowlabs.nlp.annotator._ 가져오기
org.apache.spark.ml.Pipeline 가져오기

val data = Seq("지난 여름에 그리스를 방문했습니다. 멋진 여행이었습니다. 미코노스에서 수영하러 갔습니다.").toDF("text")

val documentAssembler = 새로운
    DocumentAssembler() .setInputCol("텍스트") .setOutputCol("문서")

val sentenceDetector = 새로운
    SentenceDetector() .setInputCols(Array("문서")) .setOutputCol("문장")

val regexTokenizer = 새로운
    Tokenizer() .setInputCols(Array("문장")) .setOutputCol("토큰")

val 피니셔 = new
    Finisher() .setInputCols("토큰") .setCleanAnnotations(거짓)

val 파이프라인 = new
    Pipeline() .setStages(Array(documentAssembler, sentenceDetector, regexTokenizer, finisher))
```

```
파이프라인.fit(Seq.empty[문자열].toDF("텍스트"))
```

```
.변환(데이터)
```

```
.보여 주다()
```

문장	문서	텍스트
----	----	-----

```
| 그리스를 방문했습니다 ...|[문서, 0, 77...|[문서, 0, 28...|
```

```
+-----+-----+-----+
```

완성된 토큰	토큰
--------	----

```
|[토큰, 0, 0, 나,...|[나, 방문, 그리...|
```

```
+-----+-----+-----+
```

스파크 NLP 라이트파이프라인

Spark NLP는 LightPipeline이라는 또 다른 클래스의 파이프라인을 제공합니다. Spark MLlib 파이프라인과 유사하지만 Spark의 분산 처리 기능을 활용하는 대신 로컬에서 실행됩니다. LightPipeline은 소량의 데이터를 처리하고 저지연 실행이 필요할 때 적합합니다(목록 4-7 참조).

목록 4-7. Spark NLP LightPipelines 예

```
import com.johnsnowlabs.nlp.base._
```

```
val 훈련된 모델 = 파이프라인.fit(Seq.empty[문자열].toDF("텍스트"))
```

```
val lightPipeline = 새로운 LightPipeline(trainedModel)
```

```
lightPipeline.annotate("저는 지난 여름에 그리스를 방문했습니다.")
```

스파크 NLP OCR 모듈

Spark NLP에는 사용자가 PDF 파일에서 Spark DataFrame을 생성할 수 있는 OCR 모듈이 포함되어 있습니다. OCR 모듈은 핵심 Spark NLP 라이브러리에 포함되어 있지 않습니다. 이를 사용하려면 Listing 4-8 의 spark-shell 명령에서 볼 수 있는 것처럼 별도의 패키지를 포함하고 추가 저장소를 지정해야 합니다.

4장 비지도 학습

목록 4-8. Spark NLP OCR 모듈 예

```
spark-shell --package JohnSnowLabs:spark-nlp:2.1.0,com.johnsnowlabs.  
nlp:spark-nlp-ocr_2.11:2.1.0,avax.media.jai:com.springsource.avax.media.  
jai.core:1.1.3  
--repositories http://repo.spring.io/plugins-release
```

```
com.johnsnowlabs.nlp.util.io.OcrHelper 가져오기
```

```
val myOcrHelper = 새로운 OcrHelper
```

```
val 데이터 = myOcrHelper.createDataset(스파크, "/my_pdf_files/")
```

```
val documentAssembler = new DocumentAssembler().setInputCol("텍스트")
```

```
documentAssembler.transform(data).select("텍스트", "파일 이름").show(1,45)
```

```
+-----+  
| | 텍스트 |  
+-----+
```

```
| 이것은 PDF 문서입니다. 좋은 하루 되세요. |  
+-----+
```

```
+-----+  
| 파일 이름 |  
+-----+  
| 파일:/my_pdf_files/document.pdf |  
+-----+
```

Spark NLP는 이 문서에서 다루지 않은 훨씬 더 많은 기능을 포함하는 강력한 라이브러리입니다.

소개. Spark NLP에 대해 자세히 알아보려면 <http://nlp.johnsnowlabs.com>을 방문하세요.

예시

이제 필요한 모든 것이 준비되었으므로 주제 모델링 예제를 진행할 수 있습니다.

Latent Dirichlet Allocation을 사용하여 15년 동안 발행된 주제별로 백만 개 이상의 뉴스 헤드라인을 분류합니다.

Kaggle에서 다운로드할 수 있는 데이터 세트는 Australian Broadcasting Corporation에서 제공했으며

로히트 쿨카르니.

John Snow Labs의 Spark NLP 또는 Stanford CoreNLP 패키지를 사용하여 추가 텍스트 처리 기능을 제공할 수 있습니다. 이 예제에서는 Stanford CoreNLP 패키지를 사용할 것입니다(목록 4-9 참조).

목록 4-9. LDA를 사용한 주제 모델링

```
spark-shell --packages databricks:spark-corenlp:0.4.0-spark2.4-scala2.11 --jars stanford-corenlp-3.9.1-models.jar
```

```
org.apache.spark.sql.functions._ 가져오기
```

```
org.apache.spark.sql.types._ 가져오기
```

```
org.apache.spark.sql._ 가져오기
```

```
// 스키마를 정의합니다.
```

```
var newsSchema = StructType(배열(
    StructField("publish_date", IntegerType, true),
    StructField("headline_text", StringType, true)
))
```

```
// 데이터를 읽습니다.
```

```
val dataDF = spark.read.format("csv")
    .option("헤더", "참")
    .schema(newsSchema)
    .load("abcnews-date-text.csv")
```

```
// 데이터를 검사합니다.
```

```
dataDF.show(가짓)
```

```
+-----+-----+-----+
```

```
|게시일|제목_텍스트
```

```
+-----+-----+-----+
```

```
|20030219 | aba는 커뮤니티 방송 라이센스에 반대하기로 결정|
```

```
20030219 | 법률 화재 목격자는 명예 훼손에 대해 알고 있어야 합니다 |
```

```
20030219 | jag는 인프라 보호 정상 회담을 요구 |
```

```
20030219 | 에어 NZ 직원, 임금 인상을 위한 파업 |
```

```
20030219 | 에어 nz 파업, 호주 여행자에게 영향을 미침 |
```

```
20030219 | 야심 찬 올슨, 트리플 점프 우승 |
```

```
20030219 | 기록을 깨는 바르셀로나에 기뻐하는 익살 |
```

```
20030219 | 호주 예산 stousur가 4개의 멤피스 경기를 낭비 |
```

4장 비지도 학습

20030219	aust, 이라크에 대한 유엔 안보리 연설
20030219	호주는 전쟁 시간표에 참가 있습니다.
20030219	호주, 이라크에 1천만 달러 기부
20030219	barcas는 robson이 생일을 축하하면서 기록을 세웁니다
20030219	목욕탕 계획은 앞으로 나아갑니다
20030219	론세스톤 사이클링 챔피언십에 대한 큰 희망
20030219	파루 물 공급을 늘리는 큰 계획
20030219	블리자드, 미국을 지폐에 묻다
20030219	여단, 군대가 괴롭힘을 당했다는 보고를 기각
20030219	쿠웨이트에 매일 도착하는 영국군 전투병
200302	브라이언트는 레이커스를 리드하여 연장전 승리를 두 배로 늘렸습니다
	산불 피해자들에게 센터링크 방문 촉구

+-----+

상위 20개 행만 표시

// 구두점을 제거합니다.

값 데이터DF2 = 데이터DF

```
.withColumn("headline_text",
    regexp_replace((dataDF("headline_text")), "[^a-zA-Z0-9 ]", ""))
```

// 우리는 표제어를 수행하기 위해 Stanford CoreNLP를 사용할 것입니다. 논의
// 이전에 표제어는 굴절된 단어의 루트 형식을 파생합니다. 을위한
// 예시, "camping", "camps", "camper", "camped"는 모두 굴절
// "캠프"의 형태. 굴절된 단어를 어근 형태로 줄이면 // 자연어 처리를 수행하는 복잡성을 줄이는 데 도움이 됩니다. 비슷한

// 형태소 분석으로 알려진 프로세스는 굴절된 단어를 루트로 줄입니다.

// 형식이지만 접사를 조잡하게 잘라서 그렇게 합니다.

// 루트 형식은 유효한 단어가 아닐 수 있습니다. 대조적으로, 표제어는 다음을 보장합니다.

// 굴절된 단어는 다음을 통해 유효한 루트 단어로 축소됩니다.

// 단어의 형태학적 분석 및 어휘 사용.vii

가져오기 com.databricks.spark.corenlp.functions._

값 데이터DF3 = 데이터DF2

```
.select(explode(ssplit('headline_text')).as('sen'))
    .select('sen, 보조 정리('sen)
        .as('단어))
```

```
dataDF3.show
```

센	단어
아바가 또 결정... [아바, 결정, 아가...] 액트 화재 목격자... [행 위, 화재, 목격자...] ag는 inf...를 호출합니다.[a, g, call, for,...] 에어 nz 스태프 in a... [에어, nz, 스태프, ...] 에어 nz 스트라이크 ~에 ... [에어, nz, 스트라이크, ...] 야심찬 올슨 ... [야심차게, 올소...] 익숙한 기쁨의 w... [흥분한, 기뻐하는...] 호주 한정자 ... [호주, 예선...] aust 주소 un... [aust, 주소, u...] 호주는 락이다... [호주, 나...] 호주에서 계속... [호주, to, c...] 바르카 테이크 레코드... [바르카, 테이크, 레코...] 목욕탕 계획... [목욕탕, 계획,...] 라우에 대한 큰 희망... [큰, 희망, ...] 에 대한, ...] 향상시킬 큰 계획... [큰, 계획, ~을, b...] 블리자드 가 널 묻어... [블리자드, 묻어, ...] 준장 해산... [준장, 해산...] 영국식 전투 tr... [영국식, 전투,...] 브라이언트 리드 레이크... [브라이언트, 리드, 라...] 산불 피해자... [산불, 피해자...]	

```
+-----+-----+
```

상위 20개 행만 표시

```
// "a", "be", "to"와 같은 불용어를 제거합니다. 중지 // 단어는 문서의 의미에 기여하  
지 않습니다.
```

```
org.apache.spark.ml.feature.StopWordsRemover 가져오기
```

```
val 제거기 = 새로운
```

```
StopWordsRemover() .setInputCol("단어") .setOutputCol("filtered_stopwords")
```

4장 비지도 학습

```
val dataDF4 = remover.transform(dataDF3)
```

```
dataDF4.show
```

	센	단어 filter_stopwords
+-----+	-----+-----+	+-----+

|아바가 또 결정...|[아바, 결정, 아가...|[아바, 결정, 컴...| |액트 화재 목격자...|[행위, 화재, 목격자...|[행위,
화재, 목격자...| |ag는 inf...를 호출합니다.|[a, g, call, for,...|[g, call, infrast...| |에어 nz 직원 in a...|[에
어, nz, 직원, ...|[에어, nz, 직원, ...| |에어 nz 스트라이크 ...|[에어, nz, 스트라이크,...|[에어, nz, 스트라이
크,...| |야심찬 올슨 ...|[야심차게, 올쏘...|[야심차게, 올쏘...| |익숙한 기쁨 w...|[익숙한, 기뻐하는...|[당당한,
기뻐하는...| |호주 한정자 ...|[호주, 예선...|[호주, 예선...| |aust 주소 un...|[aust, address, u...|[aust,
address, u...| |오스트레일리아는 자물쇠입니다...|[오스트레일리아, be, l...|[오스트레일리아, 자물쇠,...| |호
주에서 계속...|[호주, to, c...|[호주, contr...| |바르카 기록...|[바르카, 기록, 기록...|[바르카, 기록, 기록...|| |
목욕탕 계획...|[목욕탕, 계획,...|[목욕탕, 계획,...| |라우에 대한 큰 희망...|[큰, 희망, ...에 대한, ...|[큰, 희망,
발사...| |큰 계획을 부스트...|[큰, 계획, ~을, b...|[큰, 계획, 부스트...| |블리자드가 당신을 묻어...|[블리자드,
묻어, ...|[블리자드, 묻어, ...| |준장 해산...|[준장, 디스미...|[준장, 디스미...| |영국식 전투 tr...|[영국식, 전
투,...|[영국식, 전투,...| |브라이언트 리드 레이크...|[브라이언트, 리드, 라...|[브라이언트, 리드, 라...| |산불 피
해자...|[산불, 피해자...|[산불, 피해자...|

	센	단어 filter_stopwords
+-----+	-----+-----+	+-----+

상위 20개 행만 표시

// n-그램을 생성합니다. n-gram은 문서에서 단어의 관계를 발견하는 데 자주 사용되는 // "n"개의 단어 시퀀스입니다. 예
를 들어 // "Los Angeles"는 바이그램입니다. "Los"와 "Angeles"는 유니그램입니다. "Los"와 // "Angeles"는 개별 단위
로 간주할 때 큰 의미가 없을 수 있지만 // 단일 개체인 "Los Angeles"로 결합될 때 더 의미가 있습니다.

// 최적의 "n" 수를 결정하는 것은 사용 사례와 // 문서에 사용된 언어에 따라 다릅니다.viii 이 예에서는 유니그램, 바
이그램 및 트라이그램을 // 생성합니다. 212

org.apache.spark.ml.feature.NGram 가져오기

val unigram = 새로운

NGram() .setN(1) .setInputCol("filtered_stopwords") .setOutputCol("unigram_words")

val dataDF5 = unigram.transform(dataDF4)

dataDF5.printSchema 를

트

```
|-- sen: 문자열(nullable = true) |-- 단어: 배열
(nullable = true) | |-- 요소: 문자열(containsNull
= true) |-- filtered_stopwords: 배열(nullable = true) |--|
unigram_words: 배열(nullable = true) |
```

|-- 요소: 문자열(containsNull = false)

val bigram = 새로운

NGram() .setN(2) .setInputCol("filtered_stopwords") .setOutputCol("bigram_words")

val dataDF6 = bigram.transform(dataDF5)

dataDF6.printSchema 를

트

```
|-- sen: 문자열(nullable = true) |-- 단어: 배열
(nullable = true) | |-- 요소: 문자열(containsNull
= true) |-- filtered_stopwords: 배열(nullable = true) |--|
unigram_words: 배열(nullable = true) ||-- bigram_words: 배열(nullable = true) |
```

|-- 요소: 문자열(containsNull = false)

|-- 요소: 문자열(containsNull = false)

4장 비지도 학습

발 트라이그램 = 새로운 NGram()

```
.setN(3)
.setInputCol("filtered_stopwords")
.setOutputCol("trigram_words")
```

```
val dataDF7 = trigram.transform(dataDF6)
```

```
dataDF7.printSchema
```

뿌리

```
|-- sen: 문자열(nullable = true)
|-- 단어: 배열(nullable = true)
| |-- 요소: 문자열(containsNull = true)
|-- filtered_stopwords: 배열(nullable = true)
| |-- 요소: 문자열(containsNull = true)
|-- unigram_words: 배열(nullable = true)
    |-- 요소: 문자열(containsNull = false)
| |-- bigram_words: 배열(nullable = true)
    |-- 요소: 문자열(containsNull = false)
| |-- trigram_words: 배열(nullable = true)
    |-- 요소: 문자열(containsNull = false)
```

// 유니그램, 바이그램, 트라이그램을 하나의 어휘로 결합합니다.

// "ngram_words" 열에 단어를 연결하고 저장합니다.

// 스파크 SQL 사용.

```
dataDF7.createOrReplaceTempView("dataDF7")
```

```
val dataDF8 = spark.sql("sen,words,filtered_stopwords,unigram_words,
bigram_words,trigram_words,concat(concat(unigram_words,bigram_words), trigram_words)를 dataDF7
에서 ngram_words로 선택")
```

```
dataDF8.printSchema
```

뿌리

```
|-- sen: 문자열(nullable = true)
|-- 단어: 배열(nullable = true)
| |-- 요소: 문자열(containsNull = true)
|-- filtered_stopwords: 배열(nullable = true)
| |-- 요소: 문자열(containsNull = true)
```

```

|-- unigram_words: 배열(nullable = true) ||-- bigram_words:
배열(nullable = true) ||-- ngram_words: 배열(nullable =
true) ||-- 요소: 문자열(containsNull = false)

    |-- 요소: 문자열(containsNull = false)

    |-- 요소: 문자열(containsNull = false)

dataDF8.select("ngram_words").show(20,65)
+-----+
|ngram_words
+-----+
|[aba, 결정, 커뮤니티, 방송, 라이센스, aba 결정, de...| [[행위, 화재, 증인, 필수, 인식, 명예 훼손, 행위 화재, 화
재 w...| [[g, 통화, 인프라, 보호, 정상 회담, g 통화, 통화 정보...| [[air, nz, staff, aust, 파업, 급여, 상승, air
nz, nz 직원, st...| [[에어, nz, 스트라이크, 에펙트, 오스트레일리아, 트래블러, 에어 nz, nz st...| [[야심찬, 올슨,
승리, 트리플, 점프, 야심찬 올슨, 올소...| [[antic, 기쁘게, 기록, 휴식, barca, antic 기쁘게, deli...| [[호주, 한정
자, stosur, 낭비, 4, 멤피스, 경기, 호주...| [[aust, address, un, security, Council, iraq, aust address,
추가...| [[호주, 자물쇠, 전쟁, 시간표, opp, 호주 자물쇠, 자물쇠 WA...| [[호주, 기여, 10,000,000, 원조, 이라
크, 호주 계속...| [[바르카, 테이크, 녹음, 롭슨, 축하, 생일, 바르카 테이크,...| [[목욕탕, 계획, 이동, 앞으로, 목욕
탕 계획, 계획 이동, 이동...| [[큰, 희망, 론서스턴, 사이클링, 챔피언십, 큰 희망, 희망 ...| [[큰, 계획, 부스트, 파루,
물, 용품, 큰 계획, 계획 부스트...| [[블리자드, 매장, 연합, 주, 법안, 블리자드 매장, 매장 단위...| [[준장, 해산, 보
고, 부대, 괴롭힘, 준장이 해산...| [[영국, 전투, 군대, 도착, 매일, 쿠웨이트, 영국 콤바...| [[브라이언트, 리드, 레이
커, 더블, 연장전, 승리, 브라이언트 리드, 리드...| [[산불, 피해자, 충동, 참조, 센터링크, 산불 피해자, 피해자...|



+-----+

```

상위 20개 행만 표시

4장 비지도 학습

```
// CountVectorizer를 사용하여 텍스트 데이터를 토큰 수의 벡터로 변환합니다.

org.apache.spark.ml.feature.{CountVectorizer, CountVectorizerModel} 가져오기

val cv = 새로운

CountVectorizer() .setInputCol("ngram_words") .setOutputCol("기능")

val cvModel = cv.fit(dataDF8)

val dataDF9 = cvModel.transform(dataDF8)

val 어휘 = cvModel.vocabulary

vocab: Array[String] = Array(police, man, new, say, plan, charge, call, Council, govt, fire, 법
정, 승리, 인터뷰, 뒤로, 죽이기, 호주, 찾기, 죽음, 충동, 얼굴, 충돌, 뉴 사우스 웨일즈 주, 보고서, 물, 도착, 오
스트레일리아 사람, qld, 갖다, 여자, 와, 공격, 시드니, 년, 변화, 살인, 히트, 건강, 교도소, 청구, 일, 어린이, 미
스..., 병원, 차, 집, 사, 도움, 열다, 오르기, 경고하다, 학교, 세계, 시장, 절단, 세트, 고발하다, 주사위, 구하다, 약,
하다, 후원, 할 수 있다, 연안, 정부, 반, 일, 그룹, 무서움, MP, 두, 이야기, 서비스, 농장주, 장관, 선거, 축적, 남쪽,
도로, 계속, 리드, 노동자, 먼저, 전국의, 테스트, 체포, 작업, 시골의, 가기, 힘, 가격, 컵, 결정적인, 우려, 녹색, 중
국, 광산, 싸움, 노동, 시도, 반환, 흥수, 거래, 북쪽, 케이스, 푸시, 오후, 멜버른, 법, 운전사, 하나, NT, 원하는, 센
터, 기록, ...
```

// IDF를 사용하여 CountVectorizer에서 생성된 기능의 크기를 조정합니다.

// 확장 기능은 일반적으로 성능을 향상시킵니다.

```
org.apache.spark.ml.feature.IDF 가져오기
```

```
발 idf = 새로운
```

```
IDF() .setInputCol("기
능") .setOutputCol("기능2")
```

```
발 idf모델 = idf.fit(dataDF9)
```

```
val dataDF10 = idfModel.transform(dataDF9)
```

```
dataDF10.select("features2").show(20,65)
+-----+
| 기능2
+-----+
|(262144,[154,1054,1140,15338,19285],[5.276861439995834,6.84427...)|(262144,
[9,122,711,727,3141,5409] [6,734,1165,1177,1324,43291,968,432,1620,2044 호
[4.070620900306447 호 | (262144, [48,121,176,208,321,376,424,2183,623,12147,248053 호], []
(25,176,208,376,764 호) 3849,12147,41079,94670,106284],[4....||(262144,
[11,1008,1743,10833,128493,136885],[4.210146(2852849),[4.210146(280849) |
[5.1202306880,16643 호 [5.120230688038215,5.54 ... | (262144,
[160,259,4208,63,1618,4208,17,1618,4208,17,16,18744 호], [5.3211036079 ... | (262144,
[7,145,234,273,321,789,6163,10334,11101 ,32988],[4.0815...||(262144,[15,223,1510,5062,5556],
[4.393970862600795,5.555011224...||(274144,[15,1]) ...||(262144,
[27,113,554,1519,3099,13499,41664,92259],[4.5216508634...||(2646144,[4,131,232,5636,6840],
[4,131,232,5636,684] .||(262144,[119,181,1288,1697,2114,49447,80829,139670],[5.1266204...|
(262144,[4,23,60,181,2637,664,2639,89 ],[3.9634887546...||(262144,[151,267,2349,3989,7631,11862],
[5.2717309555002725,5.6...||(262144,[22,513,777,12670,346287]
[502,513,752,2211,5812,7154,30415,104812],[6.143079025...||(262144,
[11,79,156,2843,8222,8709,11447],19471 ,2877,5160,19389,42259],[4.414350240692...|
+-----+
```

+-----+

상위 20개 행만 표시

// 확장된 기능은 LDA에 전달할 수 있습니다.

org.apache.spark.ml.clustering.LDA 가져오기

발 LDA = 새로운

LDA().setK(30).setMaxIter(10)

val 모델 = lda.fit(dataDF10)

4장 비지도 학습

```
val 주제 = model.describeTopics
```

```
topic.show(20,30)
```

주제	용어 인덱스	termWeight
0 [2, 7, 16, 9482, 9348, 5, 1... [1.817876125380732E-4, 1.09...		
1 [974, 2, 3, 5189, 5846, 541... [1.949552388785536E-4, 1.89...		
2 [2253, 4886, 12, 6767, 3039... [2.7922272919208327E-4, 2.4...		
3 [6218, 6313, 5762, 3387, 27... [1.6618313204146235E-4, 1.6...		
4 [0, 1, 39, 14, 13, 11, 2, 1... [1.981809243111437E-4, 1.22...		
5 [4, 7, 22, 11, 2, 3, 79, 92... [2.49620962563534E-4, 2.032...		
6 [15, 32, 319, 45, 342, 121,... [2.885684164769467E-5, 2.45...		
7 [2298, 239, 1202, 3867, 431... [3.435238376348344E-4, 3.30...		
8 [0, 4, 110, 3, 175, 38, 8, ... [1.0177738516279581E-4, 8.7...		
9 [1, 19, 10, 2, 7, 8, 5, 0, ... [2.2854683602607976E-4, 1.4...		
10 [1951, 1964, 16, 33, 1, 5, ... [1.959705576881449E-4, 1.92...		
11 [12, 89, 72, 3, 92, 63, 62,... [4.167255720848278E-5, 3.19...		
12 [4, 23, 13, 22, 73, 18, 70,... [1.1641833113477034E-4, 1.1...		
13 [12, 1, 5, 16, 185, 132, 24... [0.008769073702733892, 0.00...		
14 [9151, 13237, 3140, 14, 166... [8.201099412213086E-5, 7.85...		
15 [9, 1, 0, 11, 3, 15, 32, 52... [0.0032039727688580703, 0.0...		
16 [1, 10, 5, 56, 27, 3, 16, 1... [5.252120584885086E-5, 4.05...		
17 [12, 1437, 4119, 1230, 5303... [5.532790361864421E-4, 2.97...		
18 [12, 2459, 7836, 8853, 7162... [6.862552774818539E-4, 1.83...		
19 [21, 374, 532, 550, 72, 773... [0.0024665346250921432, 0.0...		

상위 20개 행만 표시

// 어휘의 최대 크기를 결정합니다.

```
model.vocab크기
```

```
res27: 정수 = 262144
```

// 주제어를 추출합니다. describeTopics 메서드는 다음을 반환합니다.

// CountVectorizer의 출력에서 사전 인덱스. 사용자 정의 // 사용자 정의 함수를 사용하여 단어를 index.ix에 매핑합니다.

scala.collection.mutable.WrappedArray 가져오기

org.apache.spark.sql.functions.udf 가져오기

```
val extractWords = udf( (x: WrappedArray[Int]) => { x.map(i => vocab(i)) })
```

val 주제 = 모델

```
.describe주제
.withColumn("단어", extractWords(col("termIndices"))))
```

```
topic.select("주제", "termIndices", "단어").show(20,30)
```

주제	용어 인덱스	단어
0 [2, 7, 16, 9482, 9348, 5, 1... [신규, 위원회, 찾기, 남용 ...		
1 [974, 2, 3, 5189, 5846, 541... [2016, 새로운, 말하자면, 중국해,...		
2 [2253, 4886, 12, 6767, 3039... [나단, 인터뷰 나단, ...		
3 [6218, 6313, 5762, 3387, 27... [뉴기니, 파푸아뉴기니...		
4 [0, 1, 39, 14, 13, 11, 2, 1... [경찰, 남자, 하루, 죽여, 바...		
5 [4, 7, 22, 11, 2, 3, 79, 92... [계획, 의회, 보고, 승리...		
6 [15, 32, 319, 45, 342, 121,... [호주, 연도, 인도, 사...		
7 [2298, 239, 1202, 3867, 431... [작, 투르, 드, 투르 드, 디...		
8 [0, 4, 110, 3, 175, 38, 8, ... [경찰, 계획, nt, 말, 재미...		
9 [1, 19, 10, 2, 7, 8, 5, 0, ... [남자, 얼굴, 법원, 새, 쿠...		
10 [1951, 1964, 16, 33, 1, 5, ... [빅 컨트리, 빅 컨트리 h...		
11 [12, 89, 72, 3, 92, 63, 62,... [인터뷰, 가격, 농부, ...		
12 [4, 23, 13, 22, 73, 18, 70,... [계획, 물, 뒤, 보고,...		
13 [12, 1, 5, 16, 185, 132, 24... [인터뷰, 남자, 담당, fi...		
14 [9151, 13237, 3140, 14, 166... [캠페즈, 인터뷰 테리, ...		
15 [9, 1, 0, 11, 3, 15, 32, 52... [화재, 남자, 경찰, 승리, 사...		
16 [1, 10, 5, 56, 27, 3, 16, 1... [남자, 법원, 기소, 주사위, t...		
17 [12, 1437, 4119, 1230, 5303... [인터뷰, 레드백, 666, s...		
18 [12, 2459, 7836, 8853, 7162... [인터뷰, 사이먼, 인터뷰...		
19 [21, 374, 532, 550, 72, 773... [nsw, 망명, 신청자, 망명...		

상위 20개 행만 표시

4장 비지도 학습

```
// describeTopics에서 용어 가중치를 추출합니다.

val wordWeight = udf((x : WrappedArray[Int],
y : WrappedArray[이중]) =>
{x.map(i => 어휘(i)).zip(y)}
)

val topic2 = 모델
    .describe주제
    .withColumn("단어", wordsWeight(col("termIndices"), col("termWeights")))

val topic3 = topic2
    .select("주제", "단어")
    .withColumn("단어", 폭발(col("단어")))

topic3.show(50, false)
+---+-----+
|주제|단어
+---+-----+
| 0 | [New, 1.4723785654465323E-4] | | |
| 0 | [1.242876719889358E-4] | [목요일, |
| 0 | 1.1710009304019913E-4] | [목요일, |
| 0 | 1.0958369194828903E-4] | [2, 8.119593156862581E-5] | [충전, |
| 0 | 7.321024120305904E-5] | [6.98723717903146E-5] | [Burley |
| 0 | Griffin, 6.474176573486395E-5] | [6.448801852215021E-5] |
| 0 | [Burley, 6.390953777977556E-5] | [1.9595383103126804E-4] |
| 0 | [신규, 1.7986957579978078E-4] | [살인, 1.7156446166835784E-4] |
| 0 | [[拉斯, 1.6793241095301546E-4] | [拉斯, 1.6793241095301546E-4] |
| 0 | [[拉斯베가스, 1.65222 |
| 1 | |
| 1 | |
| 1 | |
| 1 | |
| 1 | |
| 1 | |
| 1 | |
| 1 | [[2016, 1.4906599207615762E-4] | [남자, |
| 1 | 1.3653760511354596E-4] |
```

1	[전화, 1.3277357539424398E-4]
1	[트럼프, 1.250570735309821E-4]
2	[엔치, 5.213678388314454E-4]
2	[nch 팟캐스트, 4.6907569870744537E-4]
2	[팟캐스트, 4.625754070258578E-4]
2	[인터뷰, 1.2297477650126824E-4]
2	[트렌트, 9.319817855283612E-5]
2	[인터뷰 트렌트, 8.967384560094343E-5]
2	[트렌트 로빈슨, 7.256857525120274E-5]
2	[로빈슨, 6.888930961680287E-5]
2	[트렌트 로빈슨 인터뷰, 6.821800839623336E-5]
2	[미스, 6.267572268770148E-5]
3	[신규, 8.244153432249302E-5]
3	[건강, 5.269269109549137E-5]
3	[변경, 5.1481361386635024E-5]
3	[첫번째, 3.474601129571304E-5]
3	[남부, 3.335342687995096E-5]
3	[라이즈, 3.3245575277669534E-5]
3	[국가, 3.26422466284622E-5]
3	[남용, 3.25594250748893E-5]
3	[시작, 3.139959761950907E-5]
3	[장관, 3.1327427652213426E-5]
4	[경찰, 1.756612187665565E-4]
4	[남자, 1.2903801461819285E-4]
4	[피테로, 8.259870531430337E-5]
4	[죽여, 8.251557569137285E-5]
4	[고발 허가, 8.187325944352362E-5]
4	[보석을 고발, 7.609807356711693E-5]
4	[찾기, 7.219731162848223E-5]
4	[공격, 6.804063612991027E-5]
4	[일, 6.772554893634948E-5]
4	[감옥, 6.470525327671485E-5]

+-----+-----+

상위 50개 행만 표시

4장 비지도 학습

// 마지막으로 단어와 가중치를 별도의 필드로 나눕니다.

```
val topic4 = topic3
```

```
.select(col("주제"), col("단어")
    .getField("_1").as("단어"), col("단어")
    .getField("_2").as("무게"))
```

```
topic4.show(50, 거짓)
```

	주제 단어	무게	
0	new	1.4723785654465323E-4	
0	의회 목요일	1.242876719889358E-4	
0	관람석 목요	1.1710009304019913E-4	
0	일 두 충전 찾기 벌리 그리핀 청	1.0958369194828903E-4	
0	구 벌리 말 새 살인 라스 베가	8.119593156862581E-5	
0	스 라스베가스 2016 남자 전화	7.321024120305904E-5	
0	트럼프 ntch ntch 팟캐스트 팟	6.98723717903146E-5	
0	캐스트 인터뷰 트렌트	6.474176573486395E-5	
0		6.448801852215021E-5	
0		6.390953777977556E-5	
1		1.9595383103126804E-4	
1		1.7986957579978078E-4	
1		1.7156446166835784E-4	
1		1.6793241095301546E-4	
1		1.6622904053495525E-4	
1		1.627321199362179E-4	
1		1.4906599207615762E-4	
1		1.3653760511354596E-4	
1		1.3277357539424398E-4	
1		1.250570735309821E-4	
2		5.213678388314454E-4	
2		4.6907569870744537E-4	
2		4.625754070258578E-4	
2		1.2297477650126824E-4	
2		9.319817855283612E-5	

2	인터뷰 트렌트 8.967384560094343E-5
2	트렌트 로빈슨 7.256857525120274E-5
2	로빈슨 6.888930961680287E-5
2	인터뷰 트렌트 로빈슨 6.821800839623336E-5
2	미스 6.267572268770148E-5
3	신규 8.244153432249302E-5
3	건강 5.269269109549137E-5
3	변경 5.1481361386635024E-5
3	첫째 3.474601129571304E-5
3	남쪽 3.335342687995096E-5
3	상승 3.3245575277669534E-5
3	국가 3.26422466284622E-5
3	남용 3.25594250748893E-5
3	시작 3.139959761950907E-5
3	장관 3.1327427652213426E-5
4	경찰 1.756612187665565E-4
4	남자 1.2903801461819285E-4
4	피터로 8.259870531430337E-5
4	죽이기 8.251557569137285E-5
4	고발 허가 8.187325944352362E-5
4	보석을 고발하다 7.609807356711693E-5
4	찾기 7.219731162848223E-5
4	공격 6.804063612991027E-5
4	일 6.772554893634948E-5
4	감옥 6.470525327671485E-5

+-----+-----+-----+-----+

상위 50개 행만 표시

간결함을 위해 상위 50개 행만 표시하고 30개 중 4개의 주제를 표시합니다. 각 주제의 단어를 자세히 살펴보면 헤드라인을 분류하는 데 사용할 수 있는 반복되는 주제를 볼 수 있습니다.

4장 비지도 학습

Isolation Forest를 통한 이상 탐지

비정상 또는 이상값 탐지는 데이터 세트의 대부분에서 크게 벗어나 눈에 띠는 드문 관찰을 식별합니다. 몇 가지 사용 사례를 언급하자면 사기성 금융 거래를 발견하고 사이버 보안 위협을 식별하거나 예측 유지 관리를 수행하는 데 자주 사용됩니다. 이상 탐지는 기계 학습 분야에서 인기 있는 연구 분야입니다. 여러 해에 걸쳐 다양한 수준의 효율성으로 여러 변칙 탐지 기술이 개발되었습니다. 이 장에서는 Isolation Forest라고 하는 가장 효과적인 이상 탐지 기술 중 하나를 다룰 것입니다.

Isolation Forest는 Fei Tony Liu, Kai Ming Ting 및 Zhi-Hua Zhou.x가 개발한 이상 탐지를 위한 트리 기반 양상을 알고리즘입니다.

대부분의 이상 탐지 기술과 달리 Isolation Forest는 정상적인 데이터 포인트를 식별하는 대신 실제 이상값을 명시적으로 탐지하려고 합니다. Isolation Forest는 일반적으로 데이터 세트에 적은 수의 이상값이 있으므로 격리 프로세스가 발생하기 쉽다는 사실을 기반으로 작동합니다. xi 일반 데이터 포인트에서 이상값을 격리하는 것은 더 적은 조건이 필요하기 때문에 효율적입니다. 대조적으로, 일반 데이터 포인트를 분리하려면 일반적으로 더 많은 조건이 필요합니다. 그림 4-4 (b)와 같이 비정상 데이터 포인트는 단 하나의 구획으로 분리된 반면, 정상 데이터 포인트는 분리하는데 5개의 구획이 필요합니다.

데이터가 트리 구조로 표현될 때, 비정상은 일반 데이터 포인트보다 훨씬 얕은 깊이에서 루트 노드에 더 가깝습니다.

그림 4-4와 같이

(a)에서 이상치(8, 12)는 트리 깊이가 1인 반면 일반 데이터 포인트(9, 15)는 트리 깊이가 5입니다.

Isolation Forest는 이상값을 감지하는 데 사용되는 거리 임계값이 트리 깊이를 기반으로 하기 때문에 기능 크기 조정이 필요하지 않습니다. 크고 작은 데이터 세트에 적합하며 비지도 학습 기술이므로 훈련 데이터 세트가 필요하지 않습니다.xiii

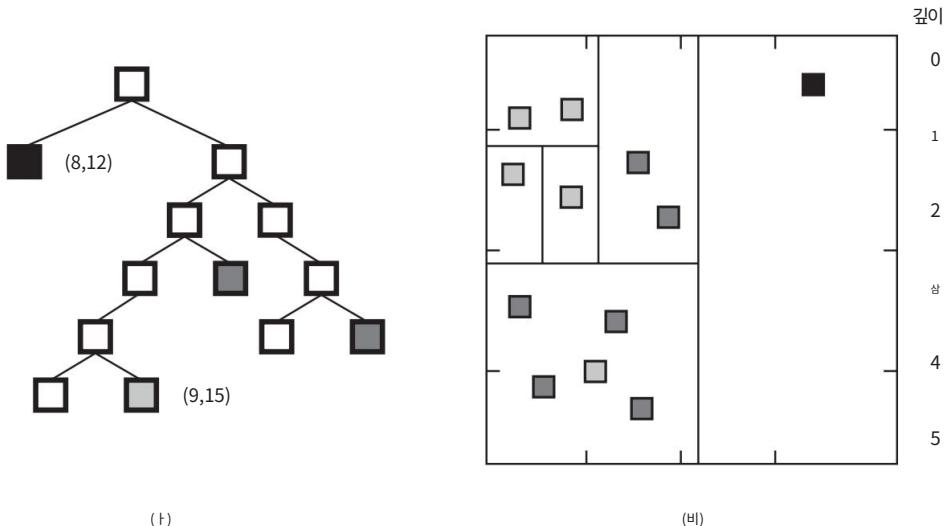


그림 4-4. Isolation Forest를 사용하여 비정상 및 정상 데이터 포인트를 분리하는데 필요한 분할 수^{xii}

다른 나무 기반 앙상블과 유사하게 Isolation Forest는 각 트리에는 전체 데이터 세트의 하위 집합이 있는 격리 트리로 알려진 의사 결정 트리가 있습니다. 이상 점수는 숲에 있는 나무의 평균 이상 점수로 계산됩니다. 이상 점수는 데이터 포인트를 분할하는데 필요한 조건의 수에서 파생됩니다. 1에 가까운 이상 점수는 이상을 의미하고 0.5 미만의 점수는 비이상 관찰을 의미합니다(그림 4-5).

4장 비지도 학습

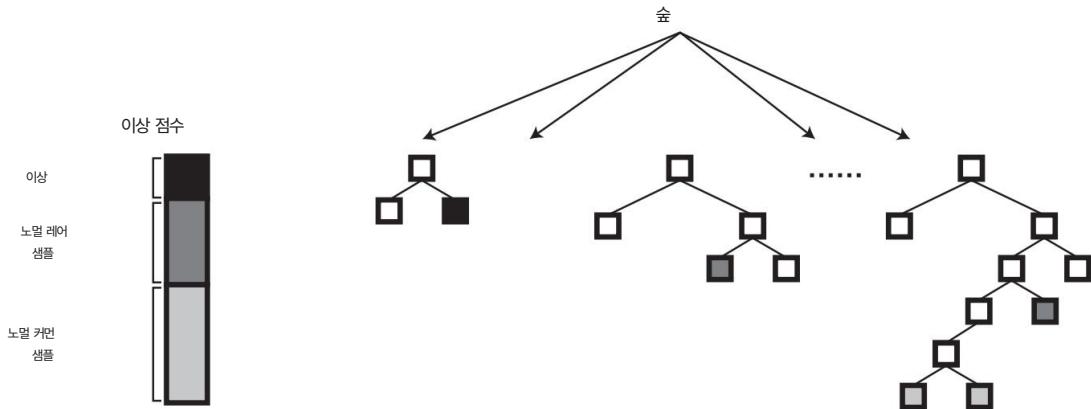


그림 4-5. Isolation Forestxiv로 이상 징후 감지

Isolation Forest는 정확도와 성능 면에서 다른 이상 탐지 방법을 능가했습니다. 그림 4-6 및 4-7은 잘 알려진 또 다른 이상값 감지 알고리즘인 단일 클래스 지원 벡터 머신에 대한 Isolation Forest의 성능 비교를 보여줍니다. -6), 두 번째 테스트에서는 두 개의 고르지 않은 클러스터에 속한 관측값에 대해 두 알고리즘을 모두 평가했습니다(그림 4-7). 두 경우 모두 Isolation Forest가 One-Class Support Vector Machine보다 더 나은 성능을 보였습니다.



그림 4-6. Isolation Forest vs. One-Class Support Vector Machine – 일반 관찰, 단일 그룹(Alejandro Correa Bahnsen의 이미지 제공)

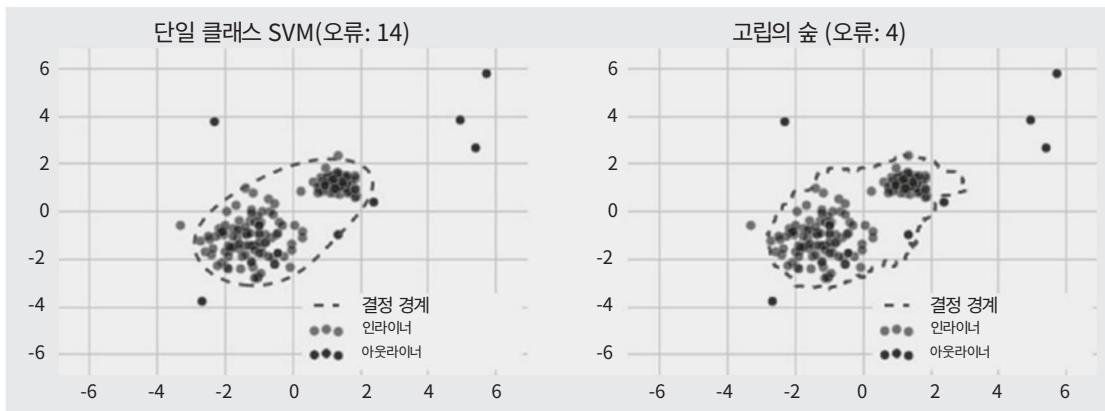


그림 4-7. Isolation Forest vs. One-Class Support Vector Machine – 고르지 않은 클러스터(Alejandro Correa Bahnsen의 이미지 제공)

Spark-iForest는 Jie Fang 및 여러 기여자의 도움으로 Fangzhou Yang이 개발한 Spark의 Isolation Forest 알고리즘 구현입니다. 외부 타사 패키지로 제공되며 표준 Apache Spark MLlib 라이브러리에 포함되어 있지 않습니다. Spark-iForest GitHub 페이지 <https://github.com/titicaca/>를 방문하면 Spark-iForest 및 최신 JAR 파일에 대한 자세한 정보를 찾을 수 있습니다.

불꽃 숲. xvi

매개변수

Spark-iForest에서 지원하는 파라미터 목록입니다. 보시다시피 일부 매개변수는 Random Forest와 같은 다른 트리 기반 양상들과 유사합니다.

- maxFeatures: 각각을 훈련하기 위해 데이터에서 끌어올 기능의 수
트리(>0). maxFeatures <= 10이면 알고리즘은 maxFeatures * totalFeatures 기능을 그립니다. maxFeatures > 10이면 알고리즘이 maxFeatures 기능을 그립니다.
- maxDepth: 트리 구성에 사용되는 높이 제한(>0). 기본값은 약 $\log_2(\text{numSamples})$ 입니다.
- numTrees: iforest 모델의 트리 수(>0).

4장 비지도 학습

- maxSamples: 각 트리를 훈련하기 위해 데이터에서 추출할 샘플 수(>0). maxSamples ≤ 1 이면 알고리즘은 maxSamples * totalSample 샘플을 그립니다. maxSamples > 1 인 경우 알고리즘은 maxSamples 샘플을 그립니다. 총 메모리는 maxSamples 정도입니다.
numTrees * 4 + maxSamples * 8바이트.
- 오염: 데이터세트에서 이상치의 비율. 값은 (0, 1)에 있어야 합니다. 예측 단계에서만 이상 점수를 예측된 레이블로 변환하는 데 사용됩니다. 성능을 향상시키기 위해 이상 점수 임계값을 얻는 방법은 approxQuantile에 의해 계산됩니다. 대규모 데이터 세트에 대한 이상 점수의 대략적인 분위수 임계값을 계산하기 위해 매개변수 approxQuantileRelativeError를 0보다 크게 설정할 수 있습니다.
- approxQuantileRelativeError: 근사값에 대한 상대 오차
분위수 계산($0 \leq \text{값} \leq 1$); 기본값은 정확한 값을 계산하는 데 0이며, 이는 큰 데이터 세트의 경우 비용이 많이 듭니다.
- 부트스트랩: true인 경우 개별 트리는 교체로 샘플링된 훈련 데이터의 무작위 하위 집합에 적합합니다. false인 경우 교체 없이 샘플링을 수행합니다.
- seed: 난수 생성기에서 사용하는 시드.
- featuresCol: 기능 열 이름, 기본 "기능".
- anomalyScoreCol: 이상 점수 열 이름, 기본값은 "anomalyScore"입니다.
- predictCol : 예측 열 이름, 기본값 "prediction".xvii

예시

UCI Machine Learning Repository.xviii에서 사용할 수 있는 Wisconsin Breast Cancer Dataset(표 4-2)을 사용하여 Spark-iForest를 사용하여 유방암(목록 4-10)의 발생을 예측합니다.

표 4-2. 위스콘신 유방암 데이터 세트

인덱스	기능	도메인
1	샘플 코드 번호	ID 번호
2	덩어리 두께	1-10
3	세포 크기의 균일성	1-10
4	세포 모양의 균일성	1-10
5	한계 접착	1-10
6	단일 상피 세포 크기	1-10
7	벌거벗은 핵	1-10
8	부드러운 염색질	1-10
9	정상 핵소체	1-10
10	유사 분열	1-10
11	수업	(양성 2개, 악성 4개)

목록 4-10. Isolation Forest를 통한 이상 탐지

```
spark-shell --jars spark-iforest-1.0-SNAPSHOT.jar
org.apache.spark.sql.types._ 가져오기

var dataSchema = StructType(배열(
StructField("id", IntegerType, true),
StructField("clump_thickness", IntegerType, true),
StructField("ucell_size", IntegerType, true),
StructField("ucell_shape", IntegerType, true),
StructField("marginal_ad", IntegerType, true),
StructField("se_cellsize", IntegerType, true),
StructField("bare_nuclei", IntegerType, true),
StructField("bland_chromatin", IntegerType, true),
StructField("normal_nucleoli", IntegerType, true),
StructField("mitosis", IntegerType, true),
StructField("클래스", IntegerType, true)
))
```

4장 비지도 학습

```
val dataDF = spark.read.option("inferSchema",
    "true") .schema(dataSchema) .csv("/files/breast-cancer-
wisconsin.csv")
```

```
dataDF.printSchema
```

//데이터 세트에는 속성 값이 누락된 16개의 행이 있습니다.
//이 연습에서는 제거할 것입니다.

```
val dataDF2 = dataDF.filter("bare_nuclei가 null이 아닙니다.")
```

```
발 시드 = 1234
```

```
val Array(trainingData, testData) = dataDF2.randomSplit(Array(0.8, 0.2), 시드)
```

org.apache.spark.ml.feature.StringIndexer 가져오기

```
val labelIndexer = 새로운 StringIndexer().setInputCol("클래스"). setOutputCol("레이블")
```

org.apache.spark.ml.feature.VectorAssembler 가져오기

```
val 어셈블러 = 새로운 VectorAssembler()
    .setInputCols(Array("clump_thickness", "ucell_size",
    "ucell_shape", "marginal_ad", "se_cellsize", "bare_nuclei", "bland_chromatin",
    "normal_nucleoli", "mitosis")) .setOutputCol("기능")
```

org.apache.spark.ml. iforest._ 가져오기

```
val iForest = 새로운
```

```
IForest() .setMaxSamples(150) .setContamination(0.30) .setBootstrap(false) .setSeed(seed) .setNumTrees
```

```
val 파이프라인 = new
```

```
Pipeline() .setStages(Array(labelIndexer, 어셈블러, iForest))
```

```
val 모델 = pipeline.fit(trainingData)
val 예측 = model.transform(testData)

predicts.select("id","features","anomalyScore","prediction").show()
```

아이디	기능	변칙점수 예측
63375 [9.0,1.0,2.0,0.6,0,... 0.6425205920636737 76389		1.0
[10.0,4.0,7.0,2.0,... 0.6475157383643779 95719 [6.0,10.0,10.0,10...		1.0
0.6413247885878359 242970 [5.0,7.0,7.0,1.0,... 0.6156526231532693		1.0
353098 [4.0,1.0,1.0,2.0,... 0.45686731187686386 369565		1.0
[4.0,1.0,1.0,1.0,... 0.45957810648090186 390840 [8.0,4.0,7.0,1.0,...		0.0
0.6387497388682214 412300 [10.0,4.0,5.0,4.0... 0.6104797020175959		0.0
466906 [1.0,1.0,1.0,1.0,... 0.41857428772927696 476903		1.0
[10.0,5.0,7.0,3.0,... 0.6152957125696049 486283 [3.0,1.0,1.0,1.0,...		1.0
0.47218763124223706 557583 [5.0,10.0,10.0,10...		0.0
0.6822227844447365 636437 [1.0,1.0,1.0,1.0,... 0.41857428772927696		1.0
654244 [1.0,1.0,1.0,1.0,... 0.4163657637214968 657753		0.0
[3.0,1.0,1.0,4.0,... 0.49314746153500594 666090 [1.0,1.0,1.0,1.0,...		1.0
0.45842258207090547 688033 [1.0,1.0,1.0,1.0,... 0.41857428772927696		0.0
690557 [5.0,1.0,1.0,1.0,... 0.4819098604217553 704097		0.0
[1.0,1.0,1.0,1.0,... 0.4163657637214968 770066 [5.0,2.0,2.0,2.0,...		0.0
0.5125093127301371		0.0
		0.0
		0.0
		0.0
		0.0
		0.0

상위 20개 행만 표시

출력에 원시 예측 필드가 있을 것으로 예상하므로 BinaryClassificationEvaluator를 사용하여 격리 포리스트 모델을 평가할 수 없습니다. Spark iForest는 rawPrediction 대신 anomalyScore 필드를 생성합니다. 대신 BinaryClassificationMetrics를 사용하여 모델을 평가합니다.

4장 비지도 학습

```
org.apache.spark.mllib.evaluation.BinaryClassificationMetrics 가져오기
```

```
val binaryMetrics = 새로운 BinaryClassificationMetrics(
predicts.select("예측", "레이블").rdd.map {
case Row(예측: Double, 레이블: Double) => (예측, 레이블)
}
)

println(s"AUC: ${binaryMetrics.areaUnderROC()}")


AUC: 0.9532866199532866
```

주성분으로 차원 축소 분석

주성분 분석(PCA)은 특징 공간의 차원을 줄이는 데 사용되는 비지도 머신 러닝 기술입니다. 기능 간의 상관 관계를 감지하고 원래 데이터 세트에서 대부분의 분산을 유지하면서 선형으로 상관되지 않은 기능의 감소된 수를 생성합니다. 이러한 보다 간결하고 선형적으로 관련이 없는 기능을 주성분이라고 합니다. 주성분은 설명된 분산의 내림차순으로 정렬됩니다. 데이터 세트에 많은 수의 기능이 있는 경우 차원 축소가 필수적입니다. 예를 들어, 유전체학 및 산업 분석 분야의 기계 학습 사용 사례에는 일반적으로 수천 또는 수백만 개의 기능이 포함됩니다. 높은 차원은 모델을 더 복잡하게 만들어 과적합 가능성을 높입니다. 특정 지점에 더 많은 기능을 추가하면 실제로 모델의 성능이 저하됩니다. 또한 고차원 데이터에 대한 교육에는 상당한 컴퓨팅 리소스가 필요합니다. 이것들은 집합 적으로 차원의 저주로 알려져 있습니다. 차원 축소 기술은 차원의 저주를 극복하는 것을 목표로 합니다.

PCA에 의해 생성된 주요 구성 요소는 해석할 수 없습니다. 이 예측이 이루어진 이유를 이해해야 하는 상황에서 거래 차단기입니다. 또한 가장 큰 규모의 기능이 다른 기능보다 더 중요하게 간주되지 않도록 PCA를 적용하기 전에 데이터 세트를 표준화하는 것이 필수적입니다.

예시

우리의 예에서는 Iris 데이터셋에서 PCA를 사용하여 4차원 특징 벡터를 2차원 주요 구성요소로 투영할 것입니다(목록 4-11 참조).

목록 4-11. PCA로 차수 줄이기

```
org.apache.spark.ml.feature.{PCA, VectorAssembler 가져오기} 가져오기
org.apache.spark.ml.feature.StringIndexer 가져오기 org.apache.spark.sql.types._  
가져오기
```

```
val irisSchema = StructType(배열(  
  StructField("sepal_length", DoubleType, true),  
  StructField("sepal_width", DoubleType, true),  
  StructField("꽃잎 길이", DoubleType, true),  
  StructField("꽃잎 너비", DoubleType, true),  
  StructField("클래스", StringType, true) ))
```

```
val dataDF = spark.read.format("csv") .option("해  
더", "거짓") .schema(irisSchema) .load("/  
files/iris.data")
```

```
dataDF.printSchema
```

뿌리

```
|-- sepal_length: double(nullable = true) |-- sepal_width:  
double(nullable = true) |-- feather_length: double(nullable  
= true) |-- feather_width: double(nullable = true) |-- 클래스:  
문자열 (nullable = true)
```

```
dataDF.show
```

4장 비지도 학습

꽃잎 길이	꽃잎 너비	꽃잎 길이	꽃잎 너비	수업
5.1	3.5	1.4	0.2	아이리스-세토사
4.9	3.0	1.4	0.2	아이리스-세토사
4.7	3.2	1.3	0.2	아이리스-세토사
4.6	3.1	1.5	0.2	아이리스-세토사
5.0	3.6	1.4	0.2	아이리스-세토사
5.4	3.9	1.7	0.4	아이리스-세토사
4.6	3.4	1.4	0.3	아이리스-세토사
5.0	3.4	1.5	0.2	아이리스-세토사
4.4	2.9	1.4	0.2	아이리스-세토사
4.9	3.1	1.5	0.1	아이리스-세토사
5.4	3.7	1.5	0.2	아이리스-세토사
4.8	3.4	1.6	0.2	아이리스-세토사
4.8	3.0	1.4	0.1	아이리스-세토사
4.3	3.0	1.1	0.1	아이리스-세토사
5.8	4.0	1.2	0.2	아이리스-세토사
5.7	4.4	1.5	0.4	아이리스-세토사
5.4	3.9	1.3	0.4	아이리스-세토사
5.1	3.5	1.4	0.3	아이리스-세토사
5.7	3.8	1.7	0.3	아이리스-세토사
5.1	3.8	1.5	0.3	아이리스-세토사

상위 20개 행만 표시

dataDF.describe().show(5,15)

요약	sepal_length	sepal_width	꽃잎 길이	꽃잎 너비
카운트	150	150	150	
평균	5.8433333333...	3.0540000000...	3.7586666666...	1.1986666666...
stddev	0.8280661279...	0.4335943113...	1.7644204199...	0.7631607417...
2.0	1.5	4.3		
4.4	6.9	2.5	7.9	

```
+-----+
|      수업|
+-----+
| 150|
| 널|
| 널|
| 아이리스세토사|
|아이리스-버지니카|
+-----+
```

```
val labelIndexer = 새로운 StringIndexer()
    .setInputCol("클래스")
    .setOutputCol("레이블")
```

```
val dataDF2 = labelIndexer.fit(dataDF).transform(dataDF)
```

```
dataDF2.printSchema
```

뿌리

```
|-- sepal_length: 이중(nullable = true)
|-- sepal_width: 이중(nullable = true)
|-- 꽃잎 길이: 이중(nullable = true)
|-- 꽃잎 너비: 이중(nullable = true)
|-- 클래스: 문자열(nullable = true)
|-- 레이블: 이중(nullable = false)
```

```
dataDF2.show
```

꽃잎 길이	꽃잎 너비	꽃잎 길이	꽃잎 너비	클래스라벨
5.1	3.5	1.4	0.2	아이리스-세토사 0.0
4.9	3.0	1.4	0.2	아이리스-세토사 0.0
4.7	3.2	1.3	0.2	아이리스-세토사 0.0
4.6	3.1	1.5	0.2	아이리스-세토사 0.0
5.0	3.6	1.4	0.2	아이리스-세토사 0.0
5.4	3.9	1.7	0.4	아이리스-세토사 0.0
4.6	3.4	1.4	0.3	아이리스-세토사 0.0

4장 비지도 학습

5.0	3.4	1.5	0.2 아이리스-세토사 0.0
4.4	2.9	1.4	0.2 아이리스-세토사 0.0
4.9	3.1	1.5	0.1 아이리스-세토사 0.0
5.4	3.7	1.5	0.2 아이리스-세토사 0.0
4.8	3.4	1.6	0.2 아이리스-세토사 0.0
4.8	3.0	1.4	0.1 아이리스-세토사 0.0
4.3	3.0	1.1	0.1 아이리스-세토사 0.0
5.8	4.0	1.2	0.2 아이리스-세토사 0.0
5.7	4.4	1.5	0.4 아이리스-세토사 0.0
5.4	3.9	1.3	0.4 아이리스-세토사 0.0
5.1	3.5	1.4	0.3 아이리스-세토사 0.0
5.7	3.8	1.7	0.3 아이리스-세토사 0.0
5.1	3.8	1.5	0.3 아이리스-세토사 0.0

상위 20개 행만 표시

org.apache.spark.ml.feature.VectorAssembler 가져오기

val 기능 = Array("sepal_length", "sepal_width", "petal_length", "petal_width")

val 어셈블러 = 새로운 VectorAssembler()

.setInputCols(기능)

.setOutputCol("기능")

val dataDF3 = assembler.transform(dataDF2)

dataDF3.printSchema

뿌리

|-- sepal_length: 이중(nullable = true)

|-- sepal_width: 이중(nullable = true)

|-- 꽃잎 길이: 이중(nullable = true)

|-- 꽃잎 너비: 이중(nullable = true)

|-- 클래스: 문자열(nullable = true)

|-- 레이블: 이중(nullable = false)

|-- 기능: 벡터(nullable = true)

dataDF3.show

꽃잎 길이	꽃잎 너비	꽃잎 길이	꽃잎 너비	클래스 라벨
5.1	3.5	1.4	0.2	아이리스-세토사 0.0
4.9	3.0	1.4	0.2	아이리스-세토사 0.0
4.7	3.2	1.3	0.2	아이리스-세토사 0.0
4.6	3.1	1.5	0.2	아이리스-세토사 0.0
5.0	3.6	1.4	0.2	아이리스-세토사 0.0
5.4	3.9	1.7	0.4	아이리스-세토사 0.0
4.6	3.4	1.4	0.3	아이리스-세토사 0.0
5.0	3.4	1.5	0.2	아이리스-세토사 0.0
4.4	2.9	1.4	0.2	아이리스-세토사 0.0
4.9	3.1	1.5	0.1	아이리스-세토사 0.0
5.4	3.7	1.5	0.2	아이리스-세토사 0.0
4.8	3.4	1.6	0.2	아이리스-세토사 0.0
4.8	3.0	1.4	0.1	아이리스-세토사 0.0
4.3	3.0	1.1	0.1	아이리스-세토사 0.0
5.8	4.0	1.2	0.2	아이리스-세토사 0.0
5.7	4.4	1.5	0.4	아이리스-세토사 0.0
5.4	3.9	1.3	0.4	아이리스-세토사 0.0
5.1	3.5	1.4	0.3	아이리스-세토사 0.0
5.7	3.8	1.7	0.3	아이리스-세토사 0.0
5.1	3.8	1.5	0.3	아이리스-세토사 0.0
<hr/>				
<hr/>				
	기능			
<hr/>				
[5.1,3.5,1.4,0.2]				
[4.9,3.0,1.4,0.2]				
[4.7,3.2,1.3,0.2]				
[4.6,3.1,1.5,0.2]				
[5.0,3.6,1.4,0.2]				
[5.4,3.9,1.7,0.4]				
[4.6,3.4,1.4,0.3]				

4장 비지도 학습

```

|[5.0,3.4,1.5,0.2]| |
|[4.4,2.9,1.4,0.2]| |
|[4.9,3.1,1.5,0.1]| |
|[5.4,3.7,1.5,0.2]| |
|[4.8,3.4,1.6,0.2]| |
|[4.8,3.0,1.4,0.1]| |
|[4.3,3.0,1.1,0.1]| |
|[5.8,4.0,1.2,0.2]| |
|[5.7,4.4,1.5,0.4]| |
|[5.4,3.9,1.3,0.4]| |
|[5.1,3.5,1.4,0.3]| |
|[5.7,3.8,1.7,0.3]| |
|[5.1,3.8,1.5,0.3]|
+-----+
// 4개의 속성(sepal_length, sepal_width, // 꽃잎 길이 및 꽃잎 너비)이 모두 동일한 스케일을 가지고 // 동일한 수
량을 측정하더라도 StandardScaler를 사용하여 표준화합니다. 앞서 논의한 바와 같이, // 표준화는 모범 사례로 간주되며 //
PCA와 같은 많은 알고리즘이 최적으로 실행되기 위한 요구 사항입니다.

```

org.apache.spark.ml.feature.StandardScaler 가져오기

발 스케일러 = 새로운 StandardScaler() .setInputCol("기

능") .setOutputCol("scaledFeatures") .setWithStd(true) .setWithMean(false)

val dataDF4 = scaler.fit(dataDF3).transform(dataDF3)

dataDF4.printSchema

뿌리

```

|-- sepal_length: double(nullable = true) |-- sepal_width:
double(nullable = true) |-- feather_length: double(nullable
= true) |-- feather_width: double(nullable = true) |-- 클래스:
문자열 (nullable = true)

```

```
|-- 레이블: 이중(nullable = false) |-- 기능: 벡터(nullable = true)
= true) |-- scaledFeatures: 벡터(nullable = true)
```

// 두 개의 주요 구성 요소를 생성합니다.

```
val pca = 새로운
```

```
PCA() .setInputCol("scaledFeatures") .setOutputCol("pcaFeatures") .setK(2) .fit(dataDF4)
```

```
val dataDF5 = pca.transform(dataDF4)
```

```
dataDF5.printSchema
```

뿌리

```
|-- sepal_length: double(nullable = true) |-- sepal_width:
double(nullable = true) |-- feather_length: double(nullable
= true) |-- feather_width: double(nullable = true) |-- 클래스:
문자열 (nullable = true) |-- 레이블: double(nullable = false) |--|
기능: 벡터(nullable = true) |-- scaledFeatures: 벡터(nullable =
true) |-- pcaFeatures: 벡터(nullable = true)
```

```
dataDF5.select("scaledFeatures","pcaFeatures").show(false)
```

```
+-----+  
|크기 조정된 기능  
+-----+  
|[6.158928408838787,8.072061621390857,0.7934616853039358,0.26206798787142]||  
|[5.9174018045706,6.9189099611921625,0.7934616853039358,0.26206798787142]||  
|[5.675875200302412,7.38017062527164,0.7367858506393691,0.26206798787142]||  
|[5.555111898168318,7.149540293231902,0.8501375199685027,0.26206798787142]||  
|[6.038165106704694,8.302691953430596,0.7934616853039358,0.26206798787142]||  
|[6.52121831524107,8.99458294954981,0.9634891892976364,0.52413597574284]||  
|[5.555111898168318,7.841431289351117,0.7934616853039358,0.39310198180713]|
```

4장 비지도 학습

[[6.038165106704694,7.841431289351117,0.8501375199685027,0.26206798787142]] |
 [5.313585293900131,6.688279629152423,0.7934616853039358,0.26206798787142]] |
 [5.9174018045706,7.149540293231902,0.8501375199685027,0.13103399393571]] |
 [6.52121831524107,8.533322285470334,0.8501375199685027,0.26206798787142]] |
 [5.7966385024365055,7.841431289351117,0.9068133546330697,0.262067987871]] |
 [5.7966385024365055,6.9189099611921625,0.7934616853039358,0.131033993935]] |
 [5.192821991766037,6.9189099611921625,0.6234341813102354,0.1310339939351]] |
 [7.004271523777445,9.22521328158955,0.6801100159748021,0.26206798787142]] |
 [6.883508221643351,10.147734609748506,0.8501375199685027,0.524135975742]] |
 [6.52121831524107,8.99458294954981,0.7367858506393691,0.52413597574284]] |
 [6.158928408838787,8.072061621390857,0.7934616853039358,0.39310198180713]] |
 [6.883508221643351,8.763952617510071,0.9634891892976364,0.39310198180713]] |
 [6.158928408838787,8.763952617510071,0.8501375199685027,0.39310198180713]] |

+-----+

+-----+ |

|pca특징
 +-----+ |
 [[-1.7008636408214346,-9.798112476165109]] |
 [-1.8783851549940478,-8.640880678324866]] |
 [-1.597800192305247,-8.976683127367169]] |
 [-1.6613406138855684,-8.720650458966217]] |
 [-1.5770426874367196,-9.96661148272853]] |
 [-1.8942207975522354,-10.80757533867312]] |
 [-1.5202989381570455,-9.368410789070643]] |
 [-1.7314610064823877,-9.540884243679617]] |
 [-1.6237061774493644,-8.202607301741613]] |
 [-1.7764763044699745,-8.846965954487347]] |
 [-1.8015813990792064,-10.361118028393015]] |
 [-1.6382374187586244,-9.452155017757546]] |
 [-1.741187558292187,-8.587346593832775]] |
 [-1.3269417814262463,-8.358947926562632]] |
 [-1.7728726239179156,-11.177765120852797]] |
 [-1.7138964933624494,-12.00737840334759]] |

```
[[-1.7624485738747564,-10.80279308233496] |
[-1.7624485738747564,-10.80279308233496] |
[-1.7624485738747564,-10.80279308233496] |
[-1.6257080769316516,-10.44826393443861] |
+-----+
```

앞서 논의한 바와 같이 Iris 데이터 세트에는 세 가지 종류의 꽃(Iris Setosa, Iris Versicolor 및 Iris Virginica)이 있습니다. 네 가지 속성(꽃받침 길이, 꽃받침 너비, 꽃잎 길이, 꽃잎 너비)이 있습니다. 두 개의 주성분에 대한 샘플을 플롯해 보겠습니다. 그림 4-8에서 볼 수 있듯이 Iris Setosa는 다른 두 클래스와 잘 분리되어 있는 반면 Iris Versicolor와 Iris Virginica는 약간 겹칩니다.

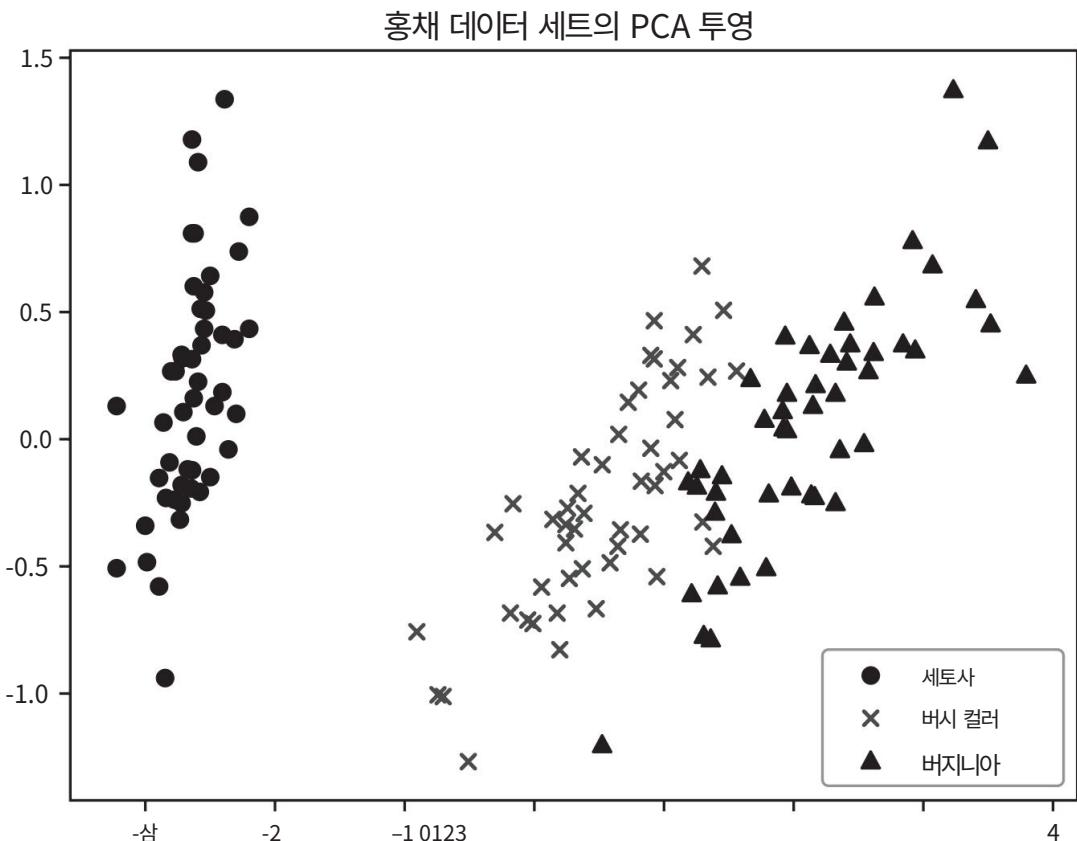


그림 4-8. 홍채 데이터 세트의 PCA 투영

4장 비지도 학습

ExplainVariance 메서드는 각 주성분이 설명하는 분산 비율을 포함하는 벡터를 반환합니다. 우리의 목표는 새로운 주성분에서 가능한 한 많은 분산을 유지하는 것입니다.

pca.explainedVariance

```
res5: org.apache.spark.ml.linalg.DenseVector = [0.7277045209380264, 0.230305  
23267679512]
```

방법의 결과를 기준으로 첫 번째 주성분은 72.77%를 설명합니다.

분산의 23.03%는 두 번째 주성분으로 설명됩니다. 두 가지 주성분이 누적하여 분산의 95.8%를 설명합니다. 보시다시피 치수를 줄였을 때 일부 정보가 손실되었습니다.

이는 우수한 모델 정확도를 유지하면서 상당한 훈련 성능 향상이 있는 경우 일반적으로 허용 가능한 절충안입니다.

요약

우리는 몇 가지 비지도 학습 기술에 대해 논의하고 실제 비즈니스 사용 사례에 적용하는 방법을 배웠습니다. 비지도 학습(Unsupervised learning)은 빅 데이터의 출현으로 최근 몇 년 동안 다시 인기를 얻었습니다. 클러스터링, 이상 감지 및 주성분 분석과 같은 기술은 모바일 및 IoT 장치, 센서, 소셜 미디어 등에서 생성되는 비정형 데이터의 홍수를 이해하는 데 도움이 됩니다. 머신 러닝 무기고에 포함할 수 있는 강력한 도구입니다.

참고문헌

나. 커트 보네거트; "18. 지구상에서 가장 가치 있는 상품,"

1998, 고양이의 요람: 소설

ii. Chris Piech, Andrew Ng, 마이클 조던; "K 평균", stanford.edu, 2013, <https://stanford.edu/~cpiech/cs221/handouts/kmeans.html>

HTML

iii. 제이슨 브라운리; "머신 러닝에서 데이터를 원-핫 인코딩하는 이유는 무엇입니까?", machinelearningmastery.com, 2017, <https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/>

- iv. 데이비드 탈비; "Apache Spark용 자연어 처리 라이브러리 소개",
Databricks, 2017, <https://databricks.com/blog/2017/10/19/introducing-natural-language-processing-library-apache-spark.html>
- v. Christopher D. Manning et al.; "스탠포드 CoreNLP 내추럴 언어 처리 도구 키트," 스탠포드 대학, <https://nlp.stanford.edu/pubs/StanfordCoreNlp2014.pdf>
- vi. 존 스노우 연구소; "빠른 시작", John Snow Labs, 2019, <https://nlp.johnsnowlabs.com/docs/en/quickstart>
- vii. 시밤 반살; "자연어 처리를 이해하고 구현하기 위한 궁극적인 가이드" Analytics Vidhya, 2017, [www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/](https://analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/)
- viii. 세바스찬 라슈카; "나이브 베이즈와 텍스트 분류 – 서론과 이론", sebastianraschka.com, 2014, https://sebastianraschka.com/Articles/2014_naive_bayes_1.html
- ix. 지그문트 자와즈키; "LDA 모델에서 주제 단어 가져오기", zstat.pl, 2018, www.zstat.pl/2018/02/07/scala-spark-get-topics-words-from-lda-model/
- 엑스. Fei Tony Liu, Kai Ming Ting, Zhia-Hua Zhou; "고립의 숲", acm.org, 2008, <https://dl.acm.org/citation.cfm?id=1511387>
- xi. 알레한드로 코레아 반센; "격리 포리스트를 사용한 이상 탐지의 이점" easysol.net, 2016, <https://blog.easysol.net/using-isolation-forests-anamoly-detection/>
- xii. 이순 등 알.; "비정상 사용자 행동 탐지 확장된 격리 포리스트 알고리즘: 기업 사례 연구," arxiv.org, 2016, <https://arxiv.org/pdf/1609.06676.pdf>
- xiii. 이순 등 알.; "비정상 사용자 행동 탐지 확장된 격리 포리스트 알고리즘: 기업 사례 연구," arxiv.org, 2016, <https://arxiv.org/pdf/1609.06676.pdf>

4장 비지도 학습

- xiv. 장지민; "Isolation Forest를 통한 대표적인 하위 집합 선택 및 이상값 감지",
github.com, 2016, <https://github.com/zmzhang/IOS>
- xv. 알레한드로 코레아 반센; "격리 포리스트를 사용한 이상 탐지의 이점" easysol.net,
2016, <https://blog.easysol.net/using-isolation-forests-anamoly-detection/>
16. Fangzhou Yang 및 기고자; "spark-iforest", github.com, 2018, <https://github.com/titicaca/spark-iforest>
- xvii. Fangzhou Yang 및 기고자; "spark-iforest", github.com, 2018, <https://github.com/titicaca/spark-iforest>
- xviii. William H. Wolberg 박사 등; "유방암 위스콘신(진단)
데이터 세트," archive.ics.uci.edu, 1995, [http://archive.ics.uci.edu/ml/데이터 세트/유방+암+위스콘신+\(진단\)](http://archive.ics.uci.edu/ml/데이터 세트/유방+암+위스콘신+(진단))

5장

권장 사항

인간은 무엇을 욕망해야 할지 모르는 피조물이며, 결정을 내리기 위해 남에게 의지한다. 우리는 다른 사람들의 욕망을 모방하기 때문에 다른 사람들이 원하는 것을 욕망합니다.

—르네 지라르디

개인화된 추천을 제공하는 것은 기계 학습의 가장 인기 있는 응용 프로그램 중 하나입니다. Amazon, Alibaba, Walmart 및 Target과 같은 거의 모든 주요 소매업체는 고객 행동을 기반으로 일종의 개인화된 추천을 제공합니다. Netflix, Hulu 및 Spotify와 같은 스트리밍 서비스는 사용자의 취향과 선호도에 따라 영화 또는 음악 추천을 제공합니다.

권장 사항은 고객 만족도와 참여를 개선하는 데 중요하며, 이는 궁극적으로 매출과 수익을 증가시킵니다. 추천의 중요성을 강조하기 위해 Amazon 고객의 44%는 Amazon에서 본 제품 추천을 통해 구매합니다. ii McKinsey 보고서에 따르면 고객 판매의 35% 가 Amazon의 추천에서 직접 발생합니다. 같은 연구에 따르면 Netflix에서 시청자가 시청하는 콘텐츠의 75%가 개인화된 추천에서 나온다고 합니다. iii Netflix의 최고 제품 책임자(CPO)는 인터뷰에서 Netflix의 개인화된 영화 및 TV 프로그램 추천이 회사에 연간 10억 달러의 가치가 있다고 밝혔습니다. iv Alibaba의 추천 엔진이 도움이 되었습니다. 기록적인 판매를 주도하여 2013년 매출이 2,480억 달러(Amazon과 eBay를 합친 것보다 많음)로 세계 최대 전자 상거래 회사 중 하나이자 소매 유통의 거인이 되었습니다. v

권장 사항은 소매업체 및 스트리밍 서비스에만 국한되지 않습니다. 은행은 추천 엔진을 표적 마케팅 도구로 사용하여 온라인 뱅킹 고객에게 인구 통계 및 심리 프로필을 기반으로 주택 또는 학자금 대출과 같은 금융 상품 및 서비스를 제공합니다. 광고 및 마케팅 대행사는 추천 엔진을 사용하여 고도로 타겟팅된 온라인 광고를 표시합니다.

5장 권장 사항

추천 엔진의 유형

추천 엔진에는 여러 유형이 있습니다.^{vi} 우리는 가장 널리 사용되는 협업 필터링, 콘텐츠 기반 필터링 및 연관 규칙에 대해 논의할 것입니다.

교대 최소 제곱을 사용한 협업 필터링

협업 필터링은 웹에서 개인화된 권장 사항을 제공하는 데 자주 사용됩니다. 협업 필터링을 활용하는 회사에는 Netflix, Amazon, Alibaba, Spotify 및 Apple이 있습니다. 협업 필터링은 다른 사람들(협업)의 선호도나 취향을 기반으로 추천(필터링)을 제공합니다. 같은 취향을 가진 사람들이 미래에 같은 관심사를 가질 가능성이 더 높다는 아이디어에 기반을 두고 있습니다. 예를 들어, Laura는 Titanic, Apollo 13 및 The Towering Inferno를 좋아합니다.

Tom은 Apollo 13 과 The Towering Inferno를 좋아합니다. Anne이 Apollo 13 을 좋아하고 우리의 계산에 따르면 Apollo 13 을 좋아하는 사람은 Towering Inferno 도 좋아해야 합니다 . The Towering Inferno 는 Anne에게 잠재적인 추천이 될 수 있습니다. 제품은 영화, 노래, 비디오 또는 책과 같은 항목일 수 있습니다.

	영화 1	영화 2	영화 3
로라		4	4
앤	2	5	?
톰		4	5

그림 5-1. ALS 등급 매트릭스^{vii}

Spark MLlib에는 ALS(Alternating Least Squares)라는 협업 필터링을 위한 인기 있는 알고리즘이 포함되어 있습니다. ALS는 등급 매트릭스(그림 5-1)를 사용자와 제품 요인^{viii} (그림 5-2)의 곱으로 모델링 합니다. ALS는 최소 제곱 계산을 사용하여 추정 오류를 최소화 하고 프로세스가 수렴될 때까지 고객 요인 수정과 제품 요인 사이를 번갈아 반복하며 그 반대의 경우도 마찬가지입니다. Spark MLlib는 Spark의 분산 처리를 활용하는 차단된 버전의 ALS를 구현합니다.

5장 권장 사항

두 가지 요소 집합("사용자" 및 "제품"이라고 함)을 블록으로 그룹화하여 기능을 제공하고 각 반복에서 각 제품 블록에 각 사용자 벡터의 사본 하나만 전송하여 통신을 줄입니다. 해당 사용자의 특징 vector.x

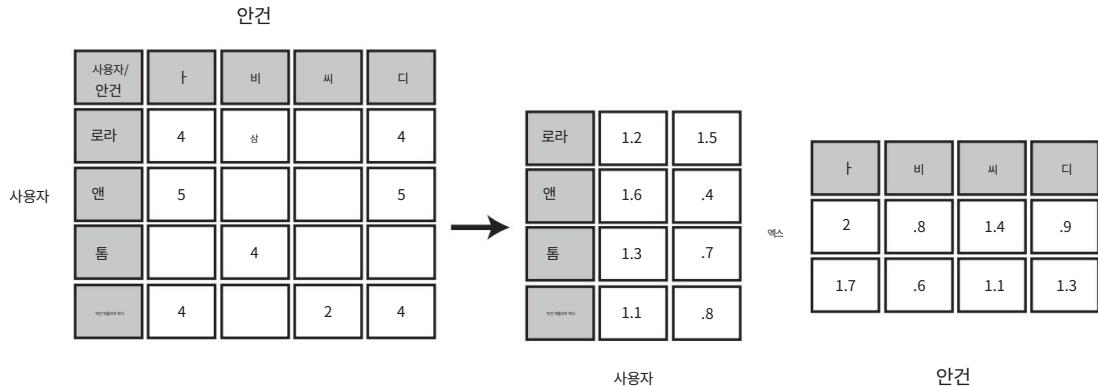


그림 5-2. ALS가 권장 사항을 계산하는 방법

Spark MLlib의 ALS 구현은 명시적 및 암시적 평가를 모두 지원합니다.

명시적 평가(기본값)는 제품에 대한 사용자의 평가가 점수(예: 1-5개의 좋아요)가 되어야 하는 반면 암시적 평가는 사용자가 제품과 상호 작용할 것이라는 확신(예: 클릭 수 또는 페이지 조회수 또는 동영상이 스트리밍된 횟수). 모든 회사가 제품에 대한 명시적 등급을 수집하는 것은 아니기 때문에 암시적 등급은 실제 시나리오에서 더 일반적입니다. 그러나 웹 로그, 시청 습관 또는 판매 거래와 같은 기업 데이터에서 암시적 등급을 추출할 수 있습니다.

Spark MLlib의 ALS 구현은 항목 및 사용자 ID에 정수를 사용합니다. 즉, 항목 및 사용자 ID는 정수 값 범위 내에 있어야 하며 최대값은 2,147,483,647이어야 합니다.

참고 Alternating Least Squares(ALS)는 Yehuda Koren 및 Robert M. Bell.xi
의 "공동으로 파생된 이웃 보간 가중치를 사용한 확장 가능한 협업 필터링" 문서에 설명되어 있습니다.

5장 권장 사항

매개변수

Spark MLlib의 ALS 구현은 다음 매개변수를 지원합니다.xii:

- 알파: 선호도 관찰에서 기준선 신뢰도를 지시하는 ALS의 암시적 피드백 버전에 적용 가능
- numBlocks: 병렬 처리에 사용됩니다. 블록의 수
항목과 사용자는 다음으로 분할됩니다.
- non-negative: non-negative 제약 조건을 사용할지 여부를 나타냅니다 .
최소제곱
- implicitPrefs: 명시적 피드백을 사용할지 암시적 피드백을 사용할지 나타냅니다 .
피드백
- k: 모델의 잠재 요인 수를 나타냅니다.
- regParam: 정규화 매개변수
- maxIter: 실행할 최대 반복 횟수를 나타냅니다.

예시

MovieLens 데이터 세트를 사용하여 장난감 영화 추천 시스템을 구축합니다. 데이터세트는 <https://grouplens.org/datasets/movielens/> 에서 다운로드할 수 있습니다 . 데이터 세트에는 여러 파일이 포함되어 있지만 주로 Ratings.csv에 관심이 있습니다. 목록 5-1 에 표시된 파일의 각 행에는 영화에 대한 사용자의 명시적 등급(등급 1-5)이 포함되어 있습니다.

목록 5-1. ALS가 있는 영화 추천

```
val dataDF = spark.read.option("헤더", "참")
    .option("inferSchema", "true")
    .csv("등급.csv")
```

dataDF.printSchema

뿌리

```
|-- userId: 정수(nullable = true)
|-- movieId: 정수(nullable = true)
|-- 등급: 이중(nullable = true)
|-- 타임스탬프: 정수(nullable = true)
```

```
dataDF.show
```

userId	movieId	등급	타임스탬프
1	1	4.0	964982703
1	3	4.0	964981247
1	6	4.0	964982224
1	47	5.0	964983815
1	50	5.0	964982931
1	70	3.0	964982400
1	101	5.0	964980868
1	110	4.0	964982176
1	151	5.0	964984041
1	157	5.0	964984100
1	163	5.0	964983650
1	216	5.0	964981208
1	223	3.0	964980985
1	231	5.0	964981179
1	235	4.0	964980908
1	260	5.0	964981680
1	296	3.0	964982967
1	316	3.0	964982310
1	333	5.0	964981179
1	349	4.0	964982563

상위 20개 행만 표시

```
val Array(trainingData, testData) = dataDF.randomSplit(Array(0.7, 0.3))
```

org.apache.spark.ml.recommendation.ALS 가져오기

```
val als = 새로운 ALS()
    .setMaxIter(15)
    .setRank(10)
    .setSeed(1234)
    .setRatingCol("등급")
    .setUserCol("사용자 ID")
    .setItemCol("동영상 ID")
```

5장 권장 사항

```
val 모델 = als.fit(trainingData)
val 예측 = model.transform(testData)
```

예측.printSchema

뿌리

```
-- userId: 정수(nullable = true)
-- movieId: 정수(nullable = true)
-- 등급: 이중(nullable = true)
-- 타임스탬프: 정수(nullable = true)
-- 예측: float(nullable = false)
```

예측.쇼

userId	movieId	등급	타임스탬프	예측
133	471	4.0	843491793	2.5253267
602	182	4.0	840876085	3.2802277
500	471	4.5	1054779644	3.6534667
387	610	1.0	1005528017	3.5033386
136	471	3.0	1139047519	2.6689813
312	287	4.0	1479544381	3.006948
32	469	4.0	832450058	3.1404104
608	471	4.0	1043175564	3.109232
373	191	4.5	1110231536	2.9776838
44	609	3.0	856737165	3.5183017
608	463	5.0	965425364	2.8298397
47	479	1.5	1117161794	3.007364
		5.0	846830388	3.9275675
		5.0	829760898	난
		2.0	869252237	2.4776468
		3.0	847221080	1.9167987
		0.5	1117506344	2.220617
		3.5	1145460096	3.0794377
		4.0	1496205519	2.4831696
		4.0	1039362157	3.5400867

org.apache.spark.ml.evaluation.RegressionEvaluator 가져오기

```
val 평가자 = 새로운 RegressionEvaluator()
    .setPredictionCol("예측")
    .setLabelCol("등급")
    .setMetricName("rmse")

val rmse = evaluator.evaluate(예측)
rmse: 더블 = NaN
```

평가자가 예측 DataFrame의 NaN 값을 좋아하지 않는 것 같습니다.

지금은 NaN 값이 있는 행을 제거하여 문제를 해결해 보겠습니다. 나중에 이 문제를 처리하기 위해 coldStartStrategy 매개변수를 사용하는 방법에 대해 논의합니다.

발 예측2 = 예측.na.drop

예측2.쇼

userId	movieId	등급	타임스탬프	예측
133	471	4.0	843491793	2.5253267
602	182	4.0	840876085	3.2802277
500		4.5	1054779644	3.6534667
387	610	1.0	1005528017	3.5033386
136		3.0	1139047519	2.6689813
312	287	4.0	1479544381	3.006948
32	469	4.0	832450058	3.1404104
608		4.0	1043175564	3.109232
373	44	4.5	1110231536	2.9776838
609		3.0	856737165	3.5183017
608	463	5.0	965425364	2.8298397
47		1.5	1117161794	3.007364
		5.0	846830388	3.9275675
		2.0	869252237	2.4776468
		3.0	847221080	1.9167987
		0.5	1117506344	2.220617
		3.5	1145460096	3.0794377
		4.0	1496205519	2.4831696

5장 권장 사항

```
| 479| 554| 1088| 4.0|1039362157| 3.5400867|
    | 1088| 5.0| 944900489| 3.3577442|
+-----+-----+-----+-----+
```

상위 20개 행만 표시

```
val 평가자 = 새로운 RegressionEvaluator()
    .setPredictionCol("예측")
    .setLabelCol("등급")
    .setMetricName("rmse")

val rmse = evaluator.evaluate(예측2)
rmse: 더블 = 0.9006479893684061
```

참고 ALS를 사용할 때 모델이 훈련되었을 때 존재하지 않았던 테스트 데이터 세트의 사용자 및/또는 항목을 접하게 되는 경우가 있습니다. 신규 사용자 또는 항목에는 평가가 없고 모델이 학습되지 않았을 수 있습니다.

이것은 콜드 스타트 문제로 알려져 있습니다. 데이터가 평가 데이터 세트와 훈련 데이터 세트 간에 무작위로 분할되는 경우에도 이 문제가 발생할 수 있습니다. 사용자 및/또는 항목이 모델에 없는 경우 예측이 NaN으로 설정됩니다. 이것이 우리가 모델을 평가할 때 더 일찍 NaN 결과를 만난 이유입니다. 이 문제를 해결하기 위해 Spark는 NaN 값을 포함하는 예측 DataFrame의 모든 행을 삭제하도록 설정할 수 있는 coldStartStrategy 매개변수를 제공합니다.^{xiii}

몇 가지 권장 사항을 생성해 보겠습니다.

모든 사용자에게 상위 3개 영화를 추천합니다.

```
model.recommendForAllUsers(3).show(거짓)
```

```
+-----+
|userId|권장사항
+-----+
|471|[[7008, 4.8596725], [7767, 4.8047066], [26810, 4.7513227]]|
|463|[[33649, 5.0881286], [3347, 4.7693057], [68945, 4.691733]]|
|496|[[6380, 4.946864], [26171, 4.8910613], [7767, 4.868356]]|
|148|[[183897, 4.972257], [6732, 4.561547], [33649, 4.5440807]]|
|540|[[26133, 5.19643], [68945, 5.1259947], [3379, 5.1259947]]|
|392|[[3030, 6.040107], [4794, 5.6566052], [55363, 5.4429026]]|
```

```
|243 [[1223, 6.5019746], [68945, 6.353135], [3379, 6.353135]] |
|31 [[4256, 5.3734074], [49347, 5.365612], [7071, 5.3175936]] |
|516 [[4429, 4.8486495], [48322, 4.8443394], [28, 4.8082485]] |
|580 [[86347, 5.20571], [4256, 5.0522637], [72171, 5.037114]] |
|251 [[33649, 5.6993585], [68945, 5.613014], [3379, 5.613014]] |
|451 [[68945, 5.392536], [3379, 5.392536], [905, 5.336588]] |
|85 [[25771, 5.2532864], [8477, 5.186757], [99764, 5.1611686]] |
|137 [[7008, 4.8952146], [26131, 4.8543305], [3200, 4.6918836]] |
|65 [[33649, 4.695069], [3347, 4.5379376], [7071, 4.535537]] |
|458 [[3404, 5.7415047], [7018, 5.390625], [42730, 5.343014]] |
|481 [[232, 4.393473], [3473, 4.3804317], [26133, 4.357505]] |
|53 [[3200, 6.5110188], [33649, 6.4942613], [3347, 6.452143]] |
|255 [[86377, 5.9217377], [5047, 5.184309], [6625, 4.962062]] |
|588 [[26133, 4.7600465], [6666, 4.65716], [39444, 4.613207]] |
+-----+

```

상위 20개 행만 표시

모든 영화에 대해 상위 3명의 사용자를 추천합니다.

```
model.recommendForAllItems(3).show(거짓)
```

```
+-----+-----+-----+
|movield|추천 | |
+-----+-----+-----+
|1580 [[53, 4.939177], [543, 4.8362885], [452, 4.5791063]] |
|4900 [[147, 3.0081954], [375, 2.9420073], [377, 2.6285374]] |
|5300 [[53, 4.29147], [171, 4.129584], [375, 4.1011653]] |
|6620 [[392, 5.0614614], [191, 4.820595], [547, 4.7811346]] |
|7340 [[413, 3.2256641], [578, 3.1126869], [90, 3.0790782]] |
|32460 [[53, 5.642673], [12, 5.5260286], [371, 5.2030106]] |
|54190 [[53, 5.544555], [243, 5.486003], [544, 5.243029]] |
|471 [[51, 5.073474], [53, 4.8641024], [337, 4.656805]] |
|1591 [[112, 4.250576], [335, 4.147236], [207, 4.05843]] |
|140541 [[393, 4.4335465], [536, 4.1968756], [388, 4.0388694]] |
|1342 [[375, 4.3189483], [313, 3.663758], [53, 3.5866988]] |
|2122 [[375, 4.3286233], [147, 4.3245177], [112, 3.8350344]] |
|2142 [[51, 3.9718416], [375, 3.8228302], [122, 3.8117828]] |

```

5장 권장 사항

```
|7982 [[191, 5.297085], [547, 5.020829], [187, 4.984965]] |
|44022 [[12, 4.5919843], [53, 4.501897], [523, 4.301981]] |
|141422 [[456, 2.7050805], [597, 2.6988854], [498, 2.6347125]]|
|833 [[53, 3.8047972], [543, 3.740805], [12, 3.6920836]] |
|5803 [[537, 3.8269677], [544, 3.8034997], [259, 3.76062]] |
|7993 [[375, 2.93635], [53, 2.9159238], [191, 2.8663528]] |
|160563 [[53, 4.048704], [243, 3.9232922], [337, 3.7616432]] |
+-----+

```

상위 20개 행만 표시

지정된 영화 세트에 대한 상위 3개의 사용자 추천을 생성합니다.

```
model.recommendForItemSubset(Seq((111), (202), (225), (347), (488)).
toDF("movieId"), 3).show(거짓)
```

```
+-----+
|movieId|추천
+-----+
|225    |[[53, 4.4893017], [147, 4.483344], [276, 4.2529426]]|
|111    |[[375, 5.113064], [53, 4.9947076], [236, 4.9493203]]|
|347    |[[191, 4.686208], [236, 4.51165], [40, 4.409832]] |
|202    |[[53, 3.349618], [578, 3.255436], [224, 3.245058]] |
|488    |[[558, 3.3870435], [99, 3.2978806], [12, 3.2749753]]|
+-----+
```

지정된 사용자 집합에 대한 상위 3개의 추천 영화를 생성합니다.

```
model.recommendForUserSubset(Seq((111), (100), (110), (120), (130)).
toDF("userId"), 3).show(거짓)
```

```
+-----+
|userId|권장사항
+-----+
|111  |[[106100, 4.956068], [128914, 4.9050474], [162344, 4.9050474]]|
|120  |[[26865, 4.979374], [3508, 4.6825113], [3200, 4.6406555]] |
|100  |[[42730, 5.2531567], [5867, 5.1075697], [3404, 5.0877166]] |
|130  |[[86377, 5.224841], [3525, 5.0586476], [92535, 4.9758487]] |
|110  |[[49932, 4.6330786], [7767, 4.600622], [26171, 4.5615706]] |
+-----+
```

협업 필터링은 관련성이 높은 권장 사항을 제공하는 데 매우 효과적일 수 있습니다. 확장성이 뛰어나고 매우 큰 데이터 세트를 처리할 수 있습니다. 협업 필터링이 최적으로 작동하려면 많은 양의 데이터에 액세스해야 합니다. 데이터가 많을수록 좋습니다. 시간이 흐르고 평가가 누적되기 시작하면 추천이 점점 더 정확해집니다. 대규모 데이터 세트에 대한 액세스는 구현 초기 단계에서 종종 문제가 됩니다. 한 가지 솔루션은 협업 필터링과 함께 콘텐츠 기반 필터링을 사용하는 것입니다. 콘텐츠 기반 필터링은 사용자 활동에 의존하지 않기 때문에 즉시 권장 사항을 제공하기 시작하여 시간이 지남에 따라 점차적으로 데이터 세트를 늘릴 수 있습니다.

FP-Growth와 함께 시장 바구니 분석

장바구니 분석은 소매업체에서 제품 추천을 제공하기 위해 일반적으로 사용하는 간단하지만 중요한 기술입니다. 트랜잭션 데이터 세트를 사용하여 함께 자주 구매하는 제품을 결정합니다. 소매업체는 권장 사항을 사용하여 개인화된 교차 판매 및 상향 판매 정보를 제공함으로써 전환율을 높이고 각 고객의 가치를 극대화 할 수 있습니다.

Amazon.com을 검색하는 동안 이미 장바구니 분석이 실행되는 것을 보았을 것입니다. Amazon.com 제품 페이지에는 일반적으로 현재 탐색 중인 제품과 함께 자주 구매하는 품목 목록을 표시하는 "이 품목을 구매한 고객이 함께 구매한 품목"이라는 섹션이 있습니다. 해당 목록은 장바구니 분석을 통해 생성됩니다. 장바구니 분석은 매장 최적화를 위해 오프라인 소매업체에서도 플래노그램으로 제품 배치 및 인접 항목을 알려주는 데 사용됩니다. 아이디어는 서로 보완적인 항목을 배치하여 더 많은 판매를 유도하는 것입니다.

장바구니 분석은 연관 규칙 학습을 사용하여 권장 사항을 만듭니다. 연결 규칙은 대규모 트랜잭션 데이터 집합을 사용하여 항목 간의 관계를 찾습니다.^{xiv} 연결 규칙은 항목 집합이라는 두 개 이상의 항목에서 계산됩니다. 연관 규칙은 선행(if)과 후건(then)으로 구성됩니다. 예를 들어, 누군가가 쿠키를 구매하면(선행) 우유도 구매할 가능성이 더 높아집니다(결과). 인기 있는 연관 규칙 알고리즘에는 Apriori, SETM, ECLAT 및 FP-Growth가 있습니다. Spark MLlib에는 연관 규칙 마이닝을 위한 확장성이 뛰어난 FP-Growth 구현이 포함되어 있습니다.^{xv} FP-Growth는 빈번한 항목을 식별하고 빈번한 패턴 ("FP"는 빈번한 패턴을 나타냄) 트리 구조를 사용하여 항목 빈도를 계산합니다.^{xvi}

5장 권장 사항

참고 FP-Growth는 Jiawei Han, Jian Pei 및 Iwen Yin의 "Mining Frequency Patterns Without Candidate Generation" 백서에 설명되어 있습니다 .xvii

예시

FP-Growth.xviii를 사용한 장바구니 분석 예제에 인기 있는 Instacart Online Grocery Shopping Dataset을 사용할 것입니다 . 이 데이터 세트에는 200,000 Instacart 고객의 50,000 제품에 대한 340만 식료품 주문이 포함되어 있습니다 . www.instacart.com/datasets/grocery-shopping-2017에서 데이터 세트를 다운로드할 수 있습니다 . FP-Growth의 경우 products 및 order_products_train 테이블만 필요합니다(목록 5-2 참조).

목록 5-2. FP-Growth와 함께 시장 바구니 분석

```
val 제품DF = spark.read.format("csv")
    .option("헤더", "참")
    .option("inferSchema", "true")
    .load("/instacart/products.csv")
```

ProductsDF.show(거짓)

	product_id	제품_이름	
1		초콜릿 샌드위치 쿠키	
2		사계절 소금	
3		견고한 황금 무가당 우롱차	
4		Smart Ones Classic 즐겨찾기 Mini Rigatoni With	
5		그린칠레 앤스타임 소스	
6		드라이 노즈 오일	
7		오렌지와 순수한 코코넛 워터	
8		Cut Russet Potatoes Steam N' Mash	
9		라이트 스트로베리 블루베리 요거트	
10		스파클링 오렌지 주스 & 대추 음료	
11		복숭아 망고 주스	
12		초콜릿 퍼지 레이어 케이크	

13	식염수 비강 미스트
14	상큼한 향의 식기세척기 클리너
15	야간 기저귀 사이즈 6
16	민트초코 시럽
17	렌더링된 오리 지방
18	슈프리마 냉동 피자 피자
19	글루텐 프리 퀴노아 3가지 치즈 & 버섯 블렌드
20	석류 크랜베리 & 알로에 베라 농축 음료

+-----+-----+

|aisle_id|department_id|

+-----+-----+

61	19	
104	13	
94	7 1	
38 5	13	
11	11	
98	7 1	
116	16	
120	7 7	
115	1	
31	11	
119	17	
11	18	
74	19	
56	12	
103	1 9	
35	7	
79		
63		
98		

+-----+-----+

상위 20개 행만 표시

5장 권장 사항

```
val orderProductsDF = spark.read.format("csv")
    .option("헤더", "참")
    .option("inferSchema","true")
    .load("/instacart/order_products__train.csv")
```

orderProductsDF.show()

order_id	product_id	add_to_cart_order	재주문
1	49302	1	1
1	11109	2	1
1	10246	3	0
1	49683	4	0
1	43633	5	1
1	13176	6	0
1	47209	7	0
1	22035	8	1
36	39612	1	0
36	19660	2	1
36	49235	3	0
36	43086	4	1
36	46620	5	1
36	34497	6	1
36	48679	7	1
36	46979	8	1
38	11913	1	0
38	18159	2	0
38	4461	3	0
38	21616	4	1

상위 20개 행만 표시

// 임시 테이블을 생성합니다.

```
orderProductsDF.createOrReplaceTempView("order_products_train")
productsDF.createOrReplaceTempView("제품")
```

```
val JoinData = spark.sql("order_product_train에서 p.product_name, o.order_id 선택 o 내부 조인 제품 p 여기서 p.product_id = o.product_id")
```

org.apache.spark.sql.functions.max 가져오기

org.apache.spark.sql.functions.collect_set 가져오기

val 바구니DF =

```
JoinData .groupBy("order_id") .agg(collect_set("제품  
이름") .alias("항목"))
```

```
바구니DF.createOrReplaceTempView("바구니")
```

바구니DF.show(20,55)

+-----+-----+

주문 ID	아이템
1342 [생 새우, 씨 없는 오이, 다목적 얼룩 렘...]	1591 [깨진 밀, 딸기 대황 요구르트, 유기농 ...]
4519 [비트 사과 햄버거]	1342 [생 새우, 씨 없는 오이, 다목적 얼룩 렘...]
1666 [비정화 소금]	1591 [깨진 밀, 딸기 대황 요구르트, 유기농 ...]
67089 [여자 캐스팅 티셔츠]	4519 [비트 사과 햄버거]
92317 [붉은 덩굴 토마토, 수확 캐서를 그릇, 유기농 B...]	1342 [생 새우, 씨 없는 오이, 다목적 얼룩 렘...]
99621 [유기농 야거 투스터 티셔츠]	1591 [깨진 밀, 딸기 대황 요구르트, 유기농 ...]

5장 권장 사항

상위 20개 행만 표시

org.apache.spark.ml.fpm.FPGrowth 가져오기

// FPGrowth는 항목 목록을 포함하는 문자열만 필요합니다.

```
val basketsDF = spark.sql("바구니에서 항목 선택") .as[Array[String]].toDF("items")
```

바구니DF.show(20,55)

아이템
생새우, 씨 없는 오이, 다목적 얼룩 제거제... [깨진 밀, 딸기 대황 요구르트, 유기농 ...] 사과 당근 레몬 생강 유기농 냉압착... [보드카] [글로브 가지, 판코 빵 부스러기, 생 모짜렐라 치 즈... 유기농 베이비 시금치, 유기농 스프링 믹스, 유기농 L... 저지방 크래커, 식기세척기 세제, 땅콩포... 유기농 적양파 소량 정통 타케리아 티... 유기농 크립스 핑크 사과, 유기농 황금감자 기농 아기 시금치, 푸른빛이 도는 유기농 블루스 빵... 비경화 핫도그, 유기농 아기 시금치, 훈 제... 도넛하우스 초콜릿 글레이즈드 도넛 커피 K컵, 마... 농축 정육점, 닭고기, 씨... 라즈 베리, 씨 없는 녹색 포도, 클레멘타인, 나... 오리지날 토퍼키 델리 슬라이스, 샤프 체다 치즈,... 엑스트라 헐드 논 에어로졸 헤어 스프레이, 욕실 티슈,... 유기농 코코넛 밀크, 에브리싱 베이 글, 로즈마리, ... 아니요. 485 진, 몬테레이 잭 슬라이스 치즈, 전통... 붉은 덩굴 토마토, 수 확 캐서를 그릇, 유기농 B... 유기농 아기 루꼴라, 유기농 마늘, 회향, 레몬...

상위 20개 행만 표시

```

val fpgrowth = 새로운 FPGrowth()
    .setItemsCol("항목")
    .setMinSupport(0.002)
    .setMinConfidence(0)

val 모델 = fpgrowth.fit(basketsDF)

// 빈번한 항목 집합.

val mostPopularItems = model.freqItemsets

mostPopularItems.createOrReplaceTempView("mostPopularItems")

// 결과를 확인합니다.

spark.sql("select   from mostPopularItems wheresize(items) >= 2 order by freq desc")

.show(20,55)
+-----+---+
| 항목|빈도|
+-----+---+
|[유기농 딸기, 유기농 바나나 한 봉지]|3074| |
|[유기농 하스 아보카도, 유기농 바나나 한 봉지]|2420|
|[유기농 아기 시금치, 유기농 바나나 한 봉지]|2236|
| | [유기농 아보카도, 바나나]|2216|
| | [유기농 딸기, 바나나]|2174|
| | [라지 레몬, 바나나]|2158|
| | [유기농 아기 시금치, 바나나]|2000|
| | [딸기, 바나나]|1948|
| | [유기농 산딸기, 유기농 바나나 한 봉지]|1780|
| | [유기농 산딸기, 유기농 딸기]|1670|
| | [유기농 아기 시금치, 유기농 딸기]|1639|
| | [라임, 라지 레몬]|1595|
| | [유기농 하스 아보카도, 유기농 딸기]|1539|
| | [유기농 아보카도, 유기농 베이비 시금치]|1402|
| | [유기농 아보카도 라지 레몬]|1349|
| | [라임, 바나나]|1331|

```

5장 권장 사항

```
| [[유기농 블루베리, 유기농 딸기]]|1269|
|     [[유기농 오이, 유기농 바나나 봉지]]|1268|
| [[유기농 하스 아보카도, 유기농 베이비 시금치]]|1252|
|     [[큰 레몬, 유기농 아기 시금치]]|1238|
+-----+--+
```

상위 20개 행만 표시

spark.sql("mostPopularItems에서 를 선택하십시오. 여기서

```
size(items) > 2 order by freq desc")
.show(20,65)
```

	항목 빈도
[[유기농 하스아보카도, 유기농 딸기, 유기농 바...]] 710	
[[유기농 산딸기, 유기농 딸기, 유기농 금지 봉지...]] 649	
[[유기농 아기 시금치, 유기농 딸기, 유기농 바...]] 587	
[[유기농 산딸기, 유기농 하스아보카도, 유기농 금지 봉지...]] 531	
[[유기농 하스 아보카도, 유기농 베이비 시금치, 유기농 바...]] 497	
[[유기농 아보카도, 유기농 베이비 시금치, 바나나]] 484	
[[유기농 아보카도, 라지 레몬, 바나나]] 477	
[[라임, 라지 레몬, 바나나]] 452	
[[유기농 오이, 유기농 딸기, 유기농 바나나 한 봉지]] 424	
[[라임, 유기농 아보카도, 라지 레몬]] 389	
[[유기농 산딸기, 유기농 하스아보카도, 유기농 딸기]] 381	
[[유기농 아보카도, 유기농 딸기, 바나나]] 379	
[[유기농 아기 시금치, 유기농 딸기, 바나나]] 376	
[[유기농 블루베리, 유기농 딸기, 유기농 금지 봉지...]] 374	
[[라지 레몬, 유기농 베이비 시금치, 바나나]] 371	
[[유기농 오이, 유기농 하스 아보카도, 유기농 바나나 한 봉지]] 366	
[[유기농 레몬, 유기농 하스 아보카도, 유기농 바나나 한 봉지]] 353	
[[라임, 유기농 아보카도, 바나나]] 352	
[[유기농 전유, 유기농 딸기, 유기농 바나나 한 봉지...]] 339	
[[유기농 아보카도, 라지 레몬, 유기농 베이비 시금치]] 334	

상위 20개 행만 표시

함께 구매할 가능성이 가장 높은 항목이 표시됩니다. 목록에서 가장 인기 있는 것은 유기농 아보카도, 유기농 딸기, 유기농 바나나 한 봉지의 조합입니다. 이러한 종류의 목록은 "자주 함께 구매하는" 유형의 권장 사항의 기초가 될 수 있습니다.

// FP-Growth 모델은 연관 규칙도 생성합니다. 출력에는 다음이 포함됩니다.

// 전건, 후건, 신뢰도(확률). 최소

// 연관 규칙 생성에 대한 신뢰도는 다음과 같이 결정됩니다.

// minConfidence 매개변수.

```
val AssocRules = model.associationRules
```

```
AssocRules.createOrReplaceTempView("AssocRules")
```

```
spark.sql("전건, 후건,
```

```
    AssocRules의 신뢰 순서는 신뢰 설명")
```

```
.show(20,55)
```

```
+-----+  
| |  
+-----+
```

선행|

```
| [유기농 산딸기, 유기농 하스아보카도]]  
| [딸기, 유기농 아보카도]]  
| [유기농 하스아보카도, 유기농 딸기]]  
| [유기농 레몬, 유기농 하스아보카도]]  
| [유기농 레몬, 유기농 딸기]]  
| [유기농 오이, 유기농 하스아보카도]]  
|[유기농 대형 엑스트라 팬시 후지 사과, 유기농 밀짚...]  
| [유기농 황양파, 유기농 하스아보카도]]  
| [딸기, 라지 레몬]]  
| [유기농 블루베리, 유기농 라즈베리]]  
| [유기농 오이, 유기농 딸기]]  
| [유기농 호박, 유기농 하스아보카도]]  
| [유기농 산딸기, 유기농 아기 시금치]]  
| [유기농 하스 아보카도, 유기농 베이비 시금치]]  
| [유기농 호박, 유기농 딸기]]  
| [유기농 산딸기, 유기농 딸기]]
```

5장 권장 사항

[라임, 유기농 아보카도]	
[유기농 산딸기, 유기농 하스아보카도]	
+-----+-----+	
결과	자신감
+-----+-----+	
[[유기농 바나나 봉지]] 0.521099116781158	
[[바나나]] 0.4643478260869565	
[[유기농 바나나 봉지]] 0.4613385315139701	
[[유기농 바나나 봉지]] 0.4519846350832266	
[[유기농 바나나 봉지]] 0.4505169867060561	
[[유기농 바나나 봉지]] 0.4404332129963899	
[[유기농 바나나 봉지]] 0.4338461538461538	
[[유기농 바나나 봉지]] 0.42270861833105333	
[[바나나]] 0.4187779433681073	
[[유기농 딸기]] 0.414985590778098	
[[유기농 바나나 봉지]] 0.4108527131782946	
[[유기농 바나나 봉지]] 0.40930232558139534	
[[유기농 바나나 봉지]] 0.40706806282722513	
[[유기농 바나나 봉지]] 0.39696485623003197	
[[유기농 바나나 봉지]] 0.3914780292942743	
[[유기농 바나나 봉지]] 0.38862275449101796	
[[바나나]] 0.3860811930405965	
[[바나나]] 0.38373305526590196	
[[라지 레몬]] 0.3751205400192864	
[[유기농 딸기]] 0.37389597644749756	
+-----+-----+	

상위 20개 행만 표시

출력에 따르면 유기농 라즈베리, 유기농 아보카도, 유기농 딸기를 구매한 고객은 유기농 바나나도 구매할 가능성이 더 높습니다. 보시다시피 바나나는 매우 인기있는 품목입니다. 이러한 종류의 목록은 "이 항목을 구입한 고객이 또한 구입한 항목" 유형 추천의 기초가 될 수 있습니다.

참고 FP-Growth 외에도 Spark MLlib에는 PrefixSpan이라는 주파수 패턴 일치 알고리즘의 또 다른 구현이 포함되어 있습니다. FP-Growth는 항목 집합이 정렬되는 방식에 대해 무관심한 반면 PrefixSpan은 데이터 집합에서 순차적 패턴을 발견하기 위해 시퀀스 또는 항목 집합의 정렬된 목록을 사용합니다. PrefixSpan은 순차 패턴 마이닝으로 알려진 알고리즘의 하위 그룹에 속합니다.

PrefixSpan은 Jian Pei et al.의 "Mining Sequential Patterns by Pattern Growth: PrefixSpan Approach" 백서에 설명되어 있습니다.

콘텐츠 기반 필터링

콘텐츠 기반 필터링은 항목 이름, 설명 또는 범주와 같은 항목에 대한 정보를 사용자 프로필과 비교하여 권장 사항을 제공합니다. 예를 들어 영화에 대한 콘텐츠 기반 추천 시스템을 살펴보겠습니다. 시스템에서 프로필을 기반으로 사용자가 캐리 그랜트 영화를 선호한다고 판단하면 North by Northwest, To Catch a Thief, An Affair to Remember 와 같은 영화를 추천하기 시작할 수 있습니다. 추천자는 Jimmy Stewart, Gregory Peck 또는 Clark Gable(클래식 영화)과 같은 같은 장르의 배우들의 영화를 추천할 수 있습니다. Cary Grant와 자주 공동 작업을 하는 Alfred Hitchcock과 George Cukor가 감독한 영화도 추천할 수 있습니다. 단순함에도 불구하고 콘텐츠 기반 추천 엔진은 일반적으로 적절한 결과를 제공합니다. 구현하기도 쉽습니다. 시스템은 사용자의 명시적 또는 묵시적 피드백을 기다리지 않고 즉시 권장 사항을 제공할 수 있습니다. 이는 협업 필터링과 같은 다른 방법을 괴롭히는 힘든 요구 사항입니다.

단점으로 콘텐츠 기반 필터링은 권장 사항에 다양성과 참신함이 부족합니다. 시청자는 때때로 더 다양한 영화를 원하거나

시청자의 프로필과 완벽하게 일치하지 않을 수 있는 조금 더 전위적인 것. 확장성은 콘텐츠 기반 추천 시스템을 괴롭히는 또 다른 과제입니다. 관련성이 높은 추천을 생성하기 위해 콘텐츠 기반 엔진은 추천하는 항목에 대한 많은 양의 도메인 관련 정보를 필요로 합니다.xix 영화 추천자가 제목, 설명 또는 장르만을 기준으로 추천하는 것만으로는 충분하지 않습니다. 사내 데이터는 IMDB 또는 Rotten Tomatoes와 같은 타사 소스의 데이터로 보강해야 할 수 있습니다. LDA(Latent Dirichlet Allocation)와 같은 비지도 학습 방법을 사용하여 이러한 데이터 소스에서 추출한 메타데이터에서 새로운 주제를 생성할 수 있습니다. LDA에 대해서는 4장에서 자세히 설명합니다.

5장 권장 사항

Netflix는 다음을 제공하는 수천 개의 마이크로 장르를 만들어 이 분야를 선도하고 있습니다.

고도로 타겟팅된 개인화된 권장 사항. 예를 들어, Netflix는 제 아내가 한국 영화를 보는 것을 좋아할 뿐만 아니라 로맨틱 뮤지컬 한국 영화, 머더 미스터리 좀비 한국 영화, 다큐드라마 갱스터 한국 좀비 영화, 그리고 그녀가 개인적으로 가장 좋아하는 법정 드라마 법률 스릴러 좀비 한국 영화를 좋아한다는 것을 알고 있습니다. 기계식 터키식 시스템을 통해 Netflix는 시간제 영화 애호가를 고용하여 라이브러리에 있는 수천 편의 영화와 TV 프로그램에 설명과 범주를 수동으로 할당했습니다. 마지막으로 Netflix에는 76,897개의 고유한 마이크로 장르가 있었습니다. 이것은 Netflix 이외의 곳에서는 볼 수 없는 새로운 차원의 개인화입니다.

Spark MLlib에는 콘텐츠 기반 필터링을 위한 알고리즘이 포함되어 있지 않지만 Spark에는 자체 구현을 개발하는 데 도움이 되는 구성 요소가 있습니다. 시작하려면 "MapReduce를 사용하는 Dimension Independent Matrix Square" 또는 줄여서 DIMSUM이라고 하는 Spark에서 사용할 수 있는 확장성이 뛰어난 유사성 알고리즘을 살펴보는 것이 좋습니다. <https://bit.ly/2YV6qTr> 확인 DIMSUM.xx에 대해 자세히 알아보려면

요약

각 방법에는 고유한 강점과 약점이 있습니다. 실제 시나리오에서는 결과를 향상시키기 위해 여러 기술을 결합하여 하이브리드 추천 엔진을 구축하는 것이 일반적입니다. 추천자는 연구를 위한 비옥한 영역입니다. 일부 세계 최대 기업에서 창출하는 수익을 고려할 때 이 분야에서 곧 더 많은 발전이 있을 것으로 기대합니다. FP-Growth 예제는 Databricks.xxi에서 Bhavin Kukadia와 Denny Lee의 작업에서 채택되었습니다.

참고문헌

나. 르네 지라르; "르네 지라르와 모방 이론," imitatio.org, 2019,
www.imitatio.org/brief-intro

ii. 마이클 오스본; "소매 브랜드가 Amazon의 전술을 사용하여 경쟁하고 승리할 수 있는 방법", forbes.com, 2017, www.forbes.com/sites/forbesagencycouncil/2017/12/21/how-retail-brands-can-compete-and-win-using-amazons-tactics/#4f4e55bc5e18

5장 권장 사항

- xii. 아파치 스파크; "협업 필터링", spark.apache.org, 2019, <https://spark.apache.org/docs/latest/ml-collaborative-filtering.html>
- xiii. 아파치 스파크; "협업 필터링", spark.apache.org, 2019, <https://spark.apache.org/docs/latest/ml-collaborative-filtering.html>
- xiv. 마가렛 라우즈; "연관 규칙(데이터 마이닝에서)", techttarget.com, 2018, <https://searchbusinessanalytics.techttarget.com/definition/association-rules-in-data-mining>
- xv. 아파치 스파크; "빈번한 패턴 마이닝", spark.apache.org, 2019년, <https://spark.apache.org/docs/2.3.0/mllib-frequent-pattern-mining.html>
16. Jiawei Han et al.; "후보 생성 없이 빈번한 패턴 마이닝", acm.org, 2000, <https://dl.acm.org/citation.cfm?doid=335191.335372>
- xvii. Jiawei Han, et al.; "후보 생성 없이 빈번한 패턴 마이닝", acm.org, 2000, <https://dl.acm.org/citation.cfm?id=335372%C3%DC>
- xviii. Bhavin Kukadia 및 Denny Lee; "Databricks에서 FP 성장을 사용하여 시장 바구니 분석 단순화", Databricks.com, 2018, <https://databricks.com/blog/2018/09/18/simplify-market-basket-analysis-using-fp-growth-on-databricks.html>
- xix. 타일러 키넌; "콘텐츠 기반 필터링이란 무엇입니까?", upwork.com, 2019, www.upwork.com/hiring/data/what-is-content-based-filtering/
- 더블 엑스. 레자 자데; "Twitter 덕분에 Apache Spark의 효율적인 유사성 알고리즘" Databricks.com, 2014년, <https://databricks.com/blog/2014/10/20/efficient-similarity-algorithm-now-in-spark-twitter.html>
- xxi. Bhavin Kukadia 및 Denny Lee; "Databricks에서 FP 성장을 사용하여 시장 바구니 분석 단순화", Databricks.com, 2018, <https://databricks.com/blog/2018/09/18/simplify-market-basket-analysis-using-fp-growth-on-databricks.html>

6장

그래프 분석

내가 거기에 가지 않도록 내가 죽을 곳을 알려주세요.

—찰리 멍게리

그래프 분석은 그래프에서 개체 간의 관계의 강도와 방향을 결정하기 위한 데이터 분석 기술입니다. 그래프는 개체 간의 관계 및 프로세스를 모델링하는 데 사용되는 수학적 구조입니다.ⁱⁱ 데이터의 복잡한 관계 및 종속성을 나타내는 데 사용할 수 있습니다. 그래프는 시스템의 엔터티를 나타내는 꼭짓점 또는 노드로 구성됩니다. 이 꼭짓점은 해당 엔터티 간의 관계를 나타내는 가장자리로 연결됩니다.ⁱⁱⁱ

그래프 소개

즉시 명확하지 않을 수도 있지만 그래프는 도처에 있습니다. LinkedIn, Facebook 및 Twitter와 같은 소셜 네트워크는 그래프입니다. 인터넷과 월드 와이드 웹은 그래프입니다. 컴퓨터 네트워크는 그래프입니다. 수도 시설 파이프라인은 그래프입니다.

도로 네트워크는 그래프입니다. GraphX와 같은 그래프 처리 프레임워크에는 그래프 지향 데이터를 처리하도록 특별히 설계된 그래프 알고리즘과 연산자가 있습니다.

무방향 그래프

무방향 그래프는 방향이 없는 간선이 있는 그래프입니다. 무방향 그래프의 간선은 양방향으로 순회할 수 있으며 양방향 관계를 나타냅니다. 그림 6-1은 3개의 노드와 3개의 간선이 있는 무방향 그래프를 보여줍니다.^{iv}

6장 그래프 분석

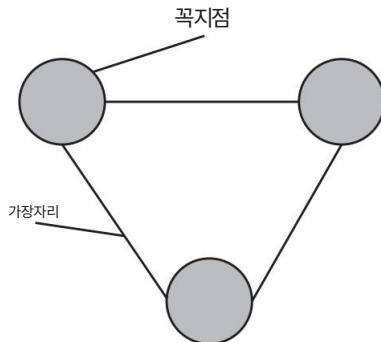


그림 6-1. 무방향 그래프

방향 그래프

유방향 그래프에는 단방향 관계를 나타내는 방향이 있는 간선이 있습니다. 방향 그래프에서 각 모서리는 한 방향으로만 이동할 수 있습니다(그림 6-2).

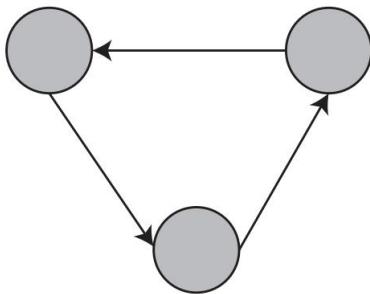


그림 6-2. 유향 그래프

방향성 다중 그래프

다중 그래프에는 노드 사이에 여러 개의 간선이 있습니다(그림 6-3).

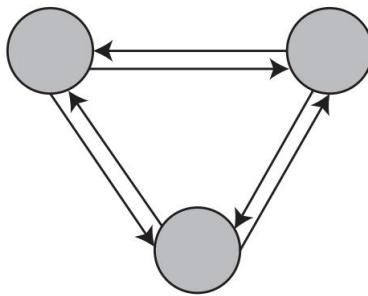


그림 6-3. 방향성 다중 그래프

속성 그래프

속성 그래프는 정점과 모서리에 사용자 정의 속성이 있는 방향성 다중 그래프입니다.v (그림 6-4).

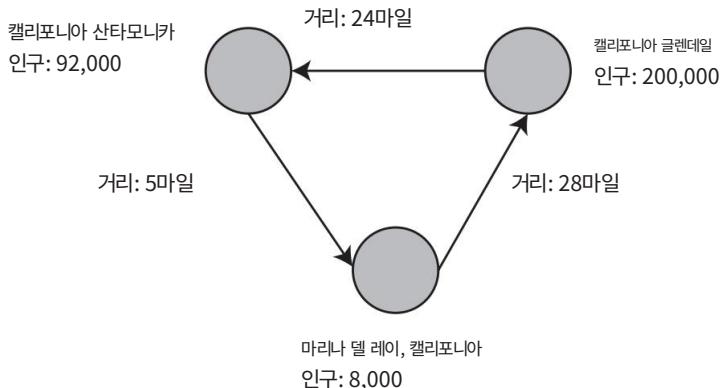


그림 6-4. 속성 그래프

그래프 분석 사용 사례

그래프 분석은 다양한 산업 분야에서 번성했습니다. 많은 양의 연결된 데이터를 활용하는 모든 프로젝트는 그래프 분석을 위한 완벽한 사용 사례입니다. 이것은 사용 사례의 포괄적인 목록을 의미하지는 않지만 그래프 분석의 가능성에 대한 아이디어를 제공해야 합니다.

6장 그래프 분석

사기 탐지 및 자금 세탁 방지(AML)

사기 탐지 및 자금 세탁 방지는 아마도 그래프 분석의 가장 잘 알려진 사용 사례 중 하나일 것입니다. 수백만 건의 은행 거래를 수동으로 선별하여 사기 및 자금 세탁 활동을 식별하는 것은 불가능하지는 않더라도 벽찬 일입니다.

복잡한 일련의 상호 연결되고 겉보기에 무해한 거래를 통해 사기를 숨기는 정교한 방법으로 인해 문제가 더욱 복잡해졌습니다. 이러한 유형의 공격을 감지하고 방지하는 전통적인 기술은 오늘날에는 쓸모가 없습니다. 그래프 분석은 의심스러운 거래를 다른 비정상적인 행동 패턴과 쉽게 연결하여 이 문제에 대한 완벽한 솔루션입니다. GraphX와 같은 고성능 그래프 처리 API를 사용하면 한 트랜잭션에서 다른 트랜잭션으로 매우 빠르게 복잡한 순회를 수행할 수 있습니다. 아마도 가장 잘 알려진 사례는 파나마 페이퍼스 스캔들일 것입니다. 분석가는 그래프 분석을 사용하여 수백만 건의 유출된 문서 간의 연관성을 식별함으로써 저명한 외국 지도자, 정치인, 심지어 Queen.vi가 소유한 역외 은행 계좌에서 숨겨진 자산을 발견할 수 있었습니다.

데이터 거버넌스 및 규정 준수

일반적인 대규모 조직은 수백만 개의 데이터 파일을 데이터 레이크와 같은 중앙 데이터 저장소에 저장합니다. 이를 위해서는 파일이 수정되거나 새 복사본이 생성될 때마다 데이터 계보를 추적하는 적절한 마스터 데이터 관리가 필요합니다. 데이터 계보를 추적함으로써 조직은 소스에서 대상으로의 데이터 이동을 추적하여 지점 간에 변경된 모든 방식에 대한 가시성을 제공할 수 있습니다.^{vii} 건전한 데이터 계보 정책은 데이터에 대한 신뢰를 촉진하고 정확한 비즈니스 결정을 가능하게 합니다. 데이터 계보는 GDPR(일반 데이터 보호 규정)과 같은 데이터 관련 규정 준수를 유지하기 위한 요구 사항이기도 합니다. GDPR 위반으로 적발되면 막대한 벌금이 부과될 수 있습니다. 그래프 분석은 이러한 사용 사례에 적합합니다.

위기 관리

위험을 관리하기 위해 헤지 펀드 및 투자 은행과 같은 금융 기관은 그래프 분석을 사용하여 "블랙 스완" 이벤트를 유발할 가능성이 있는 상호 연결된 위험 및 패턴을 식별합니다. 2008년 금융 위기를 완벽한 예로 사용하여 그래프 분석은 모기지 담보부 증권(MBS), 담보부 채무(CDO) 및 신용 채무 불이행 간의 복잡한 상호 의존성을 조명함으로써 파생 상품 증권화의 복잡한 프로세스에 대한 가시성을 제공할 수 있었습니다. CDO의 트랜치에 배치된 스왑.

운송

항공 교통 네트워크는 그래프입니다. 공항은 정점을 나타내고 경로는 가장자리를 나타냅니다. 상용 비행 계획은 그래프 분석을 사용하여 생성됩니다. 항공사 및 물류 회사는 경로 최적화를 위해 그래프 분석을 사용하여 가능한 가장 안전하거나 가장 빠른 경로를 결정합니다. 이러한 최적화는 안전을 보장하고 시간과 비용을 절약할 수 있습니다.

소셜 네트워킹

소셜 네트워킹은 그래프 분석을 위한 가장 직관적인 사용 사례입니다. 오늘날 거의 모든 사람은 소셜 네트워크에서 다른 사람 또는 그룹과의 사회적 관계를 나타내는 그래프인 "소셜 그래프"를 가지고 있습니다. Facebook, Twitter, Instagram 또는 LinkedIn을 사용 중이라면 소셜 그래프가 있습니다. 좋은 예는 Facebook이 또 다른 오픈 소스 그래프 프레임워크인 Apache Giraph를 사용하여 콘텐츠 순위 및 권장 사항에 대한 1조 개의 가장자리를 처리하고 분석하는 것입니다.^{viii}

네트워크 인프라 관리

인터넷은 라우터, 서버, 데스크탑, IoT 및 모바일 장치가 상호 연결된 거대한 그래프입니다. 정부 및 기업 네트워크는 인터넷의 하위 부분으로 간주될 수 있습니다. 작은 홈 네트워크도 그래프입니다. 장치는 정점을 나타내고 네트워크 연결은 가장자리를 나타냅니다. 그래프 분석은 네트워크 관리자에게 복잡한 네트워크 토플로지를 시각화 할 수 있는 기능을 제공하므로 모니터링 및 문제 해결에 유용합니다. 이는 수천 개의 연결된 장치로 구성된 대규모 기업 네트워크에 특히 유용합니다.

GraphX 소개

GraphX는 Spark의 RDD 기반 그래프 처리 API입니다. 그래프 연산자 및 알고리즘 외에도 GraphX는 그래프 데이터를 저장하기 위한 데이터 유형을 제공합니다.^{ix}

그래프

속성 그래프는 Graph 클래스의 인스턴스로 표시됩니다. RDD와 마찬가지로 속성 그래프는 실행 프로그램 간에 분할 및 배포되며 충돌이 발생할 경우 다시 생성할 수 있습니다. 속성 그래프는 변경할 수 없습니다. 즉, 그래프의 구조 또는 값을 변경하려면 새 그래프를 생성해야 합니다.^x

6장 그래프 분석

정점RDD

속성 그래프의 정점은 VertexRDD로 표시됩니다. 예 대한 단 하나의 항목 각 정점은 VertexRDD에 저장됩니다.

가장자리

Edge 클래스에는 edge property.xi를 저장하는 속성뿐만 아니라 소스 및 대상 정점 식별자에 해당하는 소스 및 대상 ID가 포함되어 있습니다.

엣지RDD

속성 그래프의 가장자리는 EdgeRDD로 표시됩니다.

에지트리플렛

모서리와 모서리가 연결하는 두 정점의 조합은 EdgeTriplet 클래스의 인스턴스로 표시됩니다. 그것은 또한 edge의 속성과 연결하는 정점.

에지컨텍스트

EdgeContext 클래스는 소스 및 대상 정점에 메시지를 보내는 기능뿐만 아니라 삼중항 필드를 노출합니다.xii

GraphX 예제

이 예(Listing 6-1)에서는 GraphX를 사용하여 남부 캘리포니아의 여러 도시 간의 거리를 분석합니다. 우리는 그림 6-5와 같은 속성 그래프를 만들 것입니다.
표 6-1의 데이터를 기반으로 합니다.

표 6-1. 다른 남부 캘리포니아 도시 간의 거리

원천	목적지	거리
캘리포니아 산타모니카	마리나 델 레이, 캘리포니아	5 마일
캘리포니아 산타모니카	캘리포니아 글렌데일	24마일
마리나 델 레이, 캘리포니아	캘리포니아 글렌데일	28마일
캘리포니아 글렌데일	캘리포니아 패서디나	9마일
캘리포니아 패서디나	캘리포니아 글렌데일	9마일

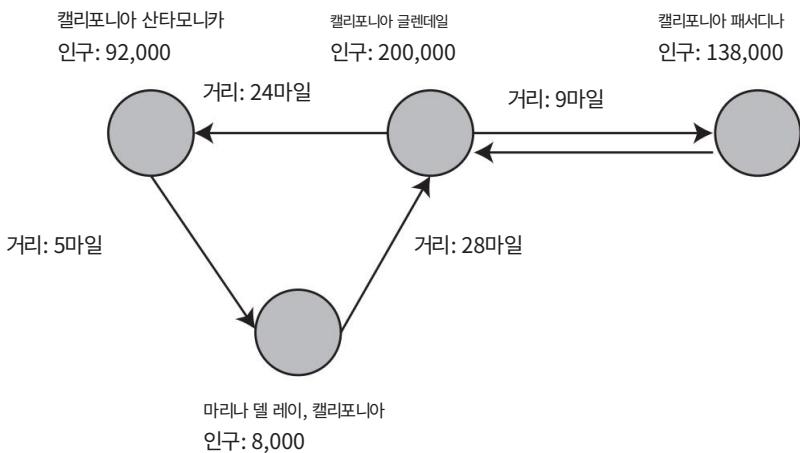


그림 6-5. 속성 그래프

목록 6-1. GraphX 예제xiii

```

import org.apache.spark.rdd.RDD
import org.apache.spark.graphx._

// 정점을 정의합니다.

val 정점 = Array((1L, ("산타모니카","CA")), (2L,("마리나 델
레이","CA")), (3L, ("글렌데일","CA")), (4L, ("패서디나","CA")))

val vRDD = sc.parallelize(정점)

```

6장 그래프 분석

```
// 가장자리를 정의합니다.

val edge = Array(Edge(1L,2L,5),Edge(1L,3L,24),Edge(2L,3L,28),Edge(3L,4L,9),
엣지(4L,3L,9))

val eRDD = sc.parallelize(가장자리)

// 속성 그래프를 생성합니다.

val 그래프 = 그래프(vRDD,eRDD)

graph.vertices.collect.foreach(println)

(3,(캘리포니아주 글렌데일))
(4,(캘리포니아 패서디나))
(1,(샌타모니카,CA))
(2,(마리나 텔 레이, CA))

graph.edges.collect.foreach(println)

엣지(1,2,5)
엣지(1,3,24)
엣지(2,3,28)
엣지(3,4,9)
엣지(4,3,9)

// 꼭짓점의 수를 반환합니다.

val numCities = graph.numVertices
numCities: Long = 4

// 모서리의 수를 반환합니다.

val numRoutes = graph.numEdges
numRoutes: Long = 5

// 각 꼭짓점에 대해 들어오는 가장자리 수를 반환합니다.

graph.inDegrees.collect.foreach(println) (3,3) (4,1)
(2,1)
```

```
// 각 정점에 대한 나가는 가장자리의 수를 반환합니다.
```

```
graph.outDegrees.collect.foreach(println) (3,1) (4,1)  
(1,2) (2,1)
```

```
// 20마일 미만의 모든 경로를 반환합니다.
```

```
graph.edges.filter{ case Edge(src, dst, prop) => prop < 20 }.collect. foreach(println)
```

```
에지(1,2,5)
```

```
에지(3,4,9)
```

```
에지(4,3,9)
```

```
// EdgeTriplet 클래스에는 소스 및 대상 속성이 포함됩니다.
```

```
graph.triplets.collect.foreach(println)
```

```
((1,(캘리포니아주 산타모니카)),(2,(캘리포니아주 마리나 델 레이)),5)  
((1,(캘리포니아주 산타모니카)),(3,(캘리포니아주 글렌데일)),24)  
((2,(Marina Del Rey,CA)),(3,(Glendale,CA)),28)  
((3,(캘리포니아주 글렌데일)),(4,(캘리포니아주 패서디나)),9)  
((4,(캘리포니아주 패서디나)),(3,(캘리포니아주 글렌데일)),9)
```

```
// 가장 먼 경로를 기준으로 정렬합니다.
```

```
graph.triplets.sortBy(_.attr, 오름차순=거짓).collect.foreach(println)  
((2,(Marina Del Rey,CA)),(3,(Glendale,CA)),28)  
((1,(캘리포니아주 산타모니카)),(3,(캘리포니아주 글렌데일)),24)  
((3,(캘리포니아주 글렌데일)),(4,(캘리포니아주 패서디나)),9)  
((4,(캘리포니아주 패서디나)),(3,(캘리포니아주 글렌데일)),9)  
((1,(캘리포니아주 산타모니카)),(2,(캘리포니아주 마리나 델 레이)),5)
```

```
// mapVertices는 모든 정점에 사용자 지정 함수를 적용합니다.
```

```
val newGraph = graph.mapVertices((vertexID, state) => "TX")
```

6장 그래프 분석

```
newGraph.vertices.collect.foreach(println)
(3,TX)
(4,TX)
(1,TX)
(2,TX)
```

// mapEdges는 모든 가장자리에 사용자 지정 함수를 적용합니다.

```
val newGraph = graph.mapEdges((edge) => "500")
```

```
엣지(1,2,500)
엣지(1,3,500)
엣지(2,3,500)
엣지(3,4,500)
엣지(4,3,500)
```

그래프 알고리즘

GraphX는 PageRank, 삼각형 개수 및 연결된 구성 요소와 같은 몇 가지 일반적인 그래프 알고리즘의 내장 구현과 함께 제공됩니다.

페이지 랭크

PageRank는 원래 Google에서 웹 페이지의 중요성을 결정하기 위해 개발한 알고리즘이다. PageRank가 높은 웹 페이지는 PageRank가 낮은 페이지보다 관련성이 높습니다. 페이지의 PageRank는 페이지에 링크되는 페이지의 PageRank에 따라 다릅니다. 따라서 반복 알고리즘입니다. 고품질 링크의 수도 페이지의 PageRank에 기여합니다. GraphX에는 PageRank의 내장 구현이 포함되어 있습니다. GraphX는 PageRank의 정적 및 동적 버전과 함께 제공됩니다.

동적 페이지 순위

동적 PageRank는 순위가 지정된 허용 오차 이상으로 업데이트를 중지할 때까지(즉, 순위가 수렴할 때까지) 실행됩니다.

```
val dynamicPageRanks = graph.pageRank(0.001).vertices
val sortedRanks = dynamicPageRanks.sortBy(_.value, ascending=false)
sortedRanks.collect.foreach(println)
```

```
(3,1.8845795504535865)
(4,1.7507334787248419)
(2,0.21430059110133595)
(1,0.15038637972023575)
```

정적 페이지 순위

정적 PageRank는 설정된 횟수만큼 실행됩니다.

```
val staticPageRanks = graph.staticPageRank(10)

val sortedRanks = staticPageRanks.vertices.sortBy(_.value, ascending=false)

sortedRanks.collect.foreach(println)
(4,1.8422463479403317) (3,1.7940036520596683)
(2,0.21375000000000008) (1,0.150000000000)
```

삼각형 개수

삼각형은 세 개의 연결된 꼭짓점으로 구성됩니다. 삼각형 카운트 알고리즘은 각 정점을 통과하는 삼각형의 수를 결정하여 클러스터링 측정을 제공합니다. 그림 6-5에서 Santa Monica, Marina Del Rey 및 Glendale은 모두 삼각형의 일부인 반면 Pasadena는 그렇지 않습니다.

```
val triangleCount = graph.triangleCount()

triangleCount.vertices.collect.foreach(println)
(3,1) (4,0)
(1,1) (2,1)
```

연결된 구성 요소

연결된 구성 요소 알고리즘은 하위 그래프에 있는 각 꼭짓점의 구성원을 결정합니다. 알고리즘은 하위 그래프에서 가장 낮은 번호의 꼭짓점의 꼭짓점 ID를 꼭짓점의 속성으로 반환합니다. 그림 6-6은 두 개의 연결된 구성 요소를 보여줍니다. Listing 6-2에 예제를 보여줍니다.

6장 그래프 분석

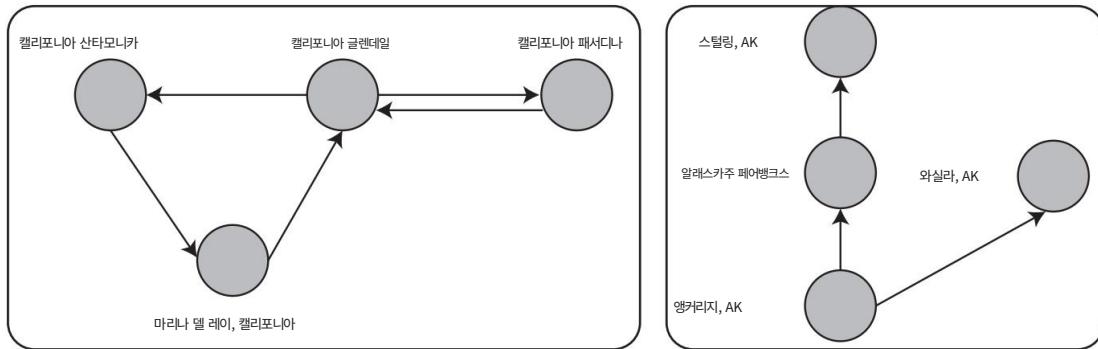


그림 6-6. 연결된 구성 요소

목록 6-2. 연결된 구성 요소 확인

```
val 정점 = Array((1L, ("산타모니카","CA")),(2L,("마리나 델 레이","CA")),
(3L, ("Glendale","CA")),(4L, ("Pasadena","CA")),(5L, ("앵커리지","AK")),
(6L, ("Fairbanks","AK")),(7L, ("Sterling","AK")),(8L, ("Wasilla","AK")))
```

```
val vRDD = sc.parallelize(정점)
```

```
val edge = Array(Edge(1L,2L,5),Edge(1L,3L,24),Edge(2L,3L,28),Edge(3L,4L,9),
엣지(4L,3L,9),엣지(5L,6L,32),엣지(6L,7L,28),엣지(5L,8L,17))
```

```
val eRDD = sc.parallelize(가장자리)
```

```
val 그래프 = 그래프(vRDD,eRDD)
```

```
graph.vertices.collect.foreach(println)
```

```
(6,(페어뱅크스, AK))
(3,(캘리포니아주 글렌데일))
(4,(캘리포니아 패서디나))
(1,(샌타모니카,CA))
(7,(스털링, AK))
(8,(와실라,AK))
(5,(앵커리지, AK))
(2,(마리나 델 레이, CA))
```

```
graph.edges.collect.foreach(println)  
  
에지(1,2,5)  
에지(1,3,24)  
에지(2,3,28)  
에지(3,4,9)  
에지(4,3,9)  
에지(5,6,32)  
엣지(5,8,17)  
엣지(6,7,28)  
  
val connectedComponents = graph.connectedComponents()  
  
ConnectedComponents.vertices.collect.foreach(println)  
  
(6,5)  
(3,1)  
(4,1)  
(1,1)  
(7,5)  
(8,5)  
(5,5)  
(2,1)
```

그래프 프레임

GraphFrames는 DataFrames 위에 구축된 그래프 처리 라이브러리입니다. 이 글을 쓰는 시점에서 GraphFrames는 여전히 활발히 개발 중이지만 핵심 Apache Spark 프레임워크의 일부가 되는 것은 시간 문제일 뿐입니다. GraphFrames를 GraphX보다 더 강력하게 만드는 몇 가지 사항이 있습니다. 모든 GraphFrames 알고리즘은 Java, Python 및 Scala에서 사용할 수 있습니다. GraphFrames는 친숙한 DataFrames API 및 Spark SQL을 사용하여 액세스할 수 있습니다. 또한 관계형 데이터 소스, CSV, JSON 및 Parquet.xiv와 같은 여러 지원 형식과 데이터 소스를 사용하여 그래프를 읽고 쓸 수 있는 DataFrame 데이터 소스를 완벽하게 지원합니다. 목록 6-3은 GraphFrames.xv를 사용하는 방법에 대한 예를 보여줍니다.

6장 그래프 분석

목록 6-3. GraphFrames 예제

```
spark-shell --package org.apache.spark.sql:spark-graphframes_2.11:0.7.0-spark2.4-s_2.11
```

org.graphframes를 가져옵니다._

```
val vDF = spark.createDataFrame(Array((1L, "산타 모니카", "CA"), (2L, "마리나 델 레이", "CA"), (3L, "글렌데일", "CA"), (4L, "패서디나", "CA"), (5L, "앵커리지", "AK"), (6L, "Fairbanks", "AK"), (7L, "Sterling", "AK"), (8L, "Wasilla", "AK"))).toDF("id", "city", "state")
```

vDF.show

아이디	시 주
1	산타모니카 캘리포니아
2	마리나 델 레이 캘리포니아
3	글렌데일 캘리포니아
4	패서디나 캘리포니아
5	앵커리지 악
6	페어뱅크스 악
7	스털링 악
8	와실라 악

```
발 eDF = spark.createDataFrame(배열((1L,2L,5),(1L,3L,24),(2L,3L,28),(3L,4L,9),(4L,3L,9),(5L,6L,32),(6L,7L,28),(5L,8L,17))).toDF("src", "dst", "거리")
```

eDF.show

src dest 거리
1 2 5
1 3 24
2 3 28
3 4 9
3 9 9

5	6	32
6 5	7	28
	8	17
+---+---+-----+		

```
val 그래프 = GraphFrame(vDF,eDF)
```

```
graph.vertices.show
```

+---+-----+-----+
아이디 시 주
+---+-----+-----+
1 산타모니카 캘리포니아
2 마리나 델 레이 캘리포니아
3 글렌데일 캘리포니아
4 패서디나 캘리포니아
5 앵커리지 악
6 페어뱅크스 악
7 스털링 악
8 와실라 악
+---+-----+-----+

```
graph.edges.show
```

+---+-----+-----+	
src dst 거리	
+---+-----+-----+	
1 2 1 5	
3 2 3 24	
3 4 4 3 28	
5 6 6 9	
7 5 8 9	
	32
	28
	17
+---+-----+-----+	

6장 그래프 분석

```
graph.triplets.show
```

src	가정자리	dst
[1, 산타모니카,...]	[1, 3, 24]	[3, 글렌데일, CA]
[1, 산타모니카,...]	[1, 2, 5]	[2, 마리나 델 레...]
[2, 마리나 델 레...]	[2, 3, 28]	[3, 글렌데일, CA]
[3, 글렌데일, CA]	[3, 4, 9]	[4, 캘리포니아 패서디나]
[4, 캘리포니아 패서디나]	[4, 3, 9]	[3, 글렌데일, CA]
[5, 앵커리지, AK]	[5, 8, 17]	[8, 와실라, AK]
[5, 앵커리지, AK]	[5, 6, 32]	[6, 페어뱅크스, AK]
[6, 페어뱅크스, AK]	[6, 7, 28]	[7, 스텔링, AK]

```
graph.inDegrees.show
```

아이디	정도
7	1
6	1
8	3
2	4
	1
	1

```
graph.outDegrees.show
```

아이디	아웃도
6	1
5	2
	2

```

| 3| | 1|
2| | 4| | 1|
           | 1|
+---+-----+
sc.setCheckpointDir("/tmp")

val 연결된 구성 요소 = graph.connectedComponents.run

ConnectedComponents.show
+---+-----+-----+
| 아이디| 시|주|구성요소|
+---+-----+-----+
| 1| 산타모니카| 캘리포니아| 1|
| 2|마리나 델 레이| 캘리포니아| 3|
          글렌데일| 캘리포니아| 1|
아| 4| 패서디나| 캘리포니아| 1|
아| 5| 앵커리지| 악| 6|
          페어뱅크스| 악| 7|
          스플링| 악| 8|
          와실라| 악| 5|
+---+-----+-----+

```

요약

이 장에서는 GraphX와 GraphFrames를 사용한 그래프 분석을 소개합니다. 그래프 분석은 광범위하고 광범위한 응용 분야에서 흥미롭고 빠르게 성장하는 연구 분야입니다. GraphX와 GraphFrames가 할 수 있는 일의 표면을 긁었습니다. 모든 사용 사례에 적합하지 않을 수 있지만 그래프 분석은 설계된 목적에 적합하며 분석 도구 세트에 없어서는 안될 추가 기능입니다.

6장 그래프 분석

참고문헌

- 나. 마이클 시몬스; "자신이 만든 억만장자 찰리 멍거가 다르게 행동하는 것", inc.com, 2019년, www.inc.com/michael-simmons/what-self-made-billionaire-charlie-munger-does-different.html
- ii. 캐롤 맥도날드; "Apache Spark 사용을 시작하는 방법
스칼라가 있는 GraphX," mapr.com, 2015, <https://mapr.com/blog/How-get-started-using-apache-spark-graphx-scala/>
- iii. 엔비디아; "그래프 분석", developer.nvidia.com, 2019년, <https://developer.nvidia.com/discover/graph-analytics>
- iv. 매스웍스; "유향 그래프와 무향 그래프", mathworks.com, 2019, www.mathworks.com/help/matlab/math/directed-and-undirected-graphs.html
- v. 리시 애다브; "제11장, GraphX를 이용한 그래프 처리
및 GraphFrames," Packt Publishing, 2017, Apache Spark 2.x 쿡북
- vi. 워커 로우; "그래프 데이터베이스의 사용 사례", bmc.com, 2019년, www.bmc.com/blogs/graph-database-use-cases/
- vii. 롭 페리; "데이터 계보: 총 GDPR 준수의 핵심" insidebigdata.com, 2018, <https://insidebigdata.com/2018/01/29/data-lineage-key-total-gdpr-compliance/>
- viii. Avery Ching et al.; "1조 에지: Facebook 규모의 그래프 처리", research.fb.com, 2015, <https://research.fb.com/publications/one-trillion-edges-graph-processing-at-facebook-scale/>
- ix. 모하메드 굴러; "Spark를 사용한 그래프 처리", Apress, 2015, Spark를 사용한 빅 데이터 분석
- 엑스. 불꽃; "속성 그래프", spark.apache.org, 2015, <https://spark.apache.org/docs/latest/graphx-programming-guide.html#속성-그래프>

- xi. 불꽃; "예제 속성 그래프", spark.apache.org, 2019,
<https://spark.apache.org/docs/latest/graphx-programming-guide.html#the-property-graph>
- xii. 불꽃; "Map Reduce Triplet Transition Guide", spark.apache.org, 2019, <https://spark.apache.org/docs/latest/graphx-programming-guide.html#the-property-graph>
- xiii. 캐롤 맥도날드; "Apache Spark 사용을 시작하는 방법
스칼라가 있는 GraphX," mapr.com, 2015, <https://mapr.com/blog/How-get-started-using-apache-spark-graphx-scala/>
- xiv. Ankur Dave et al.; "GraphFrame 소개", Databricks.com,
2016년, <https://databricks.com/blog/2016/03/03/graphframes.html> 소개
- xv. 리시 애다브; "제11장, GraphX를 이용한 그래프 처리
및 GraphFrames," Packt Publishing, 2017, Apache Spark 2.x 쿡북

7장

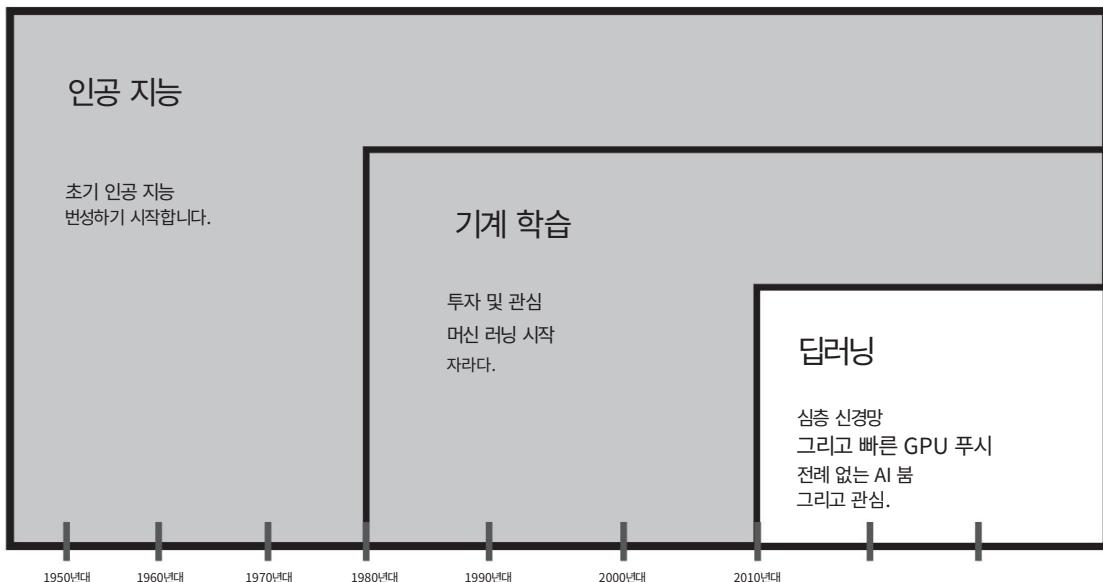
딥러닝

Minsky와 Papert가 Perceptron을 죽이기 보다는 이 문제에 대한 해결책을 제시했다면 더 많은 기여를 했을 것입니다.

—프란시스 크리키

딥 러닝은 심층 다층 인공 신경망을 사용하는 기계 학습 및 인공 지능의 하위 분야입니다(그림 7-1). 인간과 같은 작업을 탁월한 정확도로 수행할 수 있는 능력으로 인해 요즘 대세입니다. 딥 러닝 분야의 발전으로 컴퓨터 비전, 음성 인식, 게임, 이상 감지 등의 분야에서 흥미로운 가능성이 열렸습니다. GPU(그래픽 처리 장치)의 주류 가용성은 거의 하룻밤 사이에 전 세계적으로 인공 지능을 채택하게 되었습니다. 고성능 컴퓨팅은 불과 몇 년 전만 해도 계산이 불가능해 보였던 작업이 이제는 다중 GPU 클라우드 인스턴스나 저렴한 시스템 클러스터에서 일상적으로 수행될 정도로 강력해졌습니다. 이를 통해 인공 지능 분야의 수많은 혁신이 과거에는 불가능했던 속도로 발전할 수 있었습니다.

7장 딥러닝

그림 7-1. AI, 머신러닝, 딥러닝의 관계ⁱⁱ

딥 러닝은 최근 인공 지능의 많은 혁신을 주도하고 있습니다. 딥 러닝은 보다 일상적인 분류 작업에 사용 할 수 있지만 의료 진단, 얼굴 인식, 자율 주행 자동차, 사기 분석 및 지능형 음성 제어 비서와 같은 보다 복잡한 문제에 적용할 때 진정한 힘이 빛납니다.ⁱⁱⁱ 특정 영역에서, 딥 러닝은 컴퓨터가 인간의 능력과 일치하고 심지어 능가하는 것을 가능하게 했습니다. 예를 들어, 딥 러닝을 통해 Google DeepMind의 AlphaGo 프로그램은 지금까지 발명 된 가장 복잡한 보드 게임 중 하나인 바둑에서 한국의 마스터 이세돌을 물리칠 수 있었습니다. 바둑은 체스와 같은 다른 보드 게임보다 훨씬 더 복잡합니다. 그것은 우주에 있는 원자의 수보다 많은 170의 10의 거듭제곱 가능한 보드 구성을 가지고 있습니다.^{iv} 또 다른 좋은 예는 Tesla입니다. Tesla는 딥 러닝^v 를 사용 하여 자동 조종 기능을 강화하고 제조하는 모든 자동차에 내장된 여러 서라운드 카메라, 초음파 센서 및 전방 레이더의 데이터를 처리합니다. 딥 뉴럴 네트워크에 고성능 처리 기능을 제공하기 위해 Tesla는 GPU, CPU 및 초당 144조 작업(TOPS)을 제공할 수 있는 딥 러닝 가속기가 탑재된 자체 AI 칩을 사용하는 컴퓨터에 자율 주행 자동차를 장착합니다.).^{vi}

다른 기계 학습 알고리즘과 유사하게 심층 신경망은 일반적으로 더 많은 훈련 데이터에서 더 나은 성능을 보입니다. 이것이 Spark가 그림에 등장하는 곳입니다. 그러나 Spark MLlib에는 광범위한 기계 학습 알고리즘이 포함되어 있지만

이 글을 쓰는 시점에서 딥 러닝 지원은 여전히 제한적입니다(Spark MLlib에는 다층 패셉트론 분류기, 얇은 네트워크 훈련 및 전이 학습 수행으로 제한된 완전 연결된 피드포워드 신경망 포함). 그러나 Google 및 Facebook과 같은 개발자 및 회사 커뮤니티 덕분에 Spark와 통합되는 여러 타사 분산 딥 러닝 라이브러리 및 프레임워크가 있습니다. 가장 인기 있는 몇 가지를 살펴보겠습니다. 이 장에서는 가장 인기 있는 딥 러닝 프레임워크 중 하나인 Keras에 중점을 둘 것입니다. 분산 딥 러닝의 경우 Elephas와 Distributed Keras(Dist-Keras)를 사용합니다. 우리는 이 장 전체에서 파이썬을 사용할 것입니다. 우리는 분산 딥 러닝 예제에 PySpark를 사용할 것입니다.

신경망

신경망은 인간 두뇌의 상호 연결된 뉴런처럼 작동하는 알고리즘 클래스입니다. 신경망은 상호 연결된 노드로 구성된 여러 레이어를 포함합니다. 일반적으로 입력 계층, 하나 이상의 은닉 계층 및 출력 계층이 있습니다(그림 7-2). 데이터는 입력 레이어를 통해 신경망을 통과합니다. 은닉층은 가중치 연결 네트워크를 통해 데이터를 처리합니다. 은닉층의 노드는 입력에 가중치를 할당하고 그 과정에서 계수 세트와 결합합니다.

데이터는 계층의 출력을 결정하는 노드의 활성화 기능을 거칩니다. 마지막으로 데이터는 신경망의 최종 출력을 생성하는 출력 레이어에 도달합니다.vii

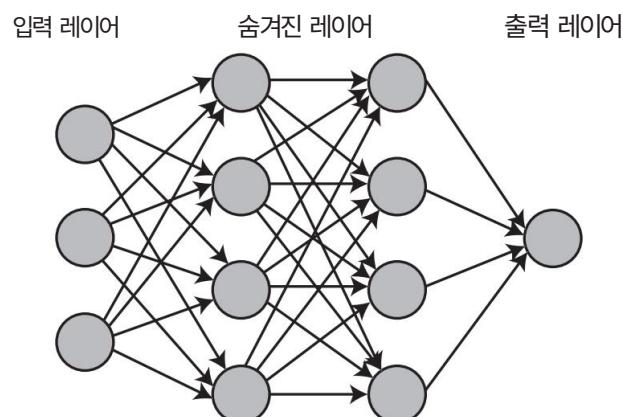


그림 7-2. 간단한 신경망

7장 딥러닝

여러 개의 은닉층이 있는 신경망을 "심층" 신경망이라고 합니다.

층이 많을수록 네트워크가 더 깊어지고 일반적으로 네트워크가 더 깊어질수록 학습이 더 정교해지고 해결할 수 있는 문제가 더 복잡해집니다. 이것이 심층 신경망이 최첨단 정확도를 달성할 수 있는 능력을 제공하는 것입니다. 신경망에는 여러 가지 유형이 있습니다. 피드포워드 신경망, 순환 신경망, 자동인코더 신경망은 각각 고유한 기능을 갖고 있으며 다른 목적을 위해 설계되었습니다. Feedforward 신경망 또는 다층 퍼셉트론은 구조화된 데이터와 잘 작동합니다. 순환 신경망은 오디오, 시계열 또는 텍스트와 같은 순차 데이터에 적합한 옵션입니다.^{viii} 자동인코더는 이상 감지 및 차원 감소에 적합한 선택입니다. 신경망의 한 유형인 합성곱 신경망은 최근 컴퓨터 비전 분야의 혁신적인 성능으로 인해 전 세계를 강타했습니다.

신경망의 간략한 역사

최초의 신경망인 McCulloch-Pitts Neuron은 1943년 Warren McCulloch와 Walter Pitts에 의해 개발되었으며 획기적인 논문 "A Logical Calculus of the Idea Immanent in Nervous Activity"에서 설명했습니다.^{ix} The McCulloch-Pitts Neuron은 현대의 신경망이지만 오늘날 우리가 알고 있는 인공 지능의 탄생을 위한 길을 밟은 씨앗이었습니다. 1958년 Frank Rosenblatt에 의해 소개된 퍼셉트론은 신경망 분야에서 차세대 혁명적인 발전이었습니다. Rosenblatt Perceptron은 이진 분류에 사용되는 간단한 단일 계층 신경망입니다. 이것은 Cornell Aeronautical Laboratory에서 처음에 IBM 704를 사용하여 소프트웨어로 시뮬레이션되었습니다. Rosenblatt의 작업은 이미지 인식을 위해 설계된 맞춤형 기계인 Mark I Perceptron의 제작으로 절정에 달했습니다.

Perceptron에 대한 관심은 1960년대 후반에 Marvin Minsky와 Seymour Papert가 Perceptron의 한계, 특히 비선형 함수를 배울 수 없는 능력에 대해 설명한 매우 영향력 있는 책인 Perceptrons를 출판한 후 줄어들기 시작했습니다. 나중에 몇 년 후에 인정할 것 상황은 부분적으로 지원 벡터 기계(SVM) 및 그래픽 모델의 성공과 상징적 인공 지능의 인기 증가, 1950년대부터 1980년대 말까지 널리 퍼진 AI 패러다임으로 인해 더욱 악화되었습니다.^{xii} 그럼에도 불구하고 많은 Jim Anderson, David Willshaw 및

7장 딥러닝

Stephen Grossberg는 조용히 신경망에 대한 연구를 계속했습니다. 1975년, 후쿠시마 쿠니히코는 일본 도쿄 세가타야에 있는 NHK 과학 기술 연구소의 연구원으로서 최초의 다층 신경망인 신인지자(neocognitron)를 개발했습니다.^{xiii}

신경망은 1980년대 중반에 부활하여 현대 딥 러닝의 중심이 되는 많은 기본 개념과 기술이 많은 개척자들에 의해 발명되었습니다. 1986년에 Geoffrey Hinton(많은 사람들이 "AI의 대부"로 간주)은 신경망이 학습하는 기본 방법인 역전파를 대중화했으며 거의 모든 현대 신경망 구현의 중심이 되었습니다.^{xiv} 마찬가지로 중요하게 Hinton과 그의 팀은 이 아이디어를 대중화했습니다. Perceptrons.^{xv}에서 Minsky와 Papert의 비판을 직접 다루면서 비선형 함수를 학습하기 위해 다층 신경망을 사용하는 것 Hinton의 작업은 신경망을 모호한 상태에서 다시 한 번 주목받는 데 중요한 역할을 했습니다.

후쿠시마의 신인지자에서 영감을 받은 Yann LeCun은 1980년대 후반에 합성곱 신경망을 개발했습니다. 1988년 AT&T Bell Labs에서 Yann LeCun은 손으로 쓴 문자를 인식하기 위해 합성곱 신경망을 사용했습니다. AT&T는 수표의 손글씨를 읽고 인식하기 위해 시스템을 은행에 판매했으며 신경망의 실제 응용 프로그램 중 첫 번째로 간주됩니다.^{xvi} Yoshua Bengio는 단어 임베딩 개념의 도입과 Ian과의 최근 작업으로 유명합니다. GAN(Generative Adversarial Networks)에 대한 Goodfellow는 서로 대립하거나 "적대적인" 두 개의 신경망으로 구성된 심층 신경망 아키텍처 유형입니다.

적대적 훈련은 Yann LeCun에 의해 "머신 러닝 분야에서 지난 10년 동안 가장 흥미로운 아이디어"라고 설명했습니다.^{xvii} 세 명의 개척자는 최근 2018년 Turing Award를 수상했습니다.

그럼에도 불구하고 이러한 혁신에도 불구하고 신경망의 실제 사용은 여전히 남아있었습니다. 모든 사람의 손이 닿지 않는 곳에 있지만 가장 큰 기업과 자금이 넉넉한 대학을 교육하는 데 필요한 상당한 양의 계산 리소스로 인해 신경망을 주류로 돌아넣은 것은 저렴한 고성능 GPU의 광범위한 가용성이었습니다. 이전에는 확장할 수 없고 훈련이 불가능하다고 생각했던 신경망은 갑자기 GPU를 사용하여 컴퓨터 비전, 음성 인식, 게임 및 기타 역사적으로 어려운 기계 학습 작업 분야에서 인간과 같은 성능을 약속하는 것을 제공하기 시작했습니다. Alex Krizhevsky, Ilya Sutskever 및 Geoff Hinton의 GPU 훈련된 컨볼루션 신경망 AlexNet이 2012년 ImageNet Large Scale Visual Recognition 대회에서 우승했을 때 세계가 주목했습니다.^{xviii} [이전](#)에 우승한 다른 GPU 훈련 CNN이 있었습니다.

7장 딥러닝

하지만 모두의 관심을 끈 것은 AlexNet의 기록적인 정확도였습니다. 2위보다 10.8% 더 낮은 15.3%의 상위 5개 오류를 달성하여 큰 차이로 대회에서 우승했습니다.^{xix} AlexNet은 또한 다음과 같은 대부분의 CNN 아키텍처에서 현재 일반적으로 사용되는 원칙 대부분을 대중화했습니다. ReLU(Rectified Linear Unit) 활성화 기능 사용, 과적합, 중복 풀링 및 여러 GPU에 대한 교육을 줄이기 위한 드롭아웃 및 데이터 증대 사용.

최근 학계와 민간 부문 모두에서 AI 분야에서 전례 없는 양의 혁신이 이루어지고 있습니다. Google, Amazon, Microsoft, Facebook, IBM, Tesla, Nvidia 등과 같은 회사는 딥 러닝에 상당한 리소스를 투자하여 인공 지능의 경계를 더욱 넓히고 있습니다. 스탠포드대, 카네기멜론대, MIT 등 유수의 대학과 OpenAI, 알렌인공지능연구소 등 연구기관이 놀라운 속도로 획기적인 연구를 발표하고 있다. 전 세계적으로 인상적인 혁신이 중국, 캐나다, 영국 및 호주와 같은 국가에서 나오고 있습니다.

컨볼루션 신경망

합성곱 신경망(줄여서 convnet 또는 CNN)은 이미지 분석에 특히 뛰어난 신경망 유형입니다(오디오 및 텍스트 데 이터에도 적용할 수 있음). 컨볼루션 신경망 계층의 뉴런은 높이, 너비 및 깊이의 3차원으로 배열됩니다. CNN은 합성곱 계층을 사용하여 텍스처 및 가장자리와 같은 입력 특징 공간(이미지)의 로컬 패턴을 학습합니다. 대조적으로, 완전히 연결된(조밀한) 계층은 전역 패턴을 학습 합니다.^{xx} 컨볼루션 계층의 뉴런은 조밀한 계층의 경우처럼 모든 뉴런 대신에 선행 계층의 작은 영역에만 연결됩니다. 조밀한 레이어의 완전히 연결된 구조는 비효율적이고 빠르게 과적합으로 이어질 수 있는 매우 많은 수의 매개변수로 이어질 수 있습니다.^{xxi}

컨볼루션 신경망에 대한 논의를 계속하기 전에,

색상 모델의 개념을 이해하는 데 중요합니다. 이미지는 픽셀 모음으로 표시됩니다.^{xxii} 픽셀은 이미지를 구성하는 정보의 가장 작은 단위입니다. 회색조 이미지의 픽셀 값 범위는 [그림 7-3\(a\)](#) 와 같이 0(검은색)에서 255(흰색)까지입니다. 회색조 이미지에는 하나의 채널만 있습니다. RGB는 가장 전통적인 색상 모델입니다. RGB 이미지에는 빨강, 녹색 및 파랑의 세 가지 채널이 있습니다. 이것은 RGB 이미지의 각 픽셀이 각각 0에서 255 사이의 3개의 8비트 숫자 ([그림 7-3\(b\)](#))로 표시된다는 것을 의미 합니다. 이러한 채널의 다른 조합을 사용하면 다른 색상이 표시됩니다.

7장 딥러닝

225	165	163	168	145	145	225	168
215	215	163	143	143	220	210	225
230	220	228	154	144	154	123	123
154	128	128	154	125	120	154	125
180	175	225	130	130	120	148	143
215	235	230	235	215	125	215	135
154	154	145	155	215	215	145	225

(a)

225	165	163	168	145	145	225	168		
2	122	243	115	115	110	245	211	211	
23	13	214	189	165	122	158	151	134	220
15	23	134	215	163	143	143	220	210	225
18	15	230	220	228	154	144	154	123	123
2	18	154	128	128	154	125	120	154	125
15	2	180	175	225	130	130	120	148	143
15	215	235	230	235	215	125	215	135	
	154	154	145	155	215	215	145	225	

(b)

그림 7-3. 회색조 및 RGB 이미지

컨볼루션 신경망은 이미지 높이, 무게 및 깊이와 같은 형태의 3차원 텐서를 입력으로 받습니다(그림 7-4). 이미지의 깊이는 채널 수입니다. 회색조 이미지의 경우 깊이는 1이고 RGB 이미지의 경우 깊이는 3입니다. 그림 7-4에서 이미지의 입력 모양은 (7, 8, 3)입니다.

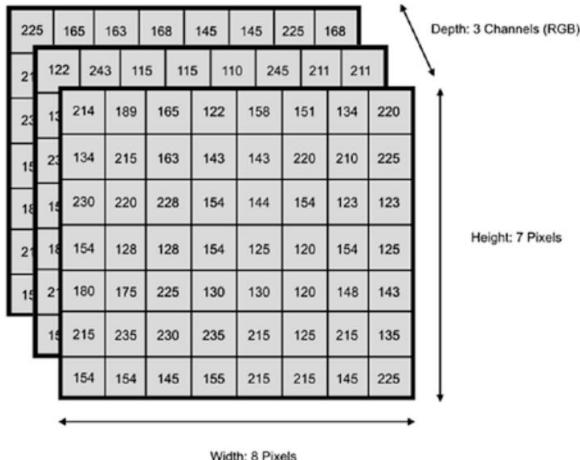


그림 7-4. 개별 이미지의 입력 모양

7장 딥러닝

컨볼루션 신경망 아키텍처

그림 7-5 는 일반적인 CNN 아키텍처가 어떻게 생겼는지 보여줍니다. 컨볼루션 신경망은 여러 계층으로 구성되며 각각은 다양한 기능을 식별하고 학습하려고 합니다.

계층의 주요 유형은 컨볼루션 계층, 풀링 계층 및 완전 연결 계층입니다.

계층은 특징 감지 계층과 분류 계층의 두 가지 주요 범주로 더 분류할 수 있습니다.

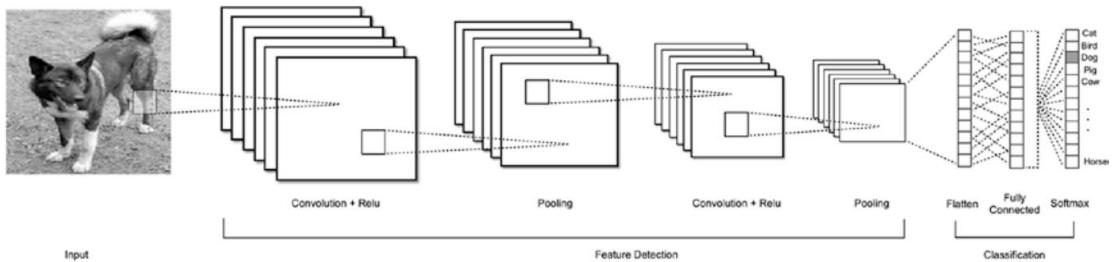


그림 7-5. 동물 이미지를 분류하기 위한 CNN 아키텍처^{xxiii}

특징 감지 계층

컨볼루션 레이어

컨볼루션 계층은 컨볼루션 신경망의 기본 빌딩 블록입니다.

컨볼루션 계층은 입력 이미지의 특정 기능을 활성화하는 일련의 컨볼루션 커널 또는 필터를 통해 입력 이미지를 실행합니다. 컨볼루션은 커널이 입력 기능 맵을 가로질러 미끄러질 때 입력 요소와 커널 요소가 겹치는 각 위치에서 요소별 곱셈을 수행하는 수학 연산입니다. 결과가 추가되어 현재 위치에 출력이 생성됩니다. 이 프로세스는 모든 보폭에 대해 반복되어 출력 특성 [map.xxiv](#)라는 최종 결과를 생성합니다. 그림 7-6은 3×3 컨볼루션 커널을 보여줍니다. 그림 7-7은 3×3 커널이 5×5 입력 기능 맵에 적용되어 3×3 출력 기능 맵이 생성되는 2D 컨볼루션을 보여줍니다. 우리는 stride 1을 사용하는데, 이것은 커널이 현재 위치에서 다음 위치로 한 픽셀을 이동함을 의미합니다.

2	0	1
1	2	0
2	2	1

그림 7-6. 3×3 컨볼루션 커널

2	3	1	3	0
0	1	1	3	1
3	1	2	2	3
2	0	2	0	2
2	0	0	0	1

(a)

17		

2	3	1	3	0
0	1	1	3	1
3	1	2	2	3
2	0	2	0	2
2	0	0	0	1

(b)

17	20	

2	3	1	3	0
0	1	1	3	1
3	1	2	2	3
2	0	2	0	2
2	0	0	0	1

(c)

17	20	20

2	3	1	3	0
0	1	1	3	1
3	1	2	2	3
2	0	2	0	2
2	0	0	0	1

(d)

17	20	20

2	3	1	3	0
0	1	1	3	1
3	1	2	2	3
2	0	2	0	2
2	0	0	0	1

(e)

17	20	20

2	3	1	3	0
0	1	1	3	1
3	1	2	2	3
2	0	2	0	2
2	0	0	0	1

(f)

17	20	20

2	3	1	3	0
0	1	1	3	1
3	1	2	2	3
2	0	2	0	2
2	0	0	0	1

(g)

17	20	20

2	3	1	3	0
0	1	1	3	1
3	1	2	2	3
2	0	2	0	2
2	0	0	0	1

(h)

17	20	20

2	3	1	3	0
0	1	1	3	1
3	1	2	2	3
2	0	2	0	2
2	0	0	0	1

(i)

17	20	20

그림 7-7. 3×3 커널이 5×5 입력 기능 맵(25개 기능)에 적용되어 9개의 출력 기능이 있는 3×3 출력 기능 맵이 생성되는 2D 컨볼루션

RGB 이미지의 경우 3D 컨볼루션에는 각 입력 채널에 대한 커널이 포함됩니다(총 3개의 커널). 결과가 모두 추가되어 하나의 출력을 형성합니다. 활성화 함수가 데이터에 더 잘 맞도록 하는 데 사용되는 편향 항이 추가되어 최종 출력 기능 맵을 생성합니다.

7장 딥러닝

ReLU(Rectified Linear Unit) 활성화 기능

각 컨볼루션 레이어 뒤에 활성화 레이어를 포함하는 것이 일반적입니다. 활성화 함수는 모든 순방향 계층에서 지원하는 활성화 인수를 통해 지정할 수도 있습니다. 활성화 함수는 노드의 입력을 다음 계층의 입력으로 사용되는 출력 신호로 변환합니다. 활성화 기능은 이미지, 오디오 및 텍스트와 같은 더 복잡한 데이터를 학습할 수 있도록 하여 신경망을 더욱 강력하게 만듭니다. 이것은 우리의 신경망에 비선형 속성을 도입하여 수행합니다. 활성화 함수가 없으면 신경망은 단순한 선형 문제만 처리할 수 있는 지나치게 복잡한 선형 회귀 모델이 됩니다. xv Sigmoid 및 tanh도 일반적인 활성화 계층입니다. ReLU(Rectified Linear Unit)는 대부분의 CNN에서 가장 인기 있고 선호되는 활성화 함수입니다. ReLU는 수신한 모든 양수 입력에 대한 값을 반환하지만 음수 값을 받으면 0을 반환합니다. ReLU는 정확도에 큰 영향을 미치지 않으면서 모델 교육을 가속화하는 데 도움이 되는 것으로 밝혀졌습니다.

풀링 레이어

풀링 레이어는 입력 이미지의 차원을 축소하여 계산 복잡성과 매개변수 수를 줄입니다. 차원을 줄임으로써 풀링 레이어는 과적합을 제어하는 데도 도움이 됩니다. 또한 모든 컨볼루션 레이어 뒤에 풀링 레이어를 삽입하는 것이 일반적입니다. 풀링에는 평균 풀링과 최대 풀링의 두 가지 주요 종류가 있습니다. Average pooling은 각 pooling 영역의 모든 값의 평균을 사용하고 (Figure 7-8(b)), max pooling은 최대값을 사용합니다 (Figure 7-8(a)). 일부 아키텍처에서는 풀링 레이어를 사용하는 것보다 더 큰 스트라이드를 사용하는 것이 좋습니다.

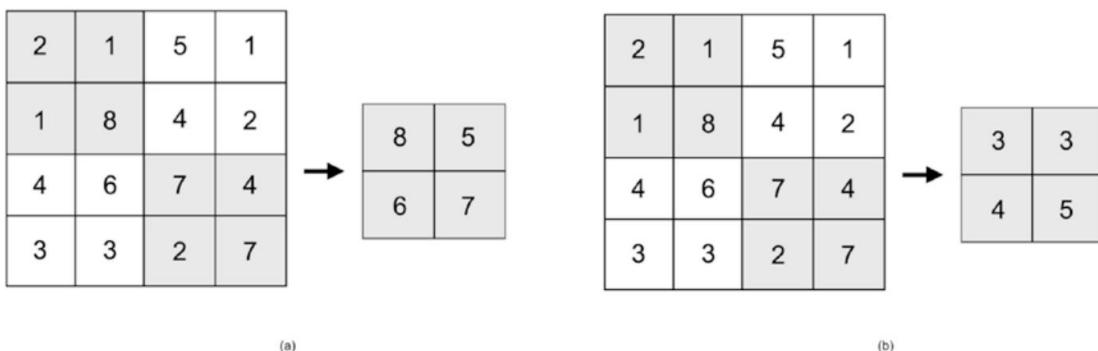


그림 7-8. 최대 및 평균 풀링

분류 레이어

레이어 병합

평면화 계층은 데이터가 완전 연결 밀집 계층에 공급되기 전에 2차원 행렬을 1차원 벡터로 변환합니다.

완전 연결(밀도) 계층

조밀한 계층이라고도 하는 완전 연결 계층은 평평한 계층에서 데이터를 수신하고 각 클래스에 대한 확률을 포함하는 벡터를 출력합니다.

드롭아웃 레이어

드롭아웃 계층은 훈련 중에 네트워크의 일부 뉴런을 무작위로 비활성화합니다. 드롭아웃은 모델 복잡성을 줄이고 과적합을 방지하는 데 도움이 되는 정규화의 한 형태입니다.

소프트맥스와 시그모이드 함수

최종 조밀 계층은 분류 출력을 제공합니다. 다중 클래스 분류 작업에 softmax 함수를 사용하거나 이진 분류 작업에 sigmoid 함수를 사용합니다.

딥 러닝 프레임워크

이 섹션에서는 현재 사용 가능한 가장 인기 있는 딥 러닝 프레임워크에 대한 간략한 개요를 제공합니다. 이 것은 결코 완전한 목록이 아닙니다. 이러한 프레임워크의 대부분은 거의 동일한 기능을 가지고 있습니다. 커뮤니티 및 업계 채택 수준, 인기도 및 생태계 규모가 다릅니다.

텐서플로우

TensorFlow는 현재 가장 널리 사용되는 딥 러닝 프레임워크입니다. 구글에서 아노를 대체하기 위해 개발했다. Theano의 원래 개발자 중 일부는 Google에 가서 TensorFlow를 개발했습니다. TensorFlow는 C/C++로 개발된 엔진 위에서 실행되는 Python API를 제공합니다.

7장 딥러닝

테아노

ano는 최초의 오픈 소스 딥 러닝 프레임워크 중 하나입니다. 몬트리올 대학의 MILA(Montreal Institute for Learning Algorithms)에서 2007년에 처음 발표했습니다. 2017년 9월 Yoshua Bengio는 공식적으로ano의 개발이 종료될 것이라고 발표했습니다 .xxvi

파이토치

PyTorch는 Facebook 인공 지능 연구(FAIR) 그룹에서 개발한 오픈 소스 딥 러닝 라이브러리입니다. PyTorch 사용자 채택은 최근 급증했으며 TensorFlow 및 Keras 다음으로 세 번째로 인기 있는 프레임워크입니다.

딥러닝4J

DeepLearning4J는 Java로 작성된 딥 러닝 라이브러리입니다. Scala와 같은 JVM 기반 언어와 호환되며 Spark 및 Hadoop과 통합됩니다. Breeze 대신 자체 오픈 소스 과학 컴퓨팅 라이브러리인 ND4J를 사용합니다. DeepLearning4J는 딥 러닝을 위해 Python보다 Java 또는 Scala를 선호하는 개발자에게 좋은 옵션입니다(DeepLearning4J에는 Keras를 사용하는 Python API가 있음).

CNTK

Microsoft Cognitive Toolkit이라고도 하는 CNTK는 Microsoft Research에서 개발하고 2015년 4월에 오픈 소스로 제공한 딥 러닝 라이브러리입니다. 신경망을 설명하기 위해 방향성 그래프를 사용하는 일련의 계산 단계를 사용합니다. CNTK는 여러 GPU와 서버에서 분산 딥 러닝을 지원합니다.

케라스

Keras는 Francois Chollet이 Google에서 개발한 고급 딥 러닝 프레임워크입니다. TensorFlow, Theano 및 CNTK 위에서 실행할 수 있는 간단한 모듈식 API를 제공합니다. 그것은 광범위한 산업 및 커뮤니티 채택과 활기찬 생태계를 즐깁니다.
Keras 모델은 Keras를 통해 브라우저에서 iOS 및 Android 장치에 배포할 수 있습니다.
Google Cloud에서 DL4J 가져오기 기능을 통해 JVM에서 Node.js 및 WebDNN

Skymind는 물론 Raspberry Pi에서도 마찬가지입니다. Keras 개발은 주로 Google의 지원을 받지만 Microsoft, Amazon, Nvidia, Uber 및 Apple도 주요 프로젝트 기여자로 간주합니다. 다른 프레임워크와 마찬가지로 뛰어난 다중 GPU 지원을 제공합니다. 가장 복잡한 심층 신경망의 경우 Keras는 Horovod, Google Cloud의 GPU 클러스터, Dist-Keras 및 Elephas를 통한 Spark를 통한 분산 교육을 지원합니다.

Keras를 사용한 딥 러닝

모든 딥 러닝 예제의 백엔드로 TensorFlow와 함께 Keras Python API를 사용할 것입니다. 이 장의 뒷부분에서 분산 딥 러닝 예제를 위해 Elephas 및 Distributed Keras와 함께 PySpark를 사용할 것입니다.

홍채 데이터 세트를 사용한 다중 클래스 분류

첫 번째 신경망 분류xxvii 작업(목록 7-1)에 친숙한 데이터 세트를 사용할 것입니다.

앞서 논의한 바와 같이 이 장의 모든 예제에 Python을 사용할 것입니다.

목록 7-1. Iris 데이터 세트를 사용하여 Keras를 사용한 다중 클래스 분류

numpy를 np로 가져오기

keras.models에서 가져오기 순차

keras.optimizers에서 Adam 가져오기

keras.layers에서 Dense 가져오기

sklearn.model_selection import train_test_split에서

sklearn.preprocessing 가져오기 OneHotEncoder에서

sklearn.datasets에서 load_iris 가져오기

```
# Iris 데이터셋을 로드합니다.
```

```
iris_data = load_iris()
```

```
x = iris_data.data
```

```
# 데이터를 검사합니다. 우리는 잘릴 것입니다
```

```
# 간결함을 위한 출력.
```

7장 딥러닝

```
print(x)
[[5.1 3.5 1.4 0.2] [4.9
 3. 1.4 0.2] [4.7 3.2
 1.3 0.2] [4.6 3.1 1.5
 0.2] [5. 3.6 1.4 0.2]
[5.4 3.9 1.7 0.4] [4.6
 3.4 1.4 0.3] [5. 3.4
 1.5 0.2] [4.4 2.9 1.4
 0.2] [4.9 3.1 1.5 0.1]]]
```

클래스 레이블을 단일 열로 변환합니다. y =

```
iris_data.target.reshape(-1, 1)
```

클래스 레이블을 검사합니다. 간결함을 위해 # 출력을 자릅니다.

인쇄(y)

```
[[0] [0]
 [0] [0]
 [0] [0]
 [0] [0]
 [0] [0]]]
```

다중 클래스 분류 작업에 신경망을 사용할 때 클래스 레이블을 원-핫 인코딩하는 것이 # 모범 사례로 간주됩니다.

```
인코더 = OneHotEncoder(sparse=False)
enc_y = 인코더.fit_transform(y) print(enc_y)
```

```
[[1. 0. 0.] [1. 0.  
0.] [1. 0. 0.]  
[1. 0. 0.] [1.  
0. 0.] [1. 0.  
0.] [1. 0. 0.]  
[1. 0. 0.] [1.  
0. 0.] [1. 0.  
0.]]
```

훈련 및 테스트를 위해 데이터를 분할합니다. # 훈련 데이터셋의 경우
우 70%, 테스트 데이터셋의 경우 30%.

```
train_x, test_x, train_y, test_y = train_test_split(x, enc_y, test_size=0.30)  
# 모델을 정의합니다.
```

선형 레이어 스택으로 구성된 Sequential 모델을 인스턴스화합니다.
우리는 1차원 특징 벡터를 다루기 때문에 네트워크에서 # 조밀한 레이어를 사용할 것입니다. 이미지로 작업할 때 # 장
의 뒷부분에서 컨볼루션 레이어를 사용합니다.

완전히 연결된 레이어에서 활성화 함수로 ReLU를 사용합니다.
name 인수를 전달하여 모든 레이어의 이름을 지정할 수 있습니다. 4인 input_shape 인수에 기능의 수 #를 전달
합니다(petal_length, # feather_width, sepal_length, sepal_width).

```
model = Sequential()  
model.add(Dense(10, input_shape=(4,), activation='relu', name='fclayer1')) model.add(Dense(10,  
activation='relu', name=' fclayer2'))
```

우리는 출력 레이어에서 softmax 활성화 함수를 사용합니다. 논의한 바와 같이 # softmax 활성화 계층을 사용
하면 모델이 다중 클래스 분류를 수행할 수 있습니다. 이진 분류의 경우 대신 시그모이드 활성화 # 함수를 사용합니
다. 우리는 클래스 수를 지정합니다. 이 경우에는 3입니다.

7장 딥러닝

```
model.add(Dense(3, 활성화='softmax', 이름='출력'))
```

모델을 컴파일합니다. 다중 클래스에 categorical_crossentropy를 사용합니다.

분류. 이진 분류의 경우 binary_crossentropy를 사용합니다.

```
model.compile(loss='categorical_crossentropy', 옵티마이저='adam', 메트릭스=['accuracy'])
```

모델 요약을 표시합니다.

```
인쇄(모델.요약())
```

모델: "sequential_1"

레이어(유형)	출력 형태	매개변수 #
fclayer1(고밀도)	(없음, 10)	50
fclayer2(고밀도)	(없음, 10)	110
출력(밀도)	(없음, 3)	33

총 매개변수: 193

훈련 가능한 매개변수: 193

훈련할 수 없는 매개변수: 0

없음

Epoch가 100으로 설정된 훈련 데이터 세트에서 모델 훈련

및 배치 크기를 5로 설정합니다. 출력은 간결함을 위해 편집됩니다.

```
model.fit(train_x, train_y, verbose=2, batch_size=5, epochs=100)
```

에포크 1/100

- 2초 - 손실: 1.3349 - acc: 0.3333

신기원 2/100

- 0초 - 손실: 1.1220 - acc: 0.2952

신기원 3/100

- 0초 - 손실: 1.0706 - 기준: 0.3429

```

에포크 4/100
- 0초 - 손실: 1.0511 - 기준: 0.3810

에포크 5/100
- 0초 - 손실: 1.0353 - 기준: 0.3810

에포크 6/100
- 0초 - 손실: 1.0175 - 기준: 0.3810

신기원 7/100
- 0초 - 손실: 1.0013 - acc: 0.4000

에포크 8/100
- 0초 - 손실: 0.9807 - 기준: 0.4857

신기원 9/100
- 0초 - 손실: 0.9614 - 기준: 0.6667

Epoch 10/100 -
0초 - 손실: 0.9322 - 기준: 0.6857

...
Epoch 97/100 -
0초 - 손실: 0.1510 - 기준: 0.9524

Epoch 98/100 -
0초 - 손실: 0.1461 - 기준: 0.9810

Epoch 99/100 -
0초 - 손실: 0.1423 - 기준: 0.9810

Epoch 100/100 -
0초 - 손실: 0.1447 - 기준: 0.9810

<keras.callbacks.History 객체 0x7fbb93a50510>

# 테스트 데이터셋에 대한 테스트.

결과 = model.evaluate(test_x, test_y) 45/45
[=====] - 0초 586us/단계

print('정확도: {:.4f}'.format(결과[1]))
정확도: 0.933333

```

재미있었어요. 그러나 신경망은 더 복잡한 분야에서 사용될 때 정말 빛을 발합니다.
 이미지와 같은 비정형 데이터와 관련된 문제. 다음 예제에서는 손으로 쓴 숫자 인식을 수행하기 위해 컨볼루션 신경
 망을 사용할 것입니다.

7장 딥러닝

MNIST로 필기 숫자 인식

MNIST 데이터 세트는 미국 NIST(National Institute of Standards and Technology)의 손으로 쓴 숫자 이미지 데이터베이스입니다. 데이터 세트에는 70,000개의 이미지가 포함되어 있습니다. 60,000개의 학습용 이미지와 10,000개의 테스트용 이미지가 있습니다. 그들은 모두 0에서 9까지의 28 x 28 그레이스케일 이미지입니다(그림 7-9). 목록 [7-2를 참조 하십시오.](#)

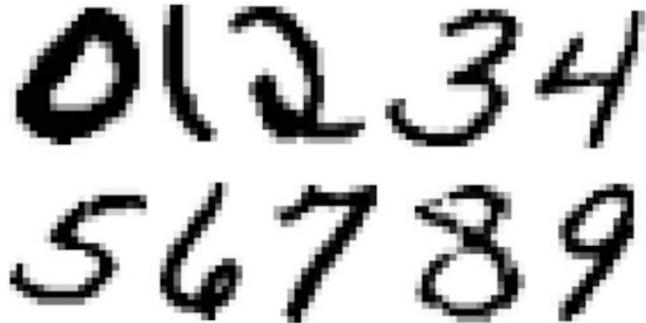


그림 7-9. MNIST 손으로 쓴 숫자 데이터베이스의 샘플 이미지

목록 7-2. Keras를 사용한 MNIST로 필기 숫자 인식

keras 가져오기

matplotlib.pyplot을 plt로 가져오기

keras.datasets에서 mnist 가져오기 keras.models
에서 가져오기 keras.layers에서 순차 가져오기 Dense,
Conv2D, Dropout, Flatten, MaxPooling2D 가져오기

MNIST 데이터를 다운로드합니다.

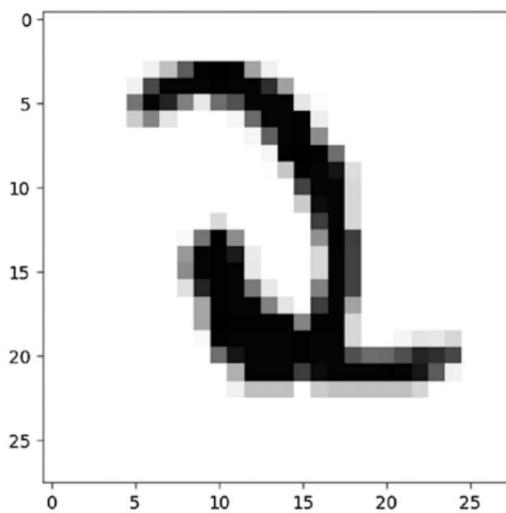
```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

image_idx = 400
print(y_train[image_idx])

# 데이터를 검사합니다.

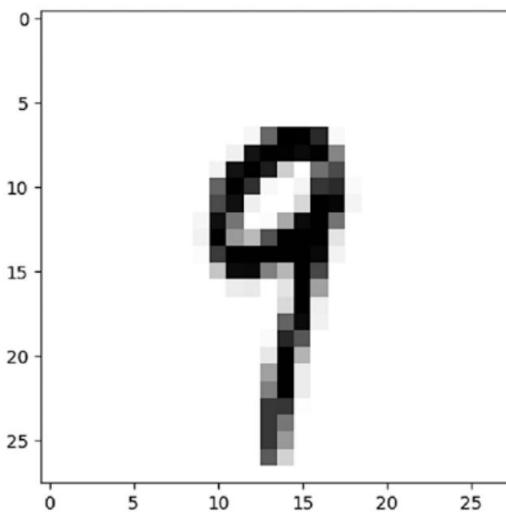
plt.imshow(x_train[image_idx], cmap='회색')
plt.show()
```

7장 딥러닝



다른 번호를 확인하세요.

```
image_idx = 600  
print(y_train[0:image_idx])  
9  
  
plt.imshow(x_train[image_idx], cmap='회색') plt.show()
```



7장 딥러닝

```
# 데이터 세트의 "모양"을 가져옵니다. x_train.shape  
(60000, 28, 28)
```

Keras와 함께 작동할 수 있도록 배열을 4차원으로 재구성해 보겠습니다.

이미지는 28 x 28 그레이스케일입니다. 마지막 매개변수 1은 # 이미지가 회색조임을 나타냅니다. RGB의 경우
매개변수를 3으로 설정합니다.

```
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1) x_test =  
x_test.reshape(x_test.shape[0], 28, 28, 1)
```

나누기 전에 값을 부동 소수점으로 변환합니다.

```
x_train = x_train.astype('float32') x_test =  
x_test.astype('float32')
```

RGB 코드를 정규화합니다.

```
x_train /= 255  
x_test /= 255  
print'x_train 모양: ', x_train.shape x_train 모양:  
60000, 28, 28, 1
```

인쇄' 아니요. 훈련 데이터 세트의 이미지 수: ', x_train.shape[0]
훈련 데이터 세트의 이미지 수: 60000

인쇄' 아니요. 테스트 데이터 세트의 이미지 수: ', x_test.shape[0]
테스트 데이터 세트의 이미지 수: 10000

클래스 벡터를 이진 클래스 행렬로 변환합니다. # 클래스 수(10)도 전달합니다.

```
y_train = keras.utils.to_categorical(y_train, 10) y_test =  
keras.utils.to_categorical(y_test, 10)
```

CNN을 구축합니다. Sequential 모델을 만들고 레이어를 추가합니다.

```
모델 = Sequential()  
model.add(Conv2D(28, kernel_size=(3,3), input_shape= (28,28,1), name='convlayer1'))
```

최대 풀링을 사용하여 차원을 줄입니다.

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```

다음으로 2차원 배열을 1차원 특징 벡터로 평면화해야 합니다. 이를 통해 분류를 수행할 수 있습니다.

```
model.add(Flatten())
```

```
model.add(Dense(128, activation='relu', name='fclayer1'))
```

데이터를 분류하기 전에 드롭아웃 레이어를 사용하여 # 일부 뉴런을 무작위로 비활성화합니다. Dropout은
모델 복잡성을 줄이고 과적합을 방지하는 데 사용되는 일종의 정규화입니다.

```
model.add(Dropout(0.2))
```

최종 레이어를 추가합니다. softmax 계층(다항 로지스틱 # 회귀)은 총 클래스 수를 매개변수로 가져야 합니다.

이 경우 매개변수의 수는 10인 자릿수(0~9)입니다.

```
model.add(Dense(10, activation='softmax', name='output'))
```

모델을 컴파일합니다.

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

모델을 훈련시킵니다.

```
model.fit(x_train, y_train, batch_size=128, verbose=1, epochs=20)
```

Epoch 1/20

60000/60000 [=====] - 11초 180us/단계 - 손실:

0.2742 - 기준: 0.9195

Epoch 2/20

60000/60000 [=====] - 9초 144us/단계 - 손실: 0.1060 - acc :

0.9682 Epoch 3/20 60000/60000 [=====] - 9초 143us/단계 - 손실:

0.0731 - acc: 0.9781 Epoch 4/20

7장 딥러닝

60000/60000 [=====] - 9초 144us/step - 손실: 0.0541 - acc: 0.9830

에포크 5/20

60000/60000 [=====] - 9s 143us/step - 손실: 0.0409 - acc: 0.9877

에포크 6/20

60000/60000 [=====] - 9s 144us/step - 손실: 0.0337 - acc: 0.9894

신기원 7/20

60000/60000 [=====] - 9초 143us/단계 - 손실:
0.0279 - 기준: 0.9910

에포크 8/20

60000/60000 [=====] - 9s 144us/step - 손실: 0.0236 - acc: 0.9922

신기원 9/20

60000/60000 [=====] - 9초 143us/단계 - 손실:
0.0200 - 기준: 0.9935

신기원 10/20

60000/60000 [=====] - 9초 144us/단계 - 손실:
0.0173 - 기준: 0.9940

신기원 11/20

60000/60000 [=====] - 9s 143us/step - 손실: 0.0163 - acc: 0.9945

신기원 12/20

60000/60000 [=====] - 9s 143us/step - 손실: 0.0125 - acc: 0.9961

신기원 13/20

60000/60000 [=====] - 9s 143us/step - 손실: 0.0129 - acc: 0.9956

신기원 14/20

60000/60000 [=====] - 9s 144us/step - 손실: 0.0125 - acc: 0.9958

신기원 15/20

```
60000/60000 [=====] - 9초 144us/step - 손실: 0.0102 - acc: 0.9968
Epoch 16 /20 60000/60000 [=====] - 9초 143us/단계 - 손실: 0.0101
- 기준: 0.9964 Epoch 17/20 60000/60000 [=====] - 9초 143us/단계
- 손실: 0.0096 - acc : 0.9969 Epoch 18/20 60000/60000 [=====]
- 9초 143us/단계 - 손실: 0.0096 - acc: 0.9968 Epoch 19/20 60000/60000
[=====] - 9초 142us/단계 - 손실 :
```

0.0090 - 기준: 0.9972

Epoch 20/20

```
60000/60000 [=====] - 9초 144us/단계 - 손실: 0.0097 - acc : 0.9966
<keras.callbacks.History 개체 0x7fc63d629850>
```

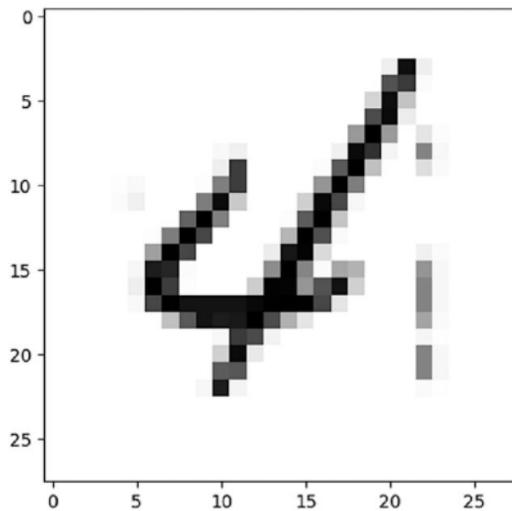
모델을 평가합니다.

```
evalscore = model.evaluate(x_test, y_test, verbose=0) print'테스트 정확도:',
evalscore[1]
테스트 정확도: 0.9851 print'테스
트 손실:', evalscore[0]
테스트 손실: 0.06053220131823819
```

몇 개의 숫자를 인식해 봅시다.

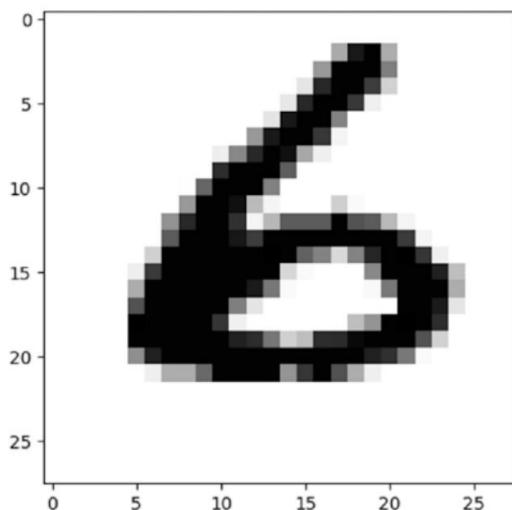
```
image_idx = 6700
plt.imshow(x_test[image_idx].reshape(28, 28),cmap='회색') plt.show()
```

7장 딥러닝



```
pred = model.predict(x_test[image_idx].reshape(1, 28, 28, 1))
print(pred.argmax())
4

image_idx = 8200
plt.imshow(x_test[image_idx].reshape(28, 28),cmap='회색') plt.show()
```



```
pred = model.predict(x_test[이미지_idx].reshape(1, 28, 28, 1))
```

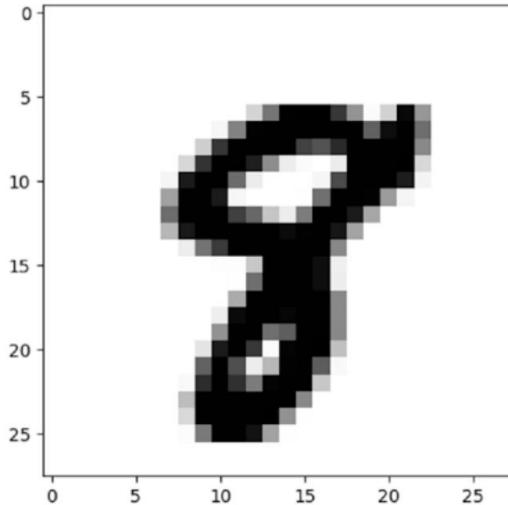
```
인쇄(pred.argmax())
```

```
6
```

이미지 IDX = 8735

```
plt.imshow(x_test[이미지_idx].reshape(28, 28),cmap='회색')
```

```
plt.show()
```



```
pred = model.predict(x_test[이미지_idx].reshape(1, 28, 28, 1))
```

```
인쇄(pred.argmax())
```

```
8
```

축하합니다! 우리 모델은 숫자를 정확하게 인식할 수 있었습니다.

Spark를 사용한 분산 딥 러닝

다중 객체 감지기와 같은 복잡한 모델을 교육하는 데 몇 시간, 며칠 또는 몇 주가 걸릴 수 있습니다. 대부분의 경우 단일 다중 GPU 머신은 합리적인 시간에 대규모 모델을 훈련하기에 충분합니다. 더 까다로운 워크로드의 경우 여러 머신에 계산을 분산하면 교육 시간이 크게 줄어들어 신속한 반복 실험이 가능하고 딥 러닝 배포를 가속화할 수 있습니다.

Spark의 병렬 컴퓨팅 및 빅 데이터 기능은 다음을 위한 이상적인 플랫폼입니다.

분산 딥 러닝. 분산 딥 러닝에 Spark를 사용하면 특히 기존 Spark 클러스터가 있는 경우 추가적인 이점이 있습니다. 하는 것이 편리하다

7장 딥러닝

HDFS, Hive, Impala 또는 HBase와 같이 데이터가 저장된 동일한 클러스터에 저장된 대용량 데이터를 분석합니다. 비즈니스 인텔리전스, 기계 학습, ETL 및 기능 엔지니어링과 같이 동일한 클러스터에서 실행되는 다른 유형의 워크로드와 결과를 공유할 수도 있습니다.^{xxviii}

모델 병렬성 대 데이터 병렬성

신경망의 분산 훈련에는 모델 병렬 처리와 데이터 병렬 처리의 두 가지 주요 접근 방식이 있습니다. 데이터 병렬 처리에서 분산 환경의 각 서버는 모델의 전체 복제본을 가져오지만 데이터의 일부만 가져옵니다. 훈련은 전체 데이터 세트의 슬라이스에 있는 모델의 복제본에 의해 각 서버에서 로컬로 수행됩니다 ([그림 7-10\(a\)](#)). 모델 병렬화에서 모델은 서로 다른 서버에 분할됩니다 ([그림 7-10\(b\)](#)). 각 서버가 할당되고 layer.xxx와 같은 단일 신경망의 다른 부분을 처리하는 역할을 합니다. 데이터 병렬 처리는 단순성과 구현 용이성으로 인해 일반적으로 더 많이 사용됩니다. 그러나 모델 병렬 처리는 너무 커서 단일 머신에 맞지 않는 훈련 모델에 선호됩니다. Google의 대규모 분산 딥 러닝 프레임워크인 DistBelief는 모델 및 데이터 병렬 처리를 모두 지원합니다. Uber의 분산 교육 프레임워크인 Horovod는 모델 및 데이터 병렬 처리도 모두 지원합니다.

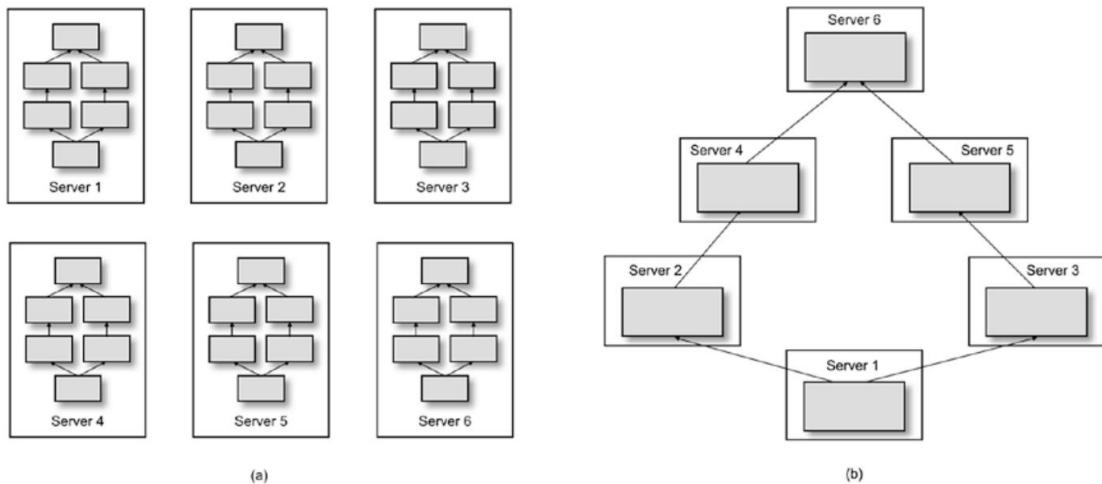


그림 7-10. 데이터 병렬도와 모델 병렬도^{xxix}

Spark용 분산 딥 러닝 프레임워크

타사 기여자 덕분에 Spark의 딥 러닝 지원은 아직 개발 중이지만 Spark에서 실행되는 여러 외부 분산 딥 러닝 프레임워크가 있습니다. 우리는 가장 인기있는 것을 설명 할 것입니다.

딥 러닝 파이프라인

Deep Learning Pipelines는 Spark ML Pipelines API에 통합되는 딥 러닝 기능을 제공하는 Databricks(Spark를 만든 동일한 사람들이 설립한 회사)의 타사 패키지입니다. Deep Learning Pipelines API는 TensorFlow와 함께 TensorFlow 및 Keras를 백엔드로 사용합니다. Spark DataFrame에 이미지를 로드하는 데 사용할 수 있는 ImageSchema가 포함되어 있습니다. 전송 학습, 분산 하이퍼파라미터 조정 및 SQL 기능으로 모델 배포를 지원합니다.^{xxxii} 딥 러닝 파이프라인은 이 글을 쓰는 시점에서 여전히 활발히 개발 중입니다.

BigDL

BigDL은 Intel의 Apache Spark용 분산 딥 러닝 라이브러리입니다. CPU만 지원한다는 점에서 대부분의 딥러닝 프레임워크와 다릅니다. 인텔 아키텍처에서 딥 러닝 프레임워크의 성능을 가속화하기 위한 오픈 소스 라이브러리인 인텔의 심층 신경망용 수학 커널 라이브러리(Intel MKL DNN)와 멀티스레딩을 사용합니다. 성능은 기존 GPU와 비슷하다고 합니다.

카페온스파크

CaffeOnSpark는 Yahoo에서 개발한 딥 러닝 프레임워크입니다. Spark 클러스터 위에서 실행되도록 설계된 Caffe의 분산 확장입니다. CaffeOnSpark는 콘텐츠 분류 및 이미지 검색을 위해 Yahoo 내에서 광범위하게 사용됩니다.

7장 딥러닝

텐서플로우온스파크

TensorFlowOnSpark는 Yahoo에서 개발한 또 다른 딥 러닝 프레임워크입니다. Spark를 사용한 분산 TensorFlow 추론 및 교육을 지원합니다. Spark ML Pipelines와 통합되고 모델 및 데이터 병렬 처리와 비동기 및 동기 교육을 지원합니다.

텐서프레임

TensorFrames는 TensorFlow가 Spark DataFrame과 쉽게 작동할 수 있도록 하는 실험적 라이브러리입니다. Scala 및 Python을 지원하며 Spark에서 TensorFlow로 또는 그 반대로 데이터를 전달하는 효율적인 방법을 제공합니다.

코끼리

Elephas는 Keras를 확장하여 Spark로 확장성이 뛰어난 분산 딥 러닝을 가능하게 하는 Python 라이브러리입니다. Max Pumperla가 개발한 Elephas는 데이터 병렬 처리를 사용하여 분산 딥 러닝을 구현하며 사용 용이성과 단순성으로 유명합니다. 또한 분산 하이퍼파라미터 최적화 및 양상을 모델의 분산 교육을 지원합니다.

분산 케라스

Distributed Keras(Dist-Keras)는 Keras 및 Spark 위에서 실행되는 또 다른 분산 딥 러닝 프레임워크입니다. CERN의 Joeri Hermans가 개발했습니다. ADAG, 동적 SGD, AEASGD(Asynchronous Elastic Averaging SGD), AEAMSGD(Asynchronous Elastic Averaging Momentum SGD) 및 Downpour SGD와 같은 여러 분산 최적화 알고리즘을 지원합니다.

앞서 언급했듯이 이 장은 Elephas와 Distributed Keras에 초점을 맞출 것입니다.

Elephas: Keras 및 Spark를 사용한 분산 딥 러닝

Elephas는 Keras를 확장하여 Spark로 확장성이 뛰어난 분산 딥 러닝을 가능하게 하는 Python 라이브러리입니다. 단일 다중 GPU 시스템보다 성장한 Keras 사용자는 기존 Keras 프로그램을 다시 작성할 필요 없이 모델 교육을 확장하는 방법을 찾고 있습니다. Elephas(및 Distributed Keras)는 이를 수행하는 쉬운 방법을 제공합니다.

Elephas는 데이터 병렬 처리를 사용하여 여러 서버에 Keras 모델 교육을 배포합니다. Elephas를 사용하면 드라이버에서 초기화한 후 Keras 모델, 데이터 및 매개변수가 직렬화되고 작업자 노드에 복사됩니다. Spark 작업자는 데이터의 일부에 대해 훈련하고 그라디언트를 다시 보내고 옵티마이저를 사용하여 동기식 또는 비동기식으로 마스터 모델을 업데이트합니다.

Elephas의 주요 추상화는 SparkModel입니다. Elephas는 컴파일된 Keras를 전달합니다. SparkModel을 초기화하는 모델입니다. 그런 다음 Keras.xxxxii를 사용하는 방법과 유사하게 Epoch 수, 배치 크기, 유효성 검사 분할 및 자세한 수준과 같은 옵션 및 교육 데이터로 RDD를 전달하여 fit 메서드를 호출할 수 있습니다. spark를 사용하여 Python 스크립트를 실행합니다. -제출 또는 pyspark.

elephas.spark_model에서 SparkModel 가져오기
elephas.utils.rdd_utils에서 _simple_rdd로 가져오기

```
rdd = to_simple_rdd(sc, x_train, y_train)

spark_model = SparkModel(모델, 주파수='에포크', 모드='비동기')
spark_model.fit(rdd, epochs=10, batch_size=16, verbose=2, validation_
분할=0.2)
```

Keras 및 Spark와 함께 Elephas를 사용하는 MNIST로 필기 숫자 인식

Elephas.xxxxii가 포함된 첫 번째 예제에서는 MNIST 데이터 세트를 사용합니다. 코드는 이전 Keras 예제와 비슷하지만 훈련 데이터가 Spark RDD로 변환되고 모델이 Spark를 사용하여 훈련된다는 점만 다릅니다. 이렇게 하면 분산 딥 러닝에 Elephas를 사용하는 것이 얼마나 쉬운지 알 수 있습니다. 예제에서는 pyspark를 사용합니다. 목록 7-3을 참조하십시오.

목록 7-3. Elephas, Keras 및 Spark를 사용한 분산 딥 러닝

```
# MNIST 데이터를 다운로드합니다.

수입 케라스
matplotlib.pyplot을 plt로 가져오기

keras.datasets에서 가져오기 mnist
keras.models에서 가져오기 순차
keras.layers에서 Dense, Conv2D, Dropout, Flatten, MaxPooling2D 가져오기
```

7장 딥러닝

```

elephas.spark_model에서 가져오기 SparkModel에서
elephas.utils.rdd_utils에서 가져오기 to_simple_rdd

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train.shape
(60000, 28, 28)

x_train = x_train.reshape(x_train.shape[0], 28, 28, 1) x_test =
x_test.reshape(x_test.shape[0], 28, 28, 1)

x_train = x_train.astype('float32') x_test =
x_test.astype('float32')

x_train /= 255
x_test /= 255

y_train = keras.utils.to_categorical(y_train, 10) y_test =
keras.utils.to_categorical(y_test, 10)

model = Sequential()
model.add(Conv2D(28, kernel_size=(3,3), input_shape= (28,28,1), name='convlayer1'))
model.add(MaxPooling2D(pool_size=(2, 2 ))) model.add(Flatten())
model.add(Dense(128, activation='relu',name='fclayer1')) model.add(Dropout(0.2))
model.add(Dense(10, 활성화 ='소프트맥스', 이름='출력'))

```

인쇄(모델.요약())

모델: "sequential_1"

레이어(유형)	출력 형태	매개변수 #
convlayer1(Conv2D)	(없음, 26, 26, 28)	280
max_pooling2d_1(MaxPooling2D(없음, 13, 13, 28))		0

flatten_1(평평하게)	(없음, 4732)	0
fclayer1(고밀도)	(없음, 128)	605824
dropout_1(탈락)	(없음, 128)	0
출력(밀도)	(없음, 10)	1290

총 매개변수: 607,394

훈련 가능한 매개변수: 607,394

훈련할 수 없는 매개변수: 0

없음

모델을 컴파일합니다.

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

기능 및 레이블에서 RDD를 빌드합니다.

```
rdd = to_simple_rdd(sc, x_train, y_train)
```

Keras 모델과 Spark 컨텍스트에서 SparkModel을 초기화합니다.

```
spark_model = SparkModel(모델, 주파수='에포크', 모드='비동기')
```

Spark 모델을 훈련시킵니다.

```
spark_model.fit(rdd, epochs=20, batch_size=128, verbose=1, validation_
분할=0.2)
```

출력은 간결함을 위해 편집되었습니다.

```
15104/16051 [=====>.....] - ETA: 0s - 손실: 0.0524 - acc: 0.9852
```

```
9088/16384 [=====>.....] - ETA: 2초 - 손실: 0.0687 -
```

계정: 0.9770

```
9344/16384 [=====>.....] - ETA: 2초 - 손실: 0.0675 -
```

계정: 0.9774

7장 딥러닝

15360/16051 [=====>..] - ETA: 0s - 손실: 0.0520 - acc: 0.9852

9600/16384 [=====>.....] - ETA: 2초 - 손실: 0.0662 -

계정: 0.9779

15616/16051 [=====>.] - ETA: 0초 - 손실: 0.0516 - 기준: 0.9852

9856/16384 [=====>.....] - ETA: 1초 - 손실: 0.0655 -

계정: 0.9781

15872/16051 [=====>.] - ETA: 0s - 손실: 0.0510 - acc: 0.9854

10112/16384 [=====>.....] - ETA: 1초 - 손실: 0.0646 - acc: 0.9782

10368/16384 [=====>.....] - ETA: 1초 - 손실: 0.0642 - acc: 0.9784

10624/16384 [=====>.....] - ETA: 1초 - 손실: 0.0645 - acc: 0.9784

10880/16384 [=====>.....] - ETA: 1초 - 손실: 0.0643 - acc: 0.9787

11136/16384 [=====>.....] - ETA: 1초 - 손실: 0.0633 -

계정: 0.9790

11392/16384 [=====>.....] - ETA: 1초 - 손실: 0.0620 -

계정: 0.9795

16051/16051 [=====] - 6초 370us/step - 손실: 0.0509 - acc: 0.9854

- val_loss : 0.0593 - val_acc: 0.9833

127.0.0.1 - [01/Sep/2019 23:18:57] "POST /update HTTP/1.1" 200 -

11648/16384 [=====>.....] - ETA: 1초 - 손실: 0.0623 -

계정: 0.9794

[스테이지 0:=====>.....] / 3]794 (2 + 1)

12288/16384 [=====>.....] - ETA: 1초 - 손실: 0.0619 - acc: 0.9798

12672/16384 [=====>.....] - ETA: 1초 - 손실: 0.0615 -

계정: 0.9799

13056/16384 [=====>.....] - ETA: 0s - 손실: 0.0610 -

계정: 0.9799

7장 딥러닝

```

13440/16384 [=====>.....] - ETA: 0s - 손실: 0.0598 - acc: 0.9803

13824/16384 [=====>.....] - ETA: 0s - 손실: 0.0588 - acc: 0.9806

14208/16384 [=====>....] - ETA: 0s - 손실: 0.0581 - acc: 0.9808

14592/16384 [=====>....] - ETA: 0s - 손실: 0.0577 - acc: 0.9809

14976/16384 [=====>...] - ETA: 0s - 손실: 0.0565 - acc: 0.9812

15360/16384 [=====>..] - ETA: 0s - 손실: 0.0566 - acc: 0.9811

15744/16384 [=====>..] - ETA: 0s - 손실: 0.0564 - acc: 0.9813

16128/16384 [=====>.] - ETA: 0s - 손실: 0.0557 - acc: 0.9815

16384/16384 [=====] - 5초 277us/step - 손실: 0.0556 -
acc: 0.9815 - val_loss : 0.0906 - val_acc: 0.9758
127.0.0.1 -- [01/Sep/2019 23:18:58] "POST /update HTTP/1.1" 200 -
>>> 비동기 학습이 완료되었습니다.
127.0.0.1 -- [01/Sep/2019 23:18:58] "GET /parameters HTTP/1.1" 200 -
# Spark 모델을 평가합니다.

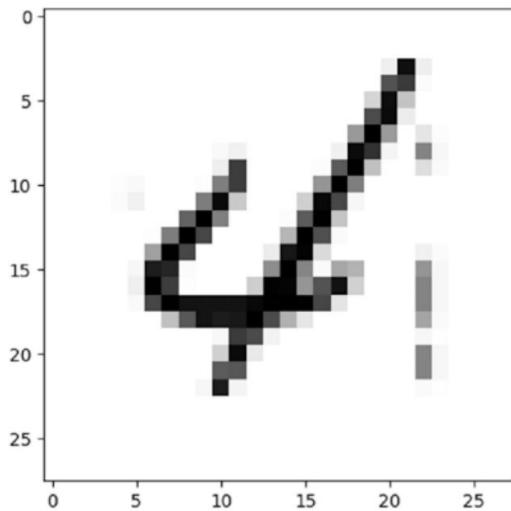
평가 점수 = spark_model.master_network.evaluate(x_test, y_test, 자세한 정보=2)
print'테스트 정확도: ', 평가 점수[1]
테스트 정확도: 0.9644
print'테스트 손실: ', 평가 점수[0]
테스트 손실: 0.12604748902269639

# Spark 모델을 사용하여 테스트 숫자 인식을 수행합니다.

이미지 IDx = 6700
plt.imshow(x_test[이미지_idx].reshape(28, 28),cmap='회색')
plt.show()

```

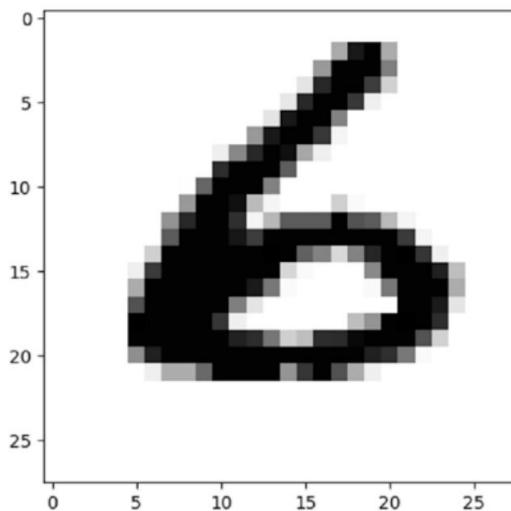
7장 딥러닝



```
pred = spark_model.predict(x_test[image_idx].reshape(1, 28, 28, 1)) print(pred.argmax())
```

4

```
image_idx = 8200  
plt.imshow(x_test[image_idx].reshape(28, 28),cmap='회색') plt.show()
```



```
pred = spark_model.predict(x_test[이미지_idx].reshape(1, 28, 28, 1))
```

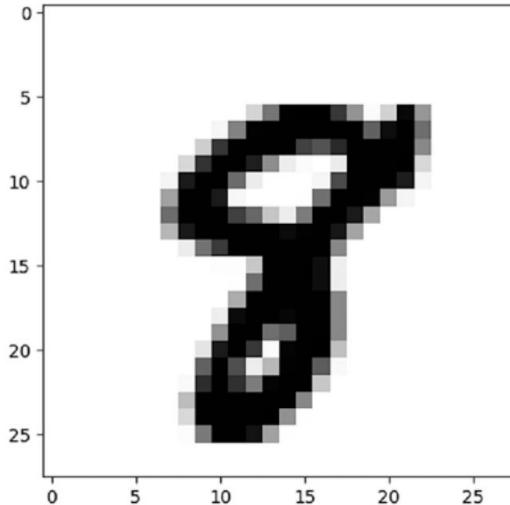
```
인쇄(pred.argmax())
```

```
6
```

이미지 IDx = 8735

```
plt.imshow(x_test[이미지_idx].reshape(28, 28), cmap='회색')
```

```
plt.show()
```



```
pred = spark_model.predict(x_test[이미지_idx].reshape(1, 28, 28, 1))
```

```
인쇄(pred.argmax())
```

```
8
```

이 예에서는 Python을 사용하여 numpy 배열에서 RDD를 생성했습니다. 이것은 정말로 큰 데이터 세트에는 적합하지 않을 수 있습니다. 데이터가 메모리에 맞지 않는 경우 분산 스토리지 엔진(예: HDFS 또는 S3)에서 직접 읽고 Spark MLlib의 변환기 및 추정기를 사용하여 모든 전처리를 수행하여 Spark를 사용하여 RDD를 생성하는 것이 가장 좋습니다. Spark의 분산 처리 기능을 활용하여 RDD를 생성하면 완전히 분산된 딥 러닝 플랫폼을 갖게 됩니다.

Elephas는 또한 Spark DataFrames와 함께 Spark MLlib 추정기를 사용하여 모델 교육을 지원합니다. 더 큰 Spark MLlib 파이프라인의 일부로 추정기를 실행할 수 있습니다. 목록 [7-4를 참조하십시오.](#)

7장 딥러닝

목록 7-4. DataFrame을 사용하여 Spark ML Estimator로 모델 학습

keras 가져오기

matplotlib.pyplot을 plt로 가져오기

```
from keras.datasets import mnist from
keras.models import Sequential from
keras.layers import Dense, Dropout from keras 가
져오기 옵티마이저
```

pyspark.sql.functions에서 rand 가져오기

pyspark.mllib.evaluation에서 가져오기 pyspark.ml 가져오기 파이프라인에
서 MulticlassMetrics 가져오기

elephas.ml_model에서 가져오기 Elephas.ml.adapter에서
ElephasEstimator 가져오기 to_data_frame

MNIST 데이터를 다운로드합니다.

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

x_train.shape

#{(60000, 28, 28)}

우리는 네트워크에 밀집된 레이어만 사용할 것입니다. # 회색조 28x28 이미지를 784 벡터($28 \times 28 \times 1 = 784$)로 병합합시다.

```
x_train = x_train.reshape(60000, 784) x_test =
```

```
x_test.reshape(10000, 784)
```

```
x_train = x_train.astype('float32') x_test =
```

```
x_test.astype('float32')
```

x_train /= 255

x_test /= 255

```
y_train = keras.utils.to_categorical(y_train, 10) y_test =
```

```
keras.utils.to_categorical(y_test, 10)
```

Spark DataFrames는 3차원 데이터에서 생성할 수 없기 때문에 # Spark DataFrames와 함께
Elephas 및 Keras를 사용할 때 고밀도 레이어를 사용해야 합니다. 컨볼루션을 사용해야 하는
경우 RDD를 사용합니다.

Elephas와 레이어.

```
model = Sequential()
model.add(Dense(128, input_dim=784, activation='relu', name='fclayer1')) model.add(Dropout(0.2))
model.add(Dense(128, activation='relu', name='fclayer2')) model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax', name='출력'))
```

기능 및 레이블에서 Spark DataFrame을 빌드합니다.

```
df = to_data_frame(sc, x_train, y_train, categorical=True)
```

```
df.show(20,50)
```

	기능 라벨
+-----+	
	기능 라벨
+-----+	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 5.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 0.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 4.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 1.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 9.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 2.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 1.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 3.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 1.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 4.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 3.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 5.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 3.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 6.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 1.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 7.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 2.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 8.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 6.0	
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 9.0	
+-----+	

7장 딥러닝

상위 20개 행만 표시

```
test_df = to_data_frame(sc, x_test, y_test, categorical=True)
```

```
test_df.show(20,50)
```

기능 라벨
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 7.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 2.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 1.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 0.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 4.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 1.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 4.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 9.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 5.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 9.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 0.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 6.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 9.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 0.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 1.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 5.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 9.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 7.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 3.0
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,... 4.0

상위 20개 행만 표시

옵티마이저를 설정하고 직렬화합니다.

```
sgd = optimizers.SGD(lr=0.01)
```

```
optimizer_conf = optimizers.serialize(sgd)
```

Spark ML Estimator를 초기화합니다.

```

추정기 = ElephasEstimator()
estimator.set_keras_model_config(model.to_yaml())
estimator.set_optimizer_config(optimizer_conf)
estimator.set_epochs(25)
estimator.set_batch_size(64)
estimator.set_categorical_labels(True)
estimator.set_validation_split(0.10)
estimator.set_nb_classes(10)
Estimator.set_mode("동기")
estimator.set_loss("categorical_crossentropy")
estimator.set_metrics(['acc'])

# 모델을 맞춥니다.

```

파이프라인 = 파이프라인(단계=[추정기])
 파이프라인 모델 = 파이프라인.핏(df)

테스트 데이터에서 피팅된 파이프라인 모델을 평가합니다.

```

예측 = pipeline_model.transform(test_df)
df2 = prediction.select("레이블", "예측")
df2.show(20)

```

라벨	예측
7.0	7.0
2.0	2.0
1.0	1.0
0.0	0.0
4.0	4.0
1.0	1.0
4.0	4.0
9.0	9.0
5.0	6.0
9.0	9.0
0.0	0.0
6.0	6.0

7장 딥러닝

9.0	9.0
0.0	0.0
1.0	1.0
5.0	5.0
9.0	9.0
7.0	7.0
3.0	2.0
4.0	4.0
+----+	-----+

상위 20개 행만 표시

```
Prediction_and_label= df2.rdd.map(람다 행: (row.label, row.prediction))
```

```
메트릭 = MulticlassMetrics(prediction_and_label)
```

```
인쇄(metrics.precision())
```

```
0.757
```

분산 케라스(Dist-Keras)

Distributed Keras(Dist-Keras)는 Keras 및 Spark에서 실행되는 또 다른 분산 딥 러닝 프레임워크입니다. CERN의 Joeri Hermans가 개발했습니다. ADAG, Dynamic SGD, AEASGD(Asynchronous Elastic Averaging SGD), AEAMSGD(Asynchronous Elastic Averaging Momentum SGD) 및 Downpour SGD와 같은 여러 분산 최적화 알고리즘을 지원합니다. Dist-Keras에는 ReshapeTransformer, MinMaxTransformer, OneHotTransformer, DenseTransformer 및 LabelIndexTransformer와 같은 다양한 데이터 변환을 위한 자체 Spark 변환기가 포함되어 있습니다. Elephas와 마찬가지로 Dist-Keras는 데이터 병렬 처리를 사용하여 분산 딥 러닝을 구현합니다.

MNIST를 사용한 필기 숫자 인식 Keras 및 Spark가 있는 Dist-Keras

일관성을 위해 Dist-Keras example.xxxiv에 MNIST 데이터 세트를 사용합니다. 이 예제를 실행하기 전에 MNIST 데이터 세트를 HDFS 또는 S3에 저장해야 합니다. 목록 [7-5를 참조하십시오](#).

목록 7-5. Dist-Keras, Keras 및 Spark를 사용한 분산 딥 러닝

distkeras.evaluators에서 가져오기 *
 distkeras.predictors에서 가져오기 *
 distkeras.trainers에서 가져오기 *
 distkeras.transformers에서 가져오기 *
 distkeras.utils에서 가져오기 가져오기
 keras.layers.convolution에서 가져오기 *
 keras.layers.core에서 가져오기 keras.models에
 서 가져오기 keras.optimizers에서 순차 가져오기 가
 져오기
 pyspark에서 가져오기 SparkConf에서
 pyspark에서 가져오기 SparkContext
 pyspark.ml.evaluation에서 가져오기 pyspark.ml.feature에서
 MulticlassClassificationEvaluator 가져오기 pyspark.ml.feature에서 OneHotEncoder
 가져오기 pyspark.ml.feature에서 StandardScaler 가져오기 pyspark.ml.feature에서
 StringIndexer 가져오기 VectorAssembler 가져오기

```

import pwd
import os #
먼저 Spark 변수를 설정합니다. 필요에 맞게 수정할 수 있습니다. application_name = "분산
Keras MNIST 노트북" using_spark_2 = False local = False
  
```

```

path_train = "data/mnist_train.csv"
path_test = "data/mnist_test.csv" 로컬인
경우:
  # 마스터에게 로컬 리소스를 사용하도록 지시합니다.
  마스터 = "로컬[*]"
  num_processes = 3
  num_executors = 1
  
```

7장 딥러닝

또 다른:

```
# 마스터에게 YARN을 사용하라고 합니다.

마스터 = "실 클라이언트"
num_executors = 20
num_processes = 1

# 이 변수는 코어 및 실행기의 수에서 파생되며
모델 트레이너의 수를 할당하는 데 사용됩니다.
num_workers = num_executors * num_processes
print("원하는 실행자 수: print("원하는 프로세스 수 / " + `num_executors` ")
실행자: + `num_processes`)
print("전체 작업자 수: + `num_workers`")
```

Databricks CSV 리더를 사용합니다. 이것은 유효하지 않은 값에 관한 몇 가지 좋은 기능을 가지고 있습니다.

```
os.environ['PYSPARK_SUBMIT_ARGS'] = '--파키지 com.databricks:spark csv_2.10:1.4.0
pyspark-shell'

conf = SparkConf()
conf.set("spark.app.name", application_name)
conf.set("스파크.마스터", 마스터)
conf.set("spark.executor.cores", `num_processes`)
conf.set("spark.executor.instances", `num_executors`)
conf.set("spark.executor.memory", "4g")
conf.set("spark.locality.wait", "0")
conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer");
conf.set("spark.local.dir", "/tmp/" + get_os_username() + "/dist-keras");
```

사용자가 Spark 2.0 +를 실행 중인지 확인

_spark_2를 사용하는 경우:

```
sc = SparkSession.builder.config(conf=conf) \
    .appName(응용 프로그램 이름) \
    .getOrCreate()
```

또 다른:

Spark 컨텍스트를 생성합니다.

```
sc = SparkContext(conf=conf)
```

누락된 가져오기를 추가합니다.

pyspark에서 가져오기 SQLContext

```
sqlContext = SQLContext(sc)
```

Spark 2.0을 사용 중인지 확인합니다.

using_spark_2: 리더 = sc인 경우

또 다른:

```
reader = sqlContext # 훈
```

련 데이터 세트를 읽습니다.

```
raw_dataset_train = reader.read.format('com.databricks.spark.csv') \
    .options(header='true', inferSchema='true') \ .load(path_train)
```

테스트 데이터 세트를 읽습니다.

```
raw_dataset_test = reader.read.format('com.databricks.spark.csv') \
    .options(header='true', inferSchema='true') \ .load(경로_테스트)
```

먼저 원시 데이터 세트에서 원하는 기능을 # 추출하고 싶습니다. 원하는 모든 열로 목록을 구성하여 이를 수행합니다.

이것은 테스트 세트에서도 동일합니다.

```
기능 = raw_dataset_train.columns
```

```
features.remove('레이블')
```

다음으로 Spark의 VectorAssembler를 사용하여 원하는 모든 기능의 벡터를 # "어셈블"(생성)합니다.

```
벡터_어셈블러 = VectorAssembler(inputCols=features, outputCol="features")
```

이 변환기는 기능에 지정된 모든 열을 가져오고 원하는 모든 것을 포함할 추가 열 "기능"을 생성합니다.

단일 벡터로 집계된 기능.

```
dataset_train = vectorAssembler.transform(raw_dataset_train) dataset_test =
vectorAssembler.transform(raw_dataset_test)
```

출력 클래스의 수를 정의합니다.

7장 딥러닝

```

nb_classes = 10 인코
더 = OneHotTransformer(nb_classes, input_col="label", output_col="label_encoded")
dataset_train = encoder.transform(dataset_train) dataset_test = 인코
더.transform(dataset_test)

# 정규화를 위해 Distributed Keras에서 MinMaxTransformer 할당
# 특징.
# o_min -> original_minimum
# n_min -> new_minimum

변환기 = MinMaxTransformer(n_min=0.0, n_max=1.0, \
                           o_min=0.0, o_max=250.0, \
                           input_col="features", \
                           output_col="features_normalized")

# 데이터셋을 변환합니다.

dataset_train = 변환기.transform(dataset_train) dataset_test = 변환
기.transform(dataset_test)

# Keras는 벡터가 특정 모양일 것으로 예상합니다. Spark를 사용하여 # 벡터를 재구성할 수 있습니다.
reshape_transformer = ReshapeTransformer("features_normalized", "matrix", (28, 28, 1))
dataset_train = reshape_transformer.transform(dataset_train) dataset_test =
reshape_transformer.transform(dataset_test)

# 이제 Keras 모델을 생성합니다.

# Keras MNIST 예제에서 가져옴.

# 모델 매개변수를 선언합니다. img_rows,
img_cols = 28, 28 # 사용할 컨볼루션 필
터 수
nb_filters = 32
# 최대 풀링을 위한 풀링 영역의 크기 pool_size = (2, 2)

# 컨볼루션 커널 크기
kernel_size = (3, 3)
input_shape = (img_rows, img_cols, 1)

```

```
# 모델을 구성합니다.

convnet = Sequential()
convnet.add(Convolution2D(nb_filters, kernel_size[0], kernel_size[1]),
            border_mode='유효',
            input_shape=input_shape))

convnet.add(활성화('relu'))
convnet.add(Convolution2D(nb_filters, kernel_size[0], kernel_size[1])) convnet.add(활성화('relu'))
convnet.add(MaxPooling2D(pool_size=pool_size)) convnet.add(Flatten()) convnet.add(Dense(225))
convnet.add(Activation('relu')) convnet.add(Dense(nb_classes)) convnet.add(Activation('softmax'))

# 옵티마이저와 손실을 정의합니다. optimizer_convnet =
'아담' loss_convnet = 'categorical_crossentropy'

# 요약을 인쇄합니다.

convnet.summary()

# 분산 방식으로 데이터 세트를 평가할 수도 있습니다.

# 그러나 이를 위해서는 이를 수행하는 방법에 대한 절차를 지정해야 합니다. def 평가_정확도(모델, 테스트_세트, 기능 =
="매트릭스"):

    evaluator = AccuracyEvaluator(prediction_col="prediction_index", label_col="label") 예측
    자 = ModelPredictor(keras_model=model, features_col=features) 변환기 =
    LabelIndexTransformer(output_dim=nb_classes) test_set = test_set.select(features, "label")
    test_set = predictor.predict(test_set) test_set = 변환기.transform(test_set) 점수 =
    evaluator.evaluate(test_set)
```

반환 점수

7장 딥러닝

```
# 원하는 열을 선택하십시오. 이렇게 하면 네트워크 사용량이 줄어듭니다.
dataset_train = dataset_train.select("features_normalized", "matrix", "label",
"label_encoded")
dataset_test = dataset_test.select("features_normalized", "matrix", "label", "label_encoded")

# Keras는 DenseVectors를 기대합니다.

Dense_transformer = DenseTransformer(input_col="features_normalized",
output_col="features_normalized_dense")
dataset_train = Dense_transformer.transform(dataset_train)
dataset_test = Dense_transformer.transform(dataset_test)
dataset_train.repartition(num_workers)
dataset_test.repartition(num_workers)

# 훈련 및 테스트 세트를 평가합니다.
training_set = dataset_train.repartition(num_workers)
test_set = dataset_test.repartition(num_workers)

# 캐시에 저장합니다.
training_set.cache()
test_set.cache()

# 단순 카운트를 사용하여 노드에서 훈련 세트를 미리 캐시합니다.
print(training_set.count())

# ADAG 옵티마이저를 사용합니다. 테스트를 위해 SingleWorker를 사용할 수도 있습니다.

# 목적 -> 전통적인 비분산 경사하강법.

트레이너 = ADAG(keras_model=convnet, worker_optimizer=optimizer_convnet,
loss=loss_convnet, num_workers=num_workers, batch_size=16, communication_
창=5, num_epoch=5, features_col="매트릭스", label_col="label_encoded")

training_model = training.train(training_set)
"""
print("훈련 시간: " + str(trainer.get_training_time()))
print("정확도: " + str(evaluate_accuracy(trained_model, test_set)))
print("매개변수 서버 업데이트 횟수: " + str(trainer.parameter_
server.num_updates))
```

딥 러닝 워크로드를 여러 머신에 분산하는 것이 항상 좋은 접근 방식은 아닙니다. 분산 환경에서 작업을 실행하는 데 오버헤드가 있습니다. 분산 Spark 환경을 설정하고 유지 관리하는 데 드는 시간과 노력은 말할 것도 없습니다. 고성능 다중 GPU 머신 및 클라우드 인스턴스를 사용하면 이미 단일 머신에서 상당히 큰 모델을 우수한 훈련 속도로 훈련할 수 있습니다. 분산 환경이 전혀 필요하지 않을 수도 있습니다. 사실 `ImageDataGenerator` 클래스를 사용하여 데이터를 로드하고 `fit_generator` 함수를 사용하여 Keras에서 모델을 훈련하는 것으로 대부분의 경우 충분할 수 있습니다. 다음 예에서 이 옵션을 살펴보겠습니다.

개와 고양이 이미지 분류

이 예에서 우리는 개와 고양이 이미지 분류기를 구축하기 위해 컨볼루션 신경망을 사용할 것입니다. 우리는 Francois Chollet이 대중화하고 Microsoft Research에서 제공하고 Kaggle.xxxv에서 사용할 수 있는 인기 있는 데이터 세트를 사용할 것입니다. MNIST 데이터 세트와 마찬가지로 이 데이터 세트는 연구에 널리 사용됩니다. 여기에는 12,500개의 고양이 이미지와 12,500개의 개 이미지가 포함되어 있지만 교육 속도를 높이기 위해 각 클래스에 대해 2000개의 이미지(총 이미지 4000개)만 사용합니다. 테스트를 위해 500개의 고양이 이미지와 500개의 강아지 이미지(총 1000개의 이미지)를 사용합니다. 그림 7-11을 참조하십시오.



그림 7-11. 데이터세트의 샘플 개와 고양이 이미지

우리는 케라스 모델을 훈련시키기 위해 `fit_generator`을 사용할 것입니다. 또한 `ImageDataGenerator` 클래스를 활용하여 데이터를 일괄적으로 로드하여 대량의 데이터를 처리할 수 있습니다. 이는 메모리에 맞지 않는 대용량 데이터 세트가 있고 분산 환경에 액세스할 수 없는 경우에 특히 유용합니다. `ImageDataGenerator` 사용의 추가 이점은 임의의 데이터 변환을 수행하여

7장 딥러닝

모델을 더 잘 일반화하고 과적합을 방지하는 데 도움이 되는 데이터.xxxxvi 이 예제에서는 Spark를 사용하지 않고 Keras에서 대규모 데이터 세트를 사용하는 방법을 보여줍니다. 목록 7-6을 참조하십시오.

목록 7-6. ImageDataGenerator 및 Fit_Generator 사용

matplotlib.pyplot을 plt로 가져오기 numpy
를 np로 가져오기 cv2 가져오기

keras.preprocessing.image 가져오기 keras.preprocessing에서
ImageDataGenerator 가져오기 keras.models에서 이미지 가져오기 keras.layers
에서 순차 가져오기 keras.layers에서 Conv2D, MaxPooling2D 가져오기
Activation, Dropout, Flatten, Dense from keras 가져오기 백엔드를 K로 가져
오기

```
# 이미지 크기는 150x150입니다. RGB = 3.

if K.image_data_format() == 'channels_first': input_shape
    = (3, 150, 150) else:

    input_shape = (150, 150, 3)

모델 = 순차()

model.add(Conv2D(32, (3, 3), input_shape=input_shape, 활성화='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3), 활성화='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), 활성화='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5)) model.add(Dense(1,
activation='sigmoid'))
```

모델을 컴파일합니다.

```
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

인쇄(모델.요약())

레이어(유형)	출력 형태	매개변수 #
conv2d_1(Conv2D)	(없음, 148, 148, 32)	896
max_pooling2d_1(MaxPooling2D)	(없음, 74, 74, 32)	0
conv2d_2(Conv2D)	(없음, 72, 72, 32)	9248
max_pooling2d_2(MaxPooling2D)	(없음, 36, 36, 32)	0
conv2d_3(Conv2D)	(없음, 34, 34, 64)	18496
max_pooling2d_3(MaxPooling2D)	(없음, 17, 17, 64)	0
flatten_1(평평하게)	(없음, 18496)	0
Dense_1(밀도)	(없음, 64)	1183808
dropout_1(탈락)	(없음, 64)	0
조밀한_2(조밀한)	(없음, 1)	65

총 매개변수: 1,212,513

훈련 가능한 매개변수: 1,212,513

훈련할 수 없는 매개변수: 0

없음

7장 딥러닝

훈련을 위해 다음의 증강 구성을 사용할 것입니다.

```
train_datagen = ImageDataGenerator( rescale=1. /  
    255, width_shift_range=0.2,  
    height_shift_range=0.2,  
    horizontal_flip=True)
```

테스트 데이터에 대한 유일한 증가는 크기 조정입니다.

```
test_datagen = ImageDataGenerator(rescale=1. / 255)  
  
train_generator = train_datagen.flow_from_directory(  
    '/data/train',  
    target_size=(150, 150),  
    batch_size=16,  
    class_mode='binary')
```

2개의 클래스에 속하는 4000개의 이미지를 찾았습니다.

```
validation_generator = test_datagen.flow_from_directory( '/data/test',  
    target_size=(150, 150), batch_size=16, class_mode='binary')
```

2개의 클래스에 속하는 1000개의 이미지를 찾았습니다.

steps_per_epoch는 총 훈련 샘플 수로 설정해야 하고, validation_steps는 테스트 샘플 수로 # 설정해야 합니다. # 모델 훈련을 촉진하기 위해 # epoch를 15로, steps_per_epoch 및 validation_steps를 100으로 설정했습니다.

```
model.fit_generator(train_generator,  
    steps_per_epoch=100, epochs=1=25, validation_data=validation_generator, validation_steps=100)
```

신기원 1/25

100/100 [=====] - 45초 451ms/단계 - 손실: 0.6439 - acc: 0.6244 -
val_loss : 0.5266 - val_acc: 0.7418

신기원 2/25

100/100 [=====] - 44초 437ms/단계 - 손실: 0.6259 - acc: 0.6681 -
val_loss : 0.5577 - val_acc: 0.7304

신기원 3/25

100/100 [=====] - 43초 432ms/단계 - 손실: 0.6326 - acc: 0.6338 -
val_loss : 0.5922 - val_acc: 0.7029

신기원 4/25

100/100 [=====] - 43초 434ms/단계 - 손실: 0.6538 - acc: 0.6300 -
val_loss : 0.5642 - val_acc: 0.7052

에포크 5/25

100/100 [=====] - 44초 436ms/단계 - 손실: 0.6263 - acc: 0.6600 -
val_loss : 0.6725 - val_acc: 0.6746

에포크 6/25

100/100 [=====] - 43초 427ms/단계 - 손실: 0.6229 - acc: 0.6606 -
val_loss : 0.5586 - val_acc: 0.7538

신기원 7/25

100/100 [=====] - 43초 426ms/단계 - 손실: 0.6470 - acc: 0.6562 -
val_loss : 0.5878 - val_acc: 0.7077

신기원 8/25

100/100 [=====] - 43초 429ms/단계 - 손실: 0.6524 - acc: 0.6437 -
val_loss : 0.6414 - val_acc: 0.6539

신기원 9/25

100/100 [=====] - 43초 427ms/단계 - 손실: 0.6131 - acc: 0.6831 -
val_loss : 0.5636 - val_acc: 0.7304

신기원 10/25

100/100 [=====] - 43초 429ms/단계 - 손실: 0.6293 - acc: 0.6538 -
val_loss : 0.5857 - val_acc: 0.7186

신기원 11/25

100/100 [=====] - 44초 437ms/단계 - 손실: 0.6207 - acc: 0.6713 -
val_loss : 0.5467 - val_acc: 0.7279

7장 딥러닝

신기원 12/25

100/100 [=====] - 43초 430ms/단계 - 손실: 0.6131 - acc: 0.6587 -
val_loss : 0.5279 - val_acc: 0.7348

신기원 13/25

100/100 [=====] - 43초 428ms/단계 - 손실: 0.6090 - acc: 0.6781 -
val_loss : 0.6221 - val_acc: 0.7054

신기원 14/25

100/100 [=====] - 42초 421ms/단계 - 손실: 0.6273 - acc: 0.6756 -
val_loss : 0.5446 - val_acc: 0.7506

신기원 15/25

100/100 [=====] - 44초 442ms/단계 - 손실: 0.6139 - acc: 0.6775 -
val_loss : 0.6073 - val_acc: 0.6954

신기원 16/25

100/100 [=====] - 44초 441ms/단계 - 손실: 0.6080 - acc: 0.6806 -
val_loss : 0.5365 - val_acc: 0.7437

신기원 17/25

100/100 [=====] - 45초 448ms/단계 - 손실: 0.6225 - acc: 0.6719 -
val_loss : 0.5831 - val_acc: 0.6935

신기원 18/25

100/100 [=====] - 43초 428ms/단계 - 손실: 0.6124 - acc: 0.6769 -
val_loss : 0.5457 - val_acc: 0.7361

신기원 19/25

100/100 [=====] - 43초 430ms/단계 - 손실: 0.6061 - acc: 0.6844 -
val_loss : 0.5587 - val_acc: 0.7399

신기원 20/25

100/100 [=====] - 43초 429ms/단계 - 손실: 0.6209 - acc: 0.6613 -
val_loss : 0.5699 - val_acc: 0.7280

신기원 21/25

100/100 [=====] - 43초 428ms/단계 - 손실: 0.6252 - acc: 0.6650 -
val_loss : 0.5550 - val_acc: 0.7247

신기원 22/25

100/100 [=====] - 43초 429ms/단계 - 손실: 0.6306 - acc: 0.6594 -
val_loss : 0.5466 - val_acc: 0.7236

신기원 23/25

100/100 [=====] - 43초 427ms/단계 - 손실: 0.6086 - acc: 0.6819 -
val_loss : 0.5790 - val_acc: 0.6824

Epoch 24/25

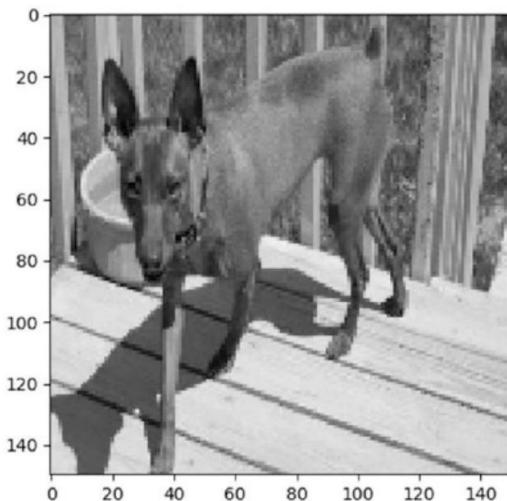
```
100/100 [=====] - 43초 425ms/단계 - 손실: 0.6059 - acc : 0.7000
- val_loss: 0.5433 - val_acc: 0.7197 Epoch 25/25 100/100 [=====]
- 43초 426ms/단계 - 손실: 0.6261 - acc: 0.6794 - val_loss: 0.5987 - val_acc: 0.7167
<keras.callbacks.History 객체 at 0x7ff72c7c3890>
```

71%의 유효성 검사 정확도를 얻습니다. 모델 정확도를 높이려면 더 많은 훈련 데이터를 추가하고 에포크 수를 늘리는 등 여러 가지를 시도할 수 있습니다.

```
model.save_weights('dogs_vs_cats.h5')
```

```
# 이제 모델을 사용하여 몇 개의 이미지를 분류해 보겠습니다. 개=1, 고양이=0 # 개부터 시작하겠습니다.
다. img = cv2.imread("/data/test/dogs/dog.148.jpg")
```

```
img = np.array(img).astype('float32')/255 img =
cv2.resize(img, (150,150)) plt.imshow(img) plt.show()
```



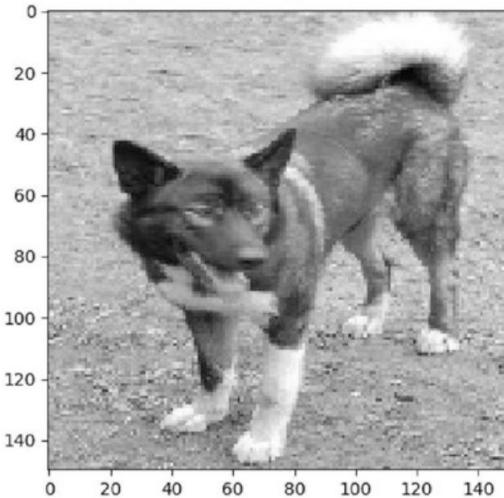
```
img = img.reshape(1, 150, 150, 3)
```

7장 딥러닝

```
print(model.predict(img))
[[0.813732]]

print(round(model.predict(img)))
1.0

# 다른 것
img = cv2.imread("/data/test/dogs/dog.235.jpg") img =
np.array(img).astype('float32')/255 img = cv2.resize(img,
(150,150)) plt.imshow(img) plt.show()
```



```
img = img.reshape(1, 150, 150, 3)

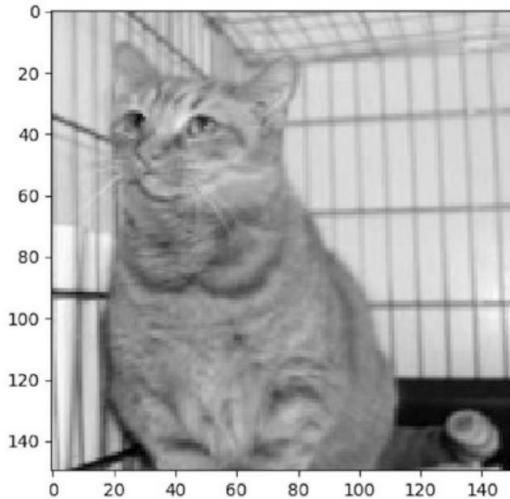
print(model.predict(img))
[[0.92639965]]

print(round(model.predict(img)))
1.0

# 고양이 사진을 몇 장 봅시다.

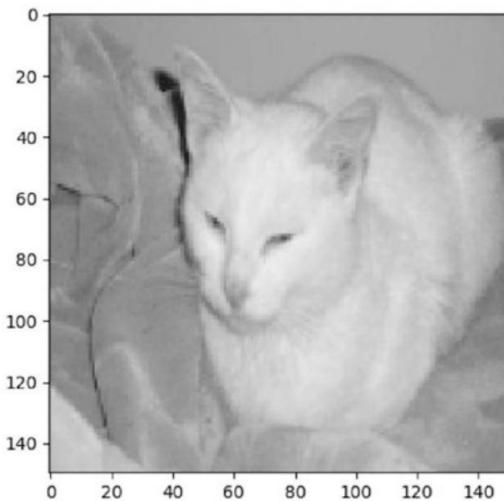
img = cv2.imread("/data/test/cats/cat.355.jpg")
```

```
img = np.array(img).astype('float32')/255 img =  
cv2.resize(img, (150,150)) plt.imshow(img) plt.show()
```



```
img = img.reshape(1, 150, 150, 3)  
  
print(model.predict(img))  
[[0.49332634]]  
  
print(round(model.predict(img)))  
0.0  
  
# 다른 것  
  
img = cv2.imread("/data/test/cats/cat.371.jpg")  
  
img = np.array(img).astype('float32')/255 img =  
cv2.resize(img, (150,150)) plt.imshow(img) plt.show()
```

7장 딥러닝



```
img = img.reshape(1, 150, 150, 3)
```

```
인쇄(모델.예측(img))
```

```
[[0.16990553]]
```

```
print(round(model.predict(img)))
```

```
0.0
```

요약

이 장에서는 Spark를 사용한 딥 러닝 및 분산 딥 러닝에 대한 소개를 제공했습니다. 저는 단순성, 사용 용이성 및 인기 때문에 딥 러닝을 위해 Keras를 선택했습니다. 저는 분산형 딥 러닝을 위해 Elephas와 Dist-Keras를 선택하여 Keras의 사용 편의성을 유지하면서 Spark로 확장성이 뛰어난 딥 러닝 워크로드를 지원했습니다. Elephas 및 Dist-Keras 외에도 Horovod와 같은 다른 비 Spark 분산 딥 러닝 프레임워크를 탐색하는 것이 좋습니다. 딥 러닝에 대한 보다 심층적인 치료를 위해서는 Francois Chollet(Manning)의 Deep Learning with Python 과 Ian Goodfellow, Yoshua Bengio, Aaron Courville(MIT Press)의 Deep Learning 을 추천합니다.

참고문헌

- 나. Francis Crick, 놀라운 가설: 영혼에 대한 과학적 탐색, Scribner, 1995
- ii. 마이클 코플랜드; “인공과의 차이점은 무엇입니까?
지능, 기계 학습 및 딥 러닝?”, nvidia.com, 2016, <https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>
- iii. 엔비디아; “딥 러닝”, developer.nvidia.com, 2019년, <https://developer.nvidia.com/deep-learning>
- iv. 딥마인드; “알파 고”, deepmind.com, 2019, <https://deepmind.com/research/case-studies/alphago-the-story-so-far>
- v. 테슬라; “운전의 미래”, tesla.com, 2019년, www.tesla.com/자동_조종_장치
- vi. 롬 송고르; “Tesla, 자율주행 자동차 제조업체의 기준을 높입니다.”
nvidia.com, 2019년, <https://blogs.nvidia.com/blog/2019/04/23/tesla-자율주행/>
- vii. SAS; “신경망 작동 방식”, sas.com, 2019년, www.sas.com/ko_us/insights/analytics/neural-networks.html
- viii. H2O; “딥러닝(신경망)”, h2o.ai, 2019, <http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/deep-learning.html>
- ix. 마이클 마살리; “McCulloch-Pitts 뉴런”, ilstu.edu, 2019, www.mind.ilstu.edu/curriculum/modOverview.php?modGUI=212
- 엑스. 프랜시스 크릭; 뇌 손상 p. 181, Simon & Schuster, 1995, 놀라운 가설: 영혼에 대한 과학적 탐색
- xi. 데이비드 B. 포겔; 인공 지능 정의 p. 2005년 와일리
진화적 계산: 기계의 새로운 철학을 향하여
인텔리전스, 제3판

7장 딥러닝

- xii. 바나바스 포초스; "머신러닝(강의노트) 퍼셉트론 입문", cmu.edu, 2017, www.cs.cmu.edu/~10701/
[슬라이드/Perceptron_Reading_Material.pdf](#)
- xiii. SAS; "신경망의 역사", sas.com, 2019년, www.sas.com/ko_us/insights/analytics/neural-networks.html
- xiv. 스카이마인드; "신경망에서 역전파에 대한 초보자 가이드", skymind.ai, 2019, <https://skymind.ai/wiki/역전파>
- xv. 코그닐리티카; "사람들이 딥러닝에 지나치게 열광하고, 실제로 전달할 수 있습니까?", cognilytica.com, 2018, www.cognilytica.com/2018/07/10/re- people -over-in-fatuated with-deep-learning-and-can-it-deliver/
16. 얀 르쿤; "불변 인식: 컨볼루션 신경망", lecun.com, 2004, <http://yann.lecun.com/ex/>
[연구/index.html](#)
- xvii. 카일 위거스; "Turing Award 수상자로 Geoffrey Hinton, Yann LeCun 및 Yoshua Bengio가 지명되었습니다." venturebeat.com, 2019, <https://venturebeat.com/2019/03/27/geoffrey-hinton-yann-lecun and-yoshua-bengio-honored-with-the-turing-award/>
- xviii. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton; "이미지넷 Deep Convolutional Neural Networks를 사용한 분류", 토론토. 에듀, 2012, www.cs.toronto.edu/~fritz/absps/imagenet.pdf
- xix. 파이토치; "알렉스넷", pytorch.org, 2019, https://pytorch.org/hub/pytorch_vision_alexnet/
- 더블 엑스. 프랑수아 솔레; "컴퓨터 비전을 위한 딥 러닝", 2018, Deep 파이썬으로 배우기
- xxi. 안드레이 카르파티; "컨볼루션 신경망(CNN/ConvNets)", github.io, 2019, <http://cs231n.github.io/>
[컨볼루션 네트워크/](#)
- xxii. Jayneil Dalal 및 Sohil Patel; "이미지 기본", Packt Publishing, 2013, 인스턴트 OpenCV 스타터

- xxiii. MATLAB; "MATLAB을 사용한 딥 러닝 소개". 수학공부.
[com, 2019년, www.mathworks.com/content/dam/mathworks/tag_team/Objects/d/80879v00_Deep_Learning_ebook.pdf](http://www.mathworks.com/content/dam/mathworks/tag_team/Objects/d/80879v00_Deep_Learning_ebook.pdf)
- xxiv. Vincent Dumoulin 및 Francesco Visin; "컨볼루션 가이드
 딥 러닝을 위한 산술", arxiv.org, 2018, <https://arxiv.org/pdf/1603.07285.pdf>
- xxv. 아니쉬 싱 왈리아; "활성화 기능과 그 종류
 어느 것이 더 낫습니까?", directiondatascience.com, 2017, <https://directiondatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f>
- xxvi. 스카이마인드; "AI 프레임워크 비교", skymind.ai, 2019, <https://skymind.ai/wiki/comparison-frameworks-dl4j-tensorflow-pytorch>
- xxvii. 니하르 가자레; "홍채 데이터 세트를 분류하기 위한 Keras + TensorFlow의 간단한
 신경망", github.com, 2017, <https://gist.github.com/NiharG15/cd8272c9639941cf8f481a7c4478d525>
- xxviii. 빅DL; "BigDL이란 무엇입니까?" github.com, 2019, <https://github.com/intel/bigdl>
- xxix. 스카이마인드; "분산 딥 러닝, 1부: 신경망의 분산 교육 소개", skymind.ai, 2017,
<https://blog.skymind.ai/distributed-deep-learning-part-1-an-introduction-to-distributed-training/>
- 트리플 엑스. 스카이마인드; "분산 딥 러닝, 1부: 신경망의 분산 교육 소개", skymind.ai,
 2017, <https://blog.skymind.ai/distributed-deep-learning-part-1-an-introduction-to-distributed-training/>
- xxx. 데이터브릭; "Apache Spark용 딥 러닝 파이프라인", github.
 com, 2019, <https://github.com/databricks/spark-deep-learning>

7장 딥러닝

xxxii. 맥스 펌펠라; "Elephas: Keras 및 Spark를 사용한 분산 딥 러닝", github.com,

2019년, <https://github.com/maxpumperla/>

[코끼리](#)

xxxiii. 맥스 펌펠라; "mnist_lp_spark.py", github.com, 2019, https://github.com/maxpumperla/elephas/blob/master/examples/mnist_mlp_spark.py

xxxiv. Joeri R. Hermans, CERN IT-DB; "분산 케라스: 분산

Apache Spark 및 Keras를 사용한 딥 러닝," Github.com, 2016, <https://github.com/JoeriHermans/dist-keras/>

xxxv. 마이크로소프트 리서치; "개 대 고양이", Kaggle.com, 2013년,

www.kaggle.com/c/dogs-vs-cats/data

xxxvi. 프랑수아 솔레; "매우 적은 데이터를 사용하여 강력한 이미지 분류 모델 구축", keras.io,

2016, <https://blog.keras.io/>

[건물-강력한-이미지-분류-모델-사용 very-little-data.html](#)

색인

ㅏ

AI 지원 자율주행차, 3
 AI 지원 사이버 보안, 4
 AI 구동 로봇, 4
 알록시오
 Apache Spark, 93 아
 키텍처, 84–86 데이터 처리, 86 정의, 84 하드웨어, 감소, 92 메모리 사용량, 92 데이터 공유

HDFS/S3, 87, 88 메모리 속도, 88
 Spark 작업
 총돌/완료, 89–91 힙 메모리, 89 오프힙 스토리지, 90

교대 최소 제곱(ALS), 15, 246–255
 이상/이상치 탐지, 224 격리 포리스트, 13, 14
 SVM, 14
 아파치 제플린, 32
 곡선 아래 면적(AUC), 21, 22, 77
 수신기 작동 특성(AUROC) 아래 영역, 21–22
 오토인코더, 292

비

가방 넣기, 9, 101
 베이즈의 정리, 8, 99, 156
 빅DL, 315
 이진/이항 분류, 7, 97

씨

카페온스파크, 315
 ChiSqSelector, 74–75
 클러스터링, 12, 189
 Cognitive Toolkit, 300
 coldStartStrategy 매개변수, 251, 252
 협업 필터링, 246, 247
 연결된 구성 요소 알고리즘, 279–281
 콘텐츠 기반 필터링, 265–266
 컨볼루션 레이어, 296, 297
 컨볼루션 신경
 네트워크(CNN), 16개의 아키텍처, 296개의 고밀도 레이어, 294 개의 특징 감지 (특징 감지 레이어 참조) 회색조 이미지, 294, 295개의 입력 모양, 295

상관관계, 75–77
 교차검증기, 22, 62

색인

디

데이터

- 클래스 레이블, 6
- 기능 엔지니어링, 5
- 홍채 데이터 세트, 5
- Microsoft Researchers, 5개 모델, 6개 관찰, 6개
- 데이터브릭(XGBoost), 104
- 데이터 계보, 272
- 데이터 병렬 처리, 314
- 의사 결정 트리, 9, 99–101
- 딥러닝4J, 300
- 딥 러닝 프레임워크
 - CNTK, 300
 - 딥러닝4J, 300
 - 케라스, 300
 - 파이토치, 300
 - 텐서플로우, 299
 - 테아노, 300
- 차원 축소, 14

유향 그래프, 270

방향 다중 그래프, 270, 271

분산 케라스(Dist-Keras), 316, 328 개와 고양이 이미지

지 분류기

- CNN, 335
- fit_generator, 335, 336, 338–344

- ImageDataGenerator, 335, 336, 338–344

- 필기 숫자 인식, 328–334

드롭아웃 레이어, 299

동적 페이지 순위, 278–279

0자정

팔꿈치 방법, 198

Elephas

- 필기 숫자 인식
 - 케라스와 스파크, 317–319, 321–323
 - Spark ML 추정기, 324–328
 - 스파크 MLlib, 323
- 파이썬 스크립트, 317
- 스파크, 316
- 스파크모델, 317

임베디드 메소드, 18

평가 지표

오록, 77–78

F1 마디, 78

RMSE, 78–79

WSSSE, 79

명시적 등급, 247, 248

에프

특징 감지 계층

- 분류 계층, 299 컨볼루션 계층, 296–297 풀링 계층, 298

릴루, 298

- 특징 공학, 5개 정의, 16 개 특징 구성, 19개 특징 추출, 19개 특징 중요도, 18개 특징 선택, 17, 18개 작업, 17 개

기능 확장, 100, 117, 192

필터 방법, 18

금융 서비스, 3

fit_generator 함수, 335

평평한 층, 299
 미래 지향적인 제조업체, 4
 완전 연결 계층, 8, 99, 296, 299

지, 에이치

GBT 대 랜덤 포레스트, 103 GDPR(일반 데이터 보호 규정), 272 GAN (Generative Adversarial Networks), 293 지니 점수, 100 GBT(Gradient-Boosted Tree), 10, 102, 103 그래프 분석, 269 데이터 거버넌스 및 규정 준수, 272 사기 탐지, 272 네트워크 인프라, 273 위험 관리, 272 소셜 네트워킹, 273 교통, 273 GraphFrames DataFrames API, 281 예제, 282–285 GraphX, 83 Edge, 274 EdgeContext, 274 EdgeRDD, 274 Edge4, example, 27 276–278 그래프, 273 VertexRDD, 274

ImageDataGenerator, 335, 336 암시적 등급, 247 인스타카트 고객 256 명 Instacart 온라인 식료품 쇼핑 데이터 세트, 256

사물인터넷(IoT), 4, 242, 273 고립의 숲 이상 탐지, 224 이상 점수, 225 격리 트리, 225, 226

단일 클래스 지원 벡터 기계, 226 개 이상 값, 분리, 224개 Spark-iForest (Spark-iForest 참조) 트리 구조, 224 개 항목 집합, 255 개

제이

주피터, 32

케이

케라스, 300

필기 숫자, MNIST, 306, 308, 309, 311–313 다른 클래스 분류, 301, 302, 304, 305

텐서플로우, 301

K-평균, 12, 189

K-평균 클러스터링

고객 세분화, 191, 193, 194, 196–198

홍채 데이터 세트, 189, 190
 관찰, 190 원-핫 인코딩,
 192 최적 수, 191 실루엣 점수, 199

WSSSE, 198

엘

잠재 디리클레 할당

(LDA), 13, 199, 265 문서 그룹화, 199

인덱스

잠재 디리클레 할당(LDA) (계속)

- Kaggle, 208
- NLP 라이브러리, 200
- Spark NLP 라이브러리 (Spark NLP 참조)
- Stanford CoreNLP, 200–202, 209 주제 모델링, 199 CountVectorizer, 216, 217
 - describeTopics, 220, 221 데이터 읽기, 209–214 확장된 기능, 217, 218 사용자 정의 함수, 219 어휘, 결합, 21514, , 222, 223 LightGBM, 11, 144 장점, 145 이탈 예측, 148–153 기능 중요도, 154, 155 수준별 대 잎별 성장, 146 매개변수, 146, 147 XGBoost, 145 선형 회귀, 12 로지스틱 회귀 8, 98 Iris 데이터 세트, 105 선형 분류기, 104 선형 대 비선형, 105 다중 클래스 분류, 106–116

중

머신 러닝(ML)

- AI 및 딥 러닝, 2 정의, 1 의료, 4 장바구니 분석 콘텐츠 기반 필터 링, 265, 266 FP- 성장, 256–262, 264 매장 최적화, 255 마이크로소프트(LightGBM), 104

ML 파이프라인

- CrossValidator, 62 estimator, 61 evaluator, 62 ParamGridBuilder, 61 파이프라인, 61 Spark MLlib API, 61 변 압기, 61 모델 평가 정확도, 20 AUROC, 21 개 범주, 20개 혼동 행렬, 20개 F1 측정, 21 정밀도, 21 회상, 21

- 모델 병렬성, 314 모델 선택, 22 MovieLens 데이터 세트, 248 다중 클래스/다항 분류, 7, 98 다중 레이블 분류, 7, 98 다층 퍼셉트론, 8, 99 다중 회귀, 12 LightGBM, 176–182 RMSE, 174, 175 XGBoost4J-스파크, 168, 170–174

N

- Naive Bayes, 8, 99 정의, 156 감정 분석 코드, 157–163 데이터 세트, 157 기능 변환, 157 TF-IDF, 157 텍스트 분류, 156 신경망, 16

- AI 패러다임, 292
 - AlexNet, 293
 - backpropagation, 293
 - deep, 292 defined, 291
 - GAN, 293
 - 오픈AI, 294
 - Rosenblatt Perceptron, 292
 - 가지 유형, 292n -gram, 68
 - numTopFeatures, 74
 - 영형
 - OneHotEncoderEstimator, 68–70
 - 과적합, 22
 - 피, 큐
 - 페이지랭크, 278
 - ParamGridBuilder, 61
 - 풀링 레이어, 298
 - 주요 구성 요소 분석 (PCA), 14, 73, 74 정의됨, 232 차원 축소, 232
 - 홍채 데이터세트 차원 축소, 233–239 explainVariance 방법, 242 플롯 샘플, 241 분산, 유지, 242
 - 속성 그래프, 271
- 아크 지형
- 랜덤 포레스트, 9, 101, 102 배깅, 117 이탈 예측, 118–129
- 매개변수 추출, 132, 133 특성 중요도, 130, 131 매개변수, 118
 - Telco 이탈 예측, 131
 - 수신기 작동 특성(ROC), 21, 77
 - 추천 엔진 ALS, 246
 - 협업 필터링, 246 영화, 248, 249, 251–255 매개변수, 248
- ReLU(Rectified Linear Unit) 활성화 기능, 294, 298
- RegexTokenizer, 64
- 회귀, 11, 164
- 강화 학습, 15
- 탄력적인 분산 데이터 세트(RDD)
- 누산기, 42 개의 작업, 40–41개의 브로드캐스트 변수, 42 개의 캐싱, 42 개의 생성
- 병렬화, 35 텍스트 파일, 35 정의, 34
- 지연 평가, 42 변환 (변환,
- RDD) 소매, 3 평균 제곱근 오차(RMSE), 78–79, 174
- 에스
- 반 지도 학습, 15
- 단순 선형 회귀 정의, 164 예, 165, 166, 168 플롯, 164

인덱스

- 소셜 네트워킹, 273 Softmax
- 기능, 8, 99, 299 Spark 애플리케이션, 32 아키텍처, 30, 31 클러스터 관리자, 30 정의, 29 에코시스템, 29, 30 SparkContext, 30 Spark 데이터 소스 Amazon S3, 56 CSV, 45 Excel 스프레드시트, 57 HBase, 50, 52–55 JSON, 46, 47 Parquet, 50 관계형 및 MPP, 47–50 보안 FTP, 58, 59 Solr, 56, 57 XML, 45, 46 Spark, 분산 딥 러닝 BigDL, 315 CaffeOnSpark, 315 Deep 학습 파이프라인, 315 Dist-Keras, 316 Elephas, 316 모델 대 데이터 병렬 처리, 314 텐서플로우온스파크, 316 텐서프레임, 316 Spark-iForest 이상 감지, 229–231 BinaryClassificationMetrics, 231 정의됨, 227 매개변수, 227, 228 Spark MLlib, 15, 59, 60 이진 분류, 81–83 파일 다운로드, 80 심장 질환 데이터 세트, 79, 80 랜덤 포레스트 모델, 79 Spark MLlib 분류 알고리즘 결정 트리, 99, 100 GBT, 102–104 로지스틱 회귀, 98 다중 클래스/론, 99 나이브 베이즈, 99 랜덤 포레스트, 101, 102 서포트 벡터 머신, 98, 99 Spark NLP 주석자, 202 라이트파이프라인, 207 OCR 모듈, 207, 208 사전 훈련된 파이프라인, 202 예, 203 스파크 데이터 프레임, 203, 204 스파크 MLlib, 204–206 Spark MLlib 파이프라인 생성, 206, 207 spark-shell RDD (탄력적 분산 참조 데이터세트(RDD)) 스파크세션, 32, 34 스파크 SQL, 43 SQL변환기, 70 표준 스케일러, 66, 67, 192 정적 PageRank, 279 StopWordsRemover, 67 문자열 인덱서, 62, 63지도 학습, 97 이진/이항 분류, 7 의사 결정 트리, 9 정의, 6 GBT, 10 LightGBM, 11 로지스틱 회귀, 8 다중 클래스/다항 분류, 7

- 다중 레이블 분류, 7
다층 퍼셉트론,
8
나이브 베이즈, 8 랜
덤 포레스트, 9
SVM, 8
XGB부스트, 10
- 서포트 벡터 머신(SVM), 8, 14,
98–99, 292
- 생존 회귀, 12
- 상징적 인공지능, 1, 292
- E**
- 텐서플로우, 299
텐서플로우온스파크, 316
텐서프레임, 316
- 용어 빈도-역 문서 빈도(TF-IDF), 71–73, 157
- 테슬라, 290
텍스트 분류, 156
테아노, 300
타사 프레임워크, 84
토크나이저, 64, 65
- TrainValidationSplit, 22
변환, RDD 병합, 40개 개별,
36, 37 개 필터, 36개
- 플랫맵, 36
내부 조인, 38개
키, 37 개
LeftOuterJoin, 38
지도, 35
ReduceByKey, 37 채
분할, 40
RightOuterJoin, 38 빼
기, 39
- 합집합, 39
개 값, 37 개
삼각형 카운트 알고리즘, 279
- U**
- 언더피팅, 22
무방향 그래프, 269, 270
비지도 학습, 12, 189
이상/이상치 탐지, 13, 14
클러스터링, 12 차원
축소, 14
- K-평균**, 12
LDA, 13
PCA, 14
- V**
- VariantSpark RF, 117
벡터어셈블러, 65
- Y**
- 제곱 오차의 집합 합계 내
(WSSSE), 79, 198
래퍼 메서드, 18
- X, Y, Z**
- XGBoost4J-스파크, 134
XGBoost(eXtreme Gradient
부스팅), 10, 133 매개변
수, 134, 135
XGBoost4J-Spark, 134 이
탈 예측, 136–141 기능 중요도, 142
매개변수, 143, 144