



파이썬으로 아파치 스파크 학습하기

Wenqiang Feng 지음
홍은희 옮김

12월 29일, 2023

목차

1 서문	3
1.1 들어가기에 앞서	3
1.2 튜토리얼 동기	4
1.3 저작권 고지 및 라이선스 정보	4
1.4 감사글	5
1.5 피드백 및 제안	5
2 왜 파이썬으로 스파크를 사용할까?	7
2.1 왜 스파크인가?	7
2.2 왜 파이썬으로 스파크를 사용할까 (PySpark)?	9
3 실행 플랫폼 환경 설정	11
3.1 Databricks 커뮤니티 클라우드에서 실행하기	11
3.2 Mac과 Ubuntu에서 Spark 환경설정	18
3.3 Windows에서 Spark 환경 설정	20
3.4 텍스트 편집기 혹은 IDE에서 PySpark	21
3.5 PySparkling Water: Spark + H2O	29
3.6 클라우드에서 스파크 설치하기	30
3.7 Colaboratory에서 PySpark 사용하기	31
3.8 데모 코드	31
4 아파치 스파크 개요	33
4.1 핵심 개념	33
4.2 스파크 구성요소	34
4.3 아키텍처	36
4.4 스파크는 어떻게 작동될까?	36
5 RDD 프로그래밍	37
5.1 RDD 생성	37
5.2 스파크 연산	41
5.3 rdd.DataFrame vs pd.DataFrame	43
6 통계학 및 선형대수 예비	61
6.1 표기	61
6.2 선형대수 예비	61
6.3 측정 공식	63

6.4 혼동 행렬	64
6.5 통계 검정	65
7 데이터 탐색	67
7.1 단변량 분석	67
7.2 다변량 분석	80
8 데이터 처리: 특징 (Features)	87
8.1 특징 추출	87
8.2 특징 변환	96
8.3 특징 선택	116
8.4 불균형 데이터: 언더 샘플링	117
9 회귀분석	119
9.1 선형 회귀	119
9.2 일반화 선형 회귀	133
9.3 의사결정 나무 - 회귀	142
9.4 랜덤 포레스트 - 회귀	151
9.5 그레디언트 부스팅 트리 - 회귀	159
10 정칙화	167
10.1 최소제곱법 회귀분석	167
10.2 리지 회귀분석	168
10.3 Least Absolute Shrinkage and Selection Operator (LASSO)	168
10.4 엘라스틱 넷(Elastic net)	168
11 분류	169
11.1 이항 로지스틱 회귀분석	169
11.2 다항 로지스틱 회귀분석	181
11.3 의사결정 나무 - 분류	194
11.4 랜덤 포레스트 - 분류	206
11.5 그레디언트 부스팅 트리 - 분류	217
11.6 XGBoost: 그레디언트 부스팅 트리 분류	218
11.7 나이브 베이즈 분류	219
12 군집 분류	233
12.1 K-Means 모델	233
13 RFM 분석	247
13.1 RFM 분석 기법	248
13.2 데모	250
13.3 확장	256
14 텍스트 마이닝	263
14.1 텍스트 모음	263
14.2 텍스트 전처리	271
14.3 텍스트 분류	274
14.4 감정 분석	280
14.5 N-grams과 상관관계	287

14.6 토퍼 모델: Latent Dirichlet Allocation	287
15 소셜 네트워크 분석	305
15.1 도입	306
15.2 동시 출현 네트워크	306
15.3 부록: PySpark에서 행렬 곱셈	310
16 ALS: 주식 포트폴리오 추천	313
16.1 추천 시스템	314
16.2 교대 최소 제곱(Alternating Least Squares)	315
16.3 데모	315
17 몬테 카를로 시뮬레이션	323
17.1 카지노에서 승리 시뮬레이션	324
17.2 랜덤워크 시뮬레이션	326
18 마르코프 체인 몬테 카를로	335
18.1 메트로폴리스 알고리즘	336
18.2 메트로폴리스 토이 예제	336
18.3 데모	337
19 뉴럴 네트워크	345
19.1 순방향 신경망	345
20 Cloudera Distribution Hadoop 자동화	349
20.1 자동화 파이프라인	349
20.2 데이터 정제 및 조작 자동화	349
20.3 ML 파이프라인 자동화	352
20.4 파이프라인 모델 저장 및 로드	353
20.5 결과를 다시 Hadoop에 제출	353
21 PySpark 패키지화	355
21.1 패키지화	355
21.2 PyPI에서 패키지 퍼블리싱	357
22 PySpark 데이터 감사 라이브러리	359
22.1 pip로 설치하기	359
22.2 Repo로부터 설치하기	359
22.3 삭제하기	359
22.4 테스트	360
22.5 빅 데이터셋 감사	367
23 jupyter notebook에서 Zeppelin	371
23.1 설치 방법	371
23.2 데모 변환	372
24 커닝 폐이퍼	377

25 JDBC 연동	381
25.1 JDBC 드라이버	381
25.2 JDBC read	382
26 Databricks 텁	385
26.1 샘플 표시	385
26.2 자동 파일 다운로드	385
26.3 AWS S3에서 작업하기	389
27 PySpark API	405
27.1 Stat API	405
27.2 회귀분석 API	411
27.3 분류 API	430
27.4 군집 분류 API	450
27.5 추천 API	465
27.6 파이프라인 API	470
27.7 튜닝 API	472
27.8 평가 API	477
28 주요 참고	483
문현 정보	485
파이썬 모듈 인덱스	487
인덱스	489



파이썬 노트로 배우는 아파치 스파크에 오신 것을 환영합니다! 이 노트에서는, 데이터 마이닝, 텍스트 마이닝, 머신 러닝 및 딥 러닝의 PySpark에 대한 다양한 개념을 배울 수 있습니다. PDF 버전은 [여기서](#) 다운로드할 수 있습니다.

서문

1.1 들어가기에 앞서

1.1.1 이 노트에 대해서

이 노트는 아파치 스파크 노트를 학습하기 위한 공유 저장소입니다. PDF 버전은 [여기서](#) 다운로드받으실 수 있습니다. 첫 번째 버전은 ChenFeng ([Feng2017]) 깃허브에 게시되어 있습니다. 이 공유저장소는 주로 Wenqiang^o IMA 데이터 과학 웰로우십을 수행하는 동안 작성한 독학 또는 자습 노트의 내용을 포함하고 있습니다. 독자분들은 데이터 셋과 .ipynb 파일들에 관한 자세한 내용을 <https://github.com/runawayhorse001/LearningApacheSpark>에서 참조하시면 됩니다.

이 저장소에서는 자세한 데모 코드와 예제를 사용하여 각 주요 기능을 사용하는 방법을 보여줍니다. 만약 이 노트에 여러분의 작업이 인용되지 않았다면, 언제든지 저에게 알려주세요.

저는 데이터 마이닝 프로그래밍 및 빅 데이터 전문가는 아니지만 Pyspark 프로그래밍에 대해 배운 내용을 간단한 튜토리얼 형태로 자세한 예와 함께 공유하는 것이 제게도 도움될 것이라 생각했습니다. 해당 튜토리얼이 연구에 유용한 도구가 되기를 바랍니다.

튜토리얼은 독자가 프로그래밍과 리눅스에 대한 예비 지식을 가지고 있다고 가정합니다. 그리고 이 문서는 [sphinx](#)를 사용하여 자동으로 생성됩니다.

1.1.2 저자에 관하여

- **Wenqiang Feng**

- 데이터 사이언스 부문 이사 및 수학 박사
- 녹스빌의 테네시 대학교
- 이메일: von198@gmail.com

- 전기

Wenqiang Feng은 AMEX(American Express)의 데이터 사이언스 부문 이사입니다. AMEX 이전에 Feng 박사는 H&R Block의 머신 러닝 랩에서 시니어 데이터 사이언티스트로 일했습니다. Block에 합류하기 전엔 Feng 박사는 DST(현재 SS&C)의 응용 분석 그룹에 데이터 사이언티스트로 일했습니다.

Feng 박사의 직무는 고객들에게 빅데이터 분석 솔루션, 고급 분석 그리고 데이터 향상 기법 및 모델링을 포함하여 최첨단 기술 및 장비에 대한 접근을 제공하는 것입니다.

Feng 박사는 데이터 마이닝, 분석 시스템, 머신 러닝 알고리즘, 비즈니스 인텔리전스 및 빅 데이터 툴들을 다기능 비즈니스에서 산업 문제를 해결하기 위해 전략적으로 적용하는 것에 깊은 분석 전문 지식을 보유하고 있습니다.

DST에 합류하기 전에 평 박사는 미네소타 대학의 수학 및 응용(IMA) 연구소에 IMA 데이터 사이언스 연구원이었습니다. 그곳에 있는 동안에 그는 심층적인 예측 분석을 바탕으로 스타트업 회사들이 마케팅 의사결정을 할 수 있게 도왔습니다.

Feng 박사는 태네시 대학교 뉴스빌에서 컴퓨터 수학 박사와 통계학 석사로 졸업했습니다. 그는 미주리 과학 기술 대학교에서 계산 수학 석사 학위와 중국 과학 기술 대학교에서 응용 수학 석사 학위도 가지고 있습니다.

• 선언

Wenqiang Feng의 작업은 IMA에서 일하는 동안 IMA의 지원을 받았습니다. 그러나 이 자료에 표현된 모든 의견, 결과, 결론 또는 권고는 저자의 것이며 반드시 IMA, UTK, DST, HR & Block 및 AMEX의 견해를 반영하지 않고 있습니다.

• 홍은희

- 데이터 사이언티스트 및 통계학 석사
- 서울대학교
- 이메일: evelynwoods90@gmail.com

• 전기

저는 서울대학교에서 통계학 석사를 마치고 현대자동차, SK, 정부출연연구소 등 여러 산업 분야에서 데이터 사이언티스트로 활동했고 현재도 활발히 데이터 분석 업무를 수행하고 있습니다.

데이터 분석 플랫폼과 빅 데이터를 다루기 위한 다양한 프로그래밍 언어에 관심이 있습니다.

이 책은 실제 데이터 분석을 수행하면서 마주칠 수 있는 다양한 주제에 대해 꼭넓게 다루고 있습니다. 이 책이 여러분께 많은 도움이 되길 바랍니다.

1.2 튜토리얼 동기

저는 IMA 데이터 과학 펠로우십 프로젝트를 통해 PySpark를 배우는 동기를 얻었습니다. 그 후 저는 PySpark에 감명을 받았고 끌렸습니다. 그리고 저는 다음과 같은 것을 발견했습니다:

1. Spark는 가장 강력한 빅 데이터 도구라고 해도 과언이 아닙니다.
2. 하지만, 저는 스파크를 배우는 것이 여전히 어렵다는 것을 발견했습니다. Pyspark를 구글에 검색해서 나오는 결과들 중 어떤 것이 사실인지 확인해야 했습니다. 그리고 하나의 파일에서 전체 과정을 쉽게 배울 수 있는 자세한 예를 찾기 어려웠습니다.
3. 좋은 소스는 대학원생에게는 비쌉니다.

1.3 저작권 고지 및 라이선스 정보

파이썬으로 아파치 스파크 학습하기 PDF 파일은 무료로 사용할 수 있는 문서이며, 따라서 이 파일의 소스는 <https://runawayhorse001.github.io/LearningApacheSpark/pyspark.pdf>에서 온라인으로 사용할 수 있습니다. 그러나 이 문서는 MIT 라이선스와 Creative Commons Attribution-NonCommercial 2.0 Generic (CC BY-NC 2.0) 라이선스에 따라 라이선스가 부여됩니다.

여러분이 라이선스를 사용, 복사, 수정, 병합, 게시, 배포 또는 하위 라이선스를 사용할 계획인 경우 자세한 내용은 해당 라이선스의 조건을 참조해서 저자에게 상응하는 크레딧을 제공하시길 바랍니다.

1.4 감사글

저는 뉴스빌에 테네시 대학에서 밍 첸(Ming Chen), 지안 선(Jian Sun), 중보 리(Zhongbo Li)에게 귀중한 토론을 해주신 것과 자세한 솔루션들과 인터넷에 소스 코드를 제공해주신 아낌 없는 익명의 저자들에게 감사를 표합니다. 그들의 도움이 없이 이 저장소를 만들지 못했을 것입니다. 또한 미네소타 대학교 트윈 시티즈의 수학 및 응용 연구소(IMA)가 IMA 데이터 사이언스 연구원 시절 지원해 준 것에 감사하고 TAN Thiham Huat와 Mark Rabins가 오타를 찾아준 것에 감사드리는 바입니다.

셰필드 대학 컴퓨터 과학과의 기계 학습 강사인 하이핑 루 박사가(Dr. Haiping Lu)께서 제 튜토리얼을 자신의 수업 시간에 추천하고 많이 사용해 주신 것과 귀중한 제안에 대해 특별한 감사를 드립니다.

1.5 피드백 및 제안

여러분의 의견과 제안은 매우 감사합니다.

개선 사항에 대한 수정, 제안 또는 피드백을 이메일(von198@gmail.com)을 통해 받게 되어 매우 기쁩니다.

왜 파이썬으로 스파크를 사용할까?

중국 속담

칼을 더 길게 갈면 장작을 쉽게 짜를 수 있다 – 중국 옛 속담

저는 다음 두 가지 질문에 대한 답을 하고 싶습니다:

2.1 왜 스파크인가?

Apache Spark™ 공식 웹 사이트의 다음과 같은 네 가지 주요 이유가 스파크를 사용하도록 설득하는 데 충분하다고 생각합니다.

1. 속도

메모리에서 Hadoop MapReduce보다 최대 100배 더 빠르게 또는 디스크에서 10배 더 빠르게 프로그램을 실행할 수 있습니다.

Apache Spark에는 주기적인 데이터 흐름과 메모리 내 컴퓨팅을 지원하는 고급 DAG 실행 엔진이 있습니다.

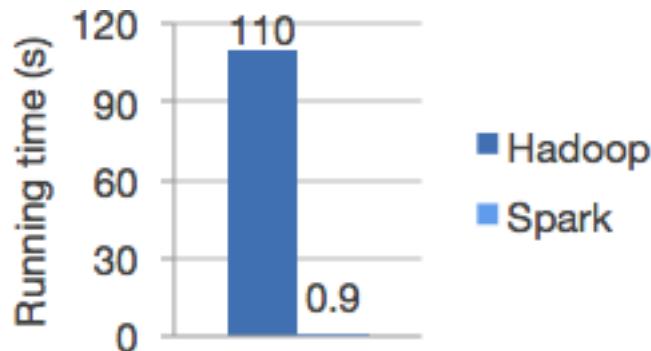


그림. 1: Hadoop과 Spark의 로지스틱 회귀 분석

2. 사용 편의성

Java, Scala, Python, R에서 애플리케이션을 빠르게 작성할 수 있습니다.

스파크는 병렬 앱을 쉽게 구축할 수 있는 80개 이상의 고급 운영자를 제공합니다. 그리고 스칼라, 파이썬 및 R 셸에서 대화식으로 사용할 수 있습니다.

3. 일반성

SQL, 스트리밍 및 복잡한 분석을 결합합니다.

스파크는 SQL 및 데이터 프레임, 머신 러닝용 MLlib, GraphX 및 스파크 스트리밍을 포함한 라이브러리 스택에 전원을 공급합니다. 이 라이브러리들을 동일한 애플리케이션에서 원활하게 결합할 수 있습니다.

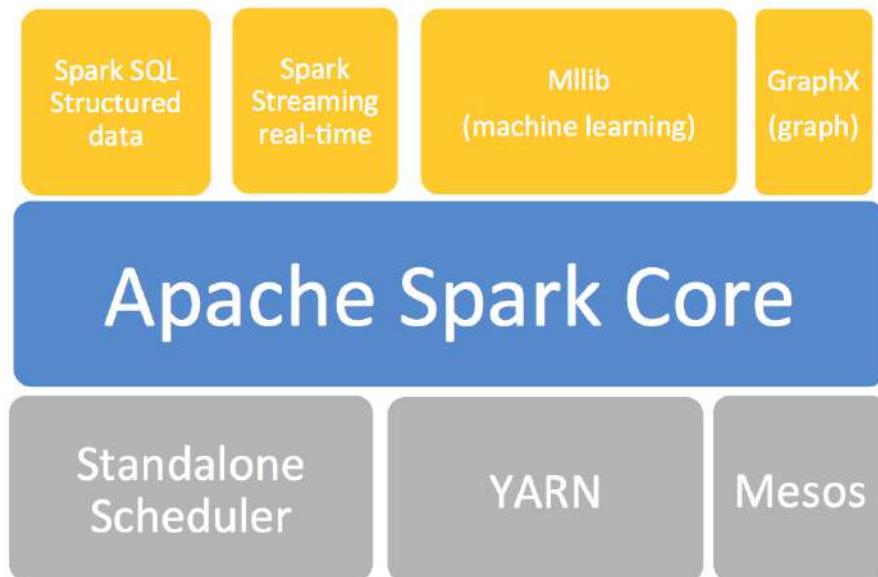


그림. 2: 스파크 스택

4. 어디에서나 실행 가능

스파크는 하둡, Mesos, Standalone 혹은 클라우드에서 실행됩니다. HDFS, 카산드라, Hbase 그리고 S3를 포함해 다양한 데이터 소스에 접근할 수 있습니다.



그림. 3: 스파크 플랫폼

2.2 왜 파이썬으로 스파크를(PySpark) 사용할까?

파이썬은 가장 인기 있는 프로그래밍 언어 중 하나입니다.

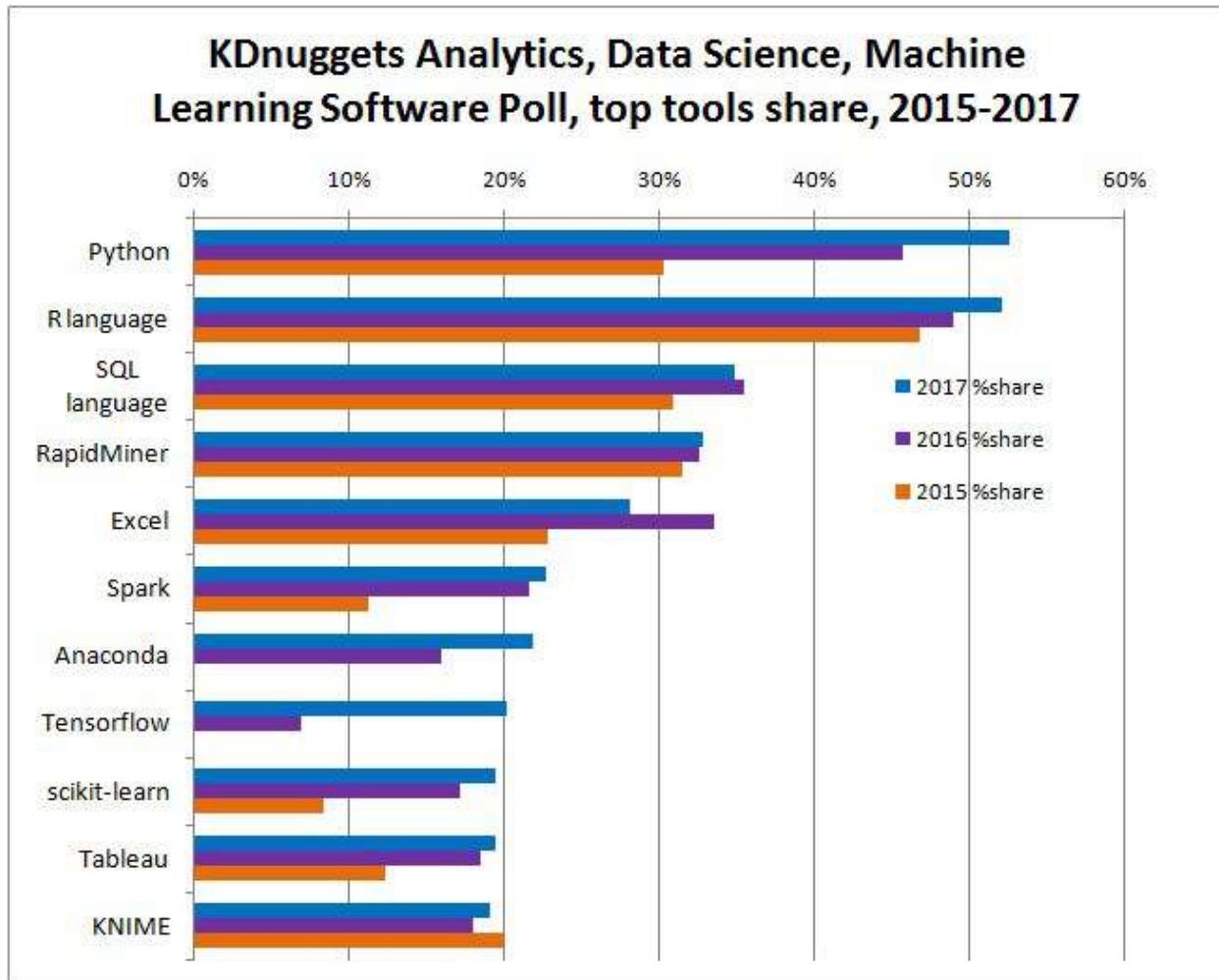


그림. 4: kd nuggets의 Kdnuggets 분석/데이터 사이언스 2017 소프트웨어 투표

실행 플랫폼 환경 설정

중국 속담

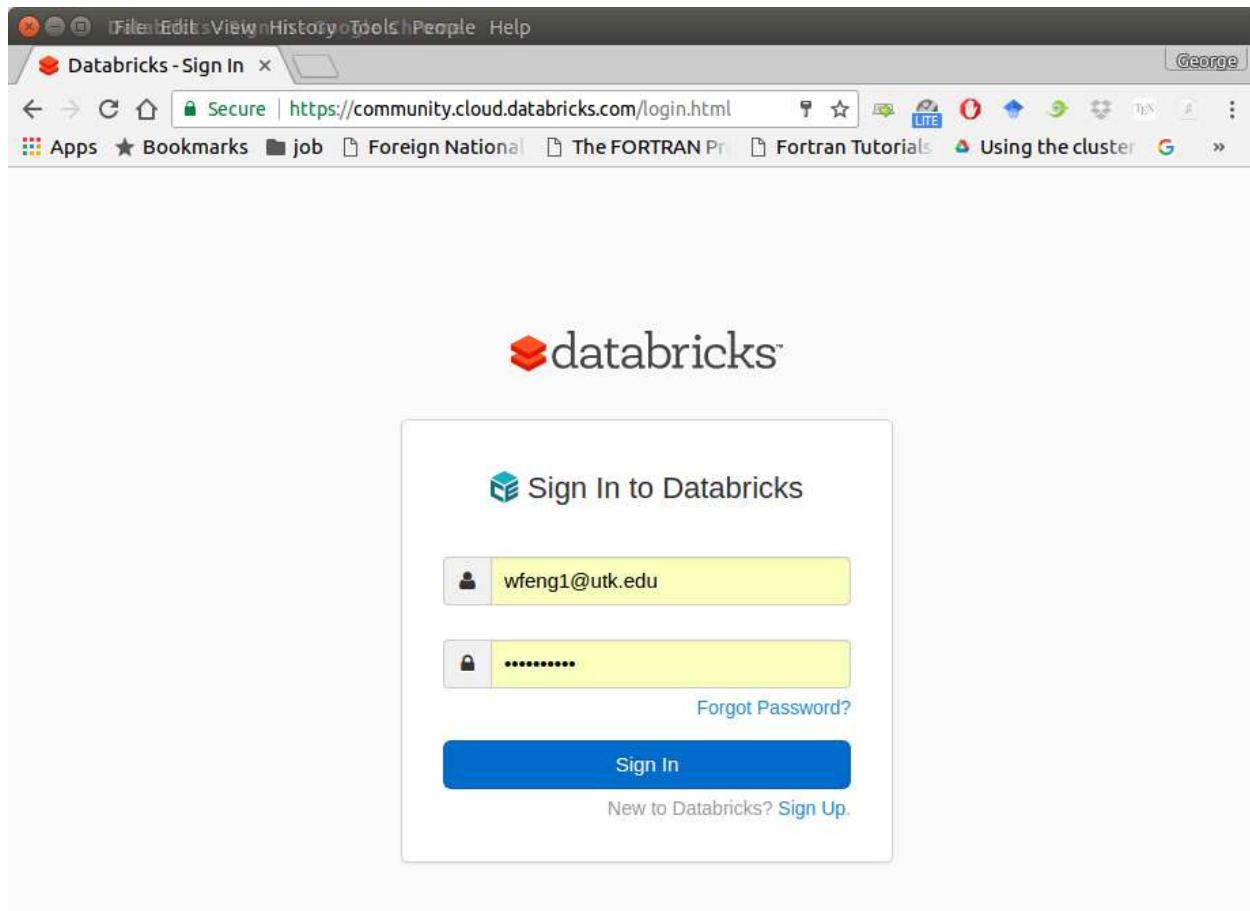
좋은 도구는 일을 성공적으로 수행하기 위한 전제조건입니다 - 중국 옛 속담

좋은 프로그래밍 플랫폼을 사용하면 많은 문제와 시간을 절약할 수 있습니다. 여기서는 제가 좋아하는 프로그래밍 플랫폼을 설치하는 방법을 제안하고 리눅스 시스템에서 이를 설치하는 가장 쉬운 방법을 보여드리겠습니다. 여러분이 다른 운영 시스템에서 이를 설치하고 싶으시다면 구글에 방법을 검색해 보실 수 있습니다. 이번 섹션에서 여러분은 해당 프로그래밍 플랫폼과 패키지에 PySpark를 설치하는 방법을 배울 수 있습니다.

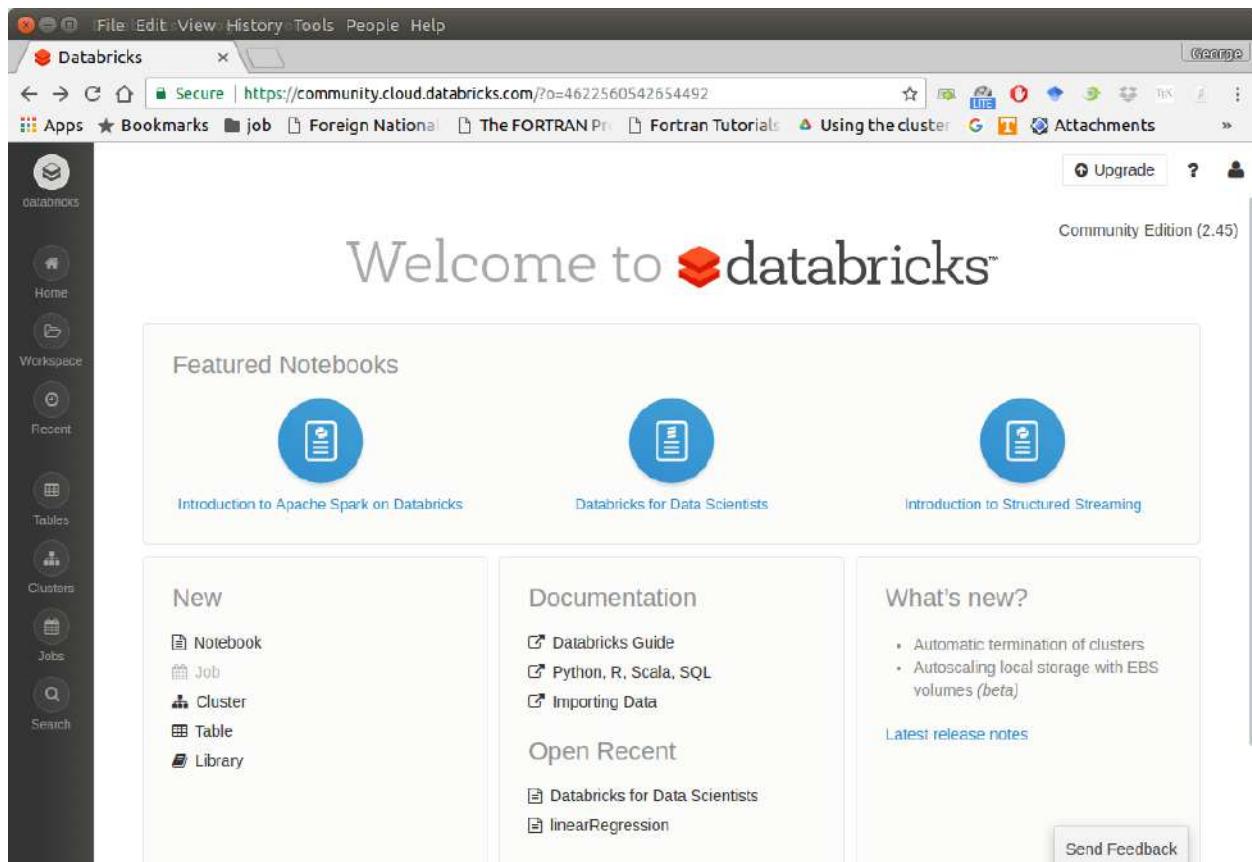
3.1 Databricks 커뮤니티 클라우드에서 실행하기

리눅스 또는 유닉스 운영자 시스템에 대한 경험이 없다면 Databricks 커뮤니티 클라우드에서 스파크를 사용하는 것을 추천합니다. 스파크를 설정할 필요가 없고 커뮤니티 에디션의 경우 완전히 무료입니다. 아래 단계를 따르세요.

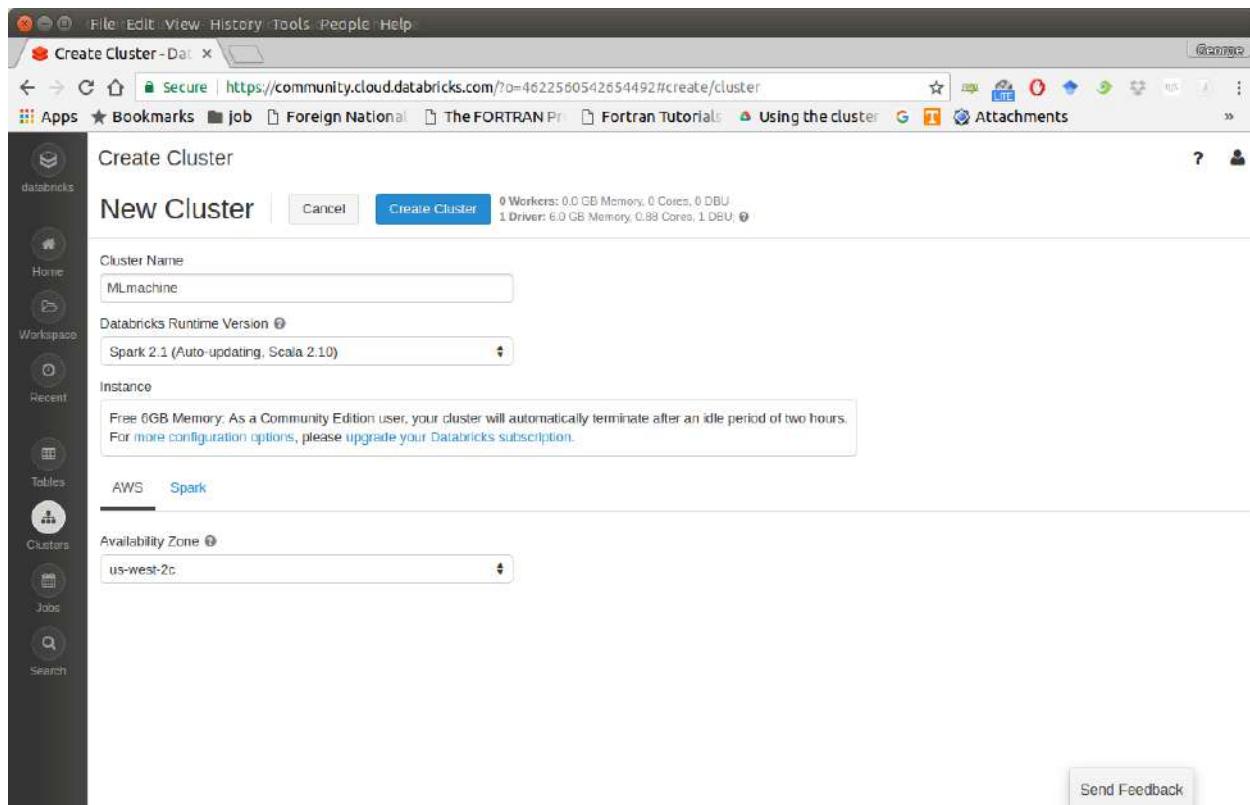
1. 계정을 등록하세요: <https://community.cloud.databricks.com/login.html>



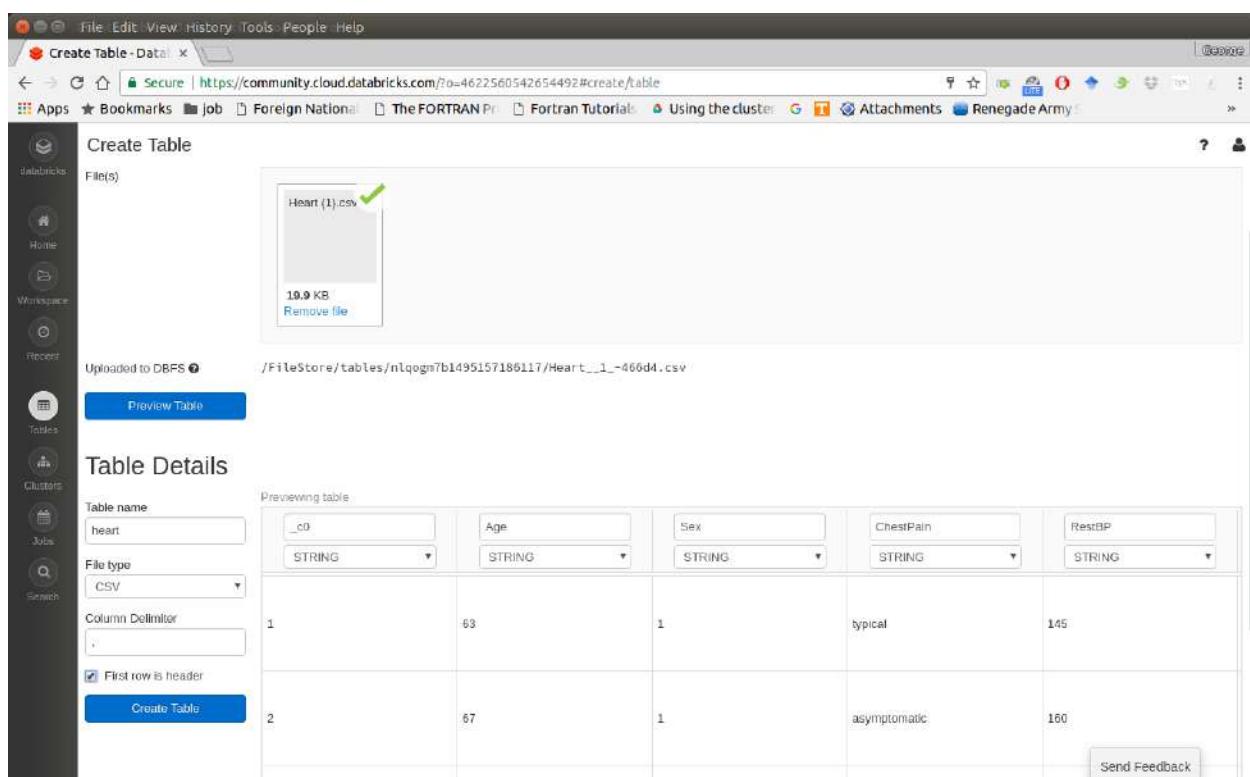
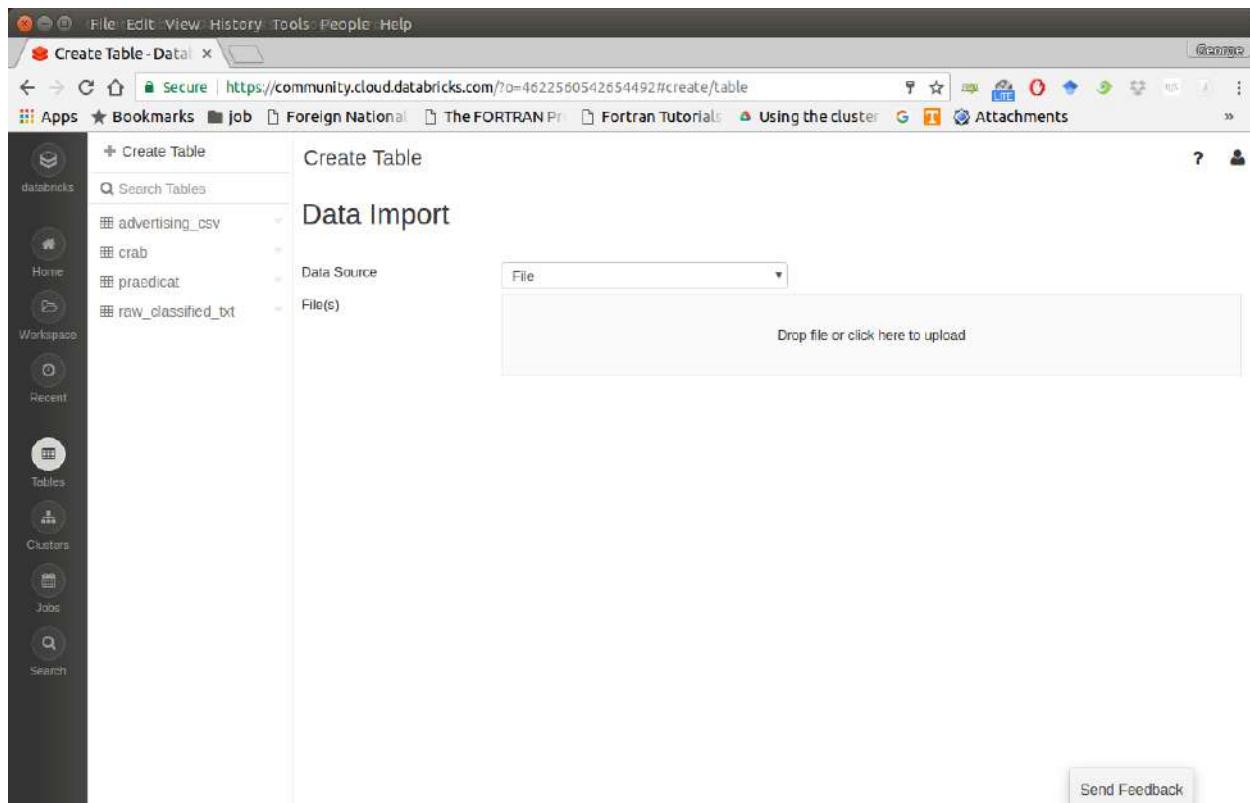
2. 계정으로 로그인한 후 여러분의 클러스터(머신), 테이블(데이터 셋), 노트북(코드)를 생성할 수 있습니다.



3. 코드가 실행될 클러스터를 생성합니다.

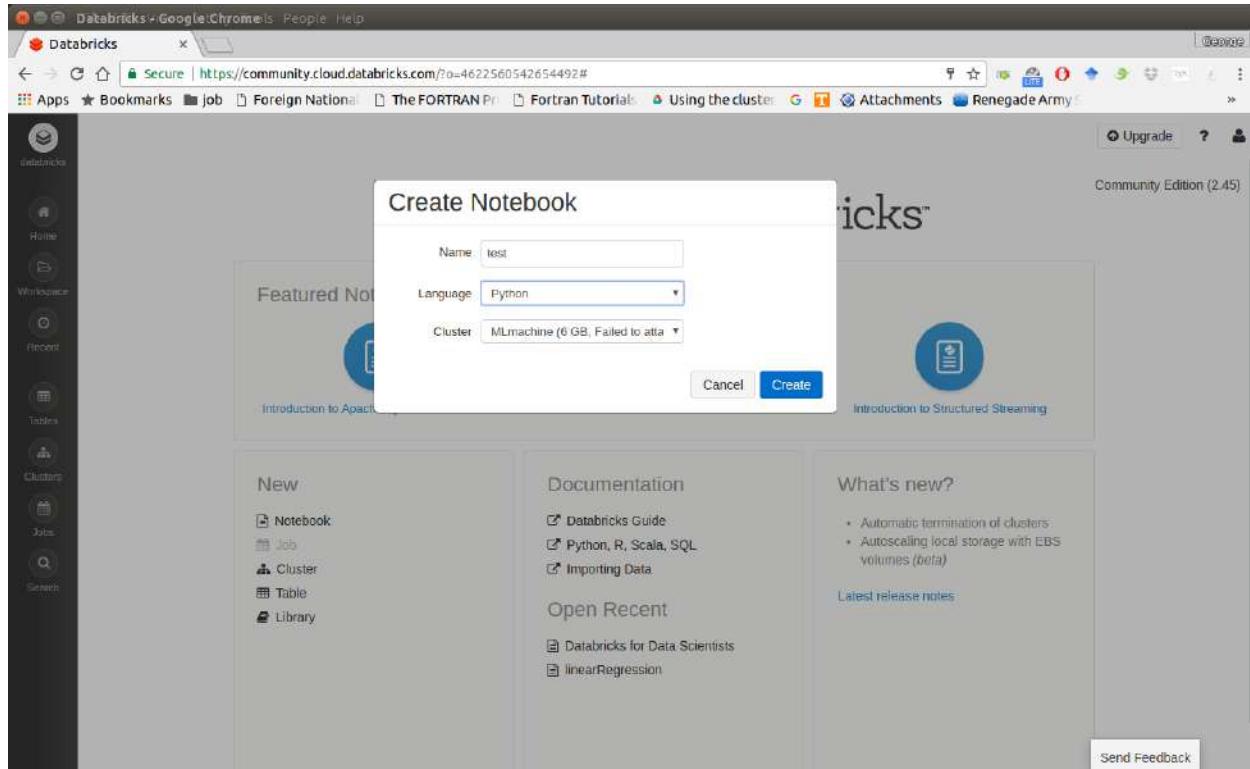


4. 여러분의 데이터셋을 가져옵니다.



노트: Uploaded to DBFS에 나타난 주소: /FileStore/tables/05mhqy14896873710/ 를 저장해야 합니다. 데이터 셋을 로드하기 위해 이 경로를 사용할 것이기 때문입니다.

5. 여러분의 노트북을 생성합니다.



```

1. Linear Regression with PySpark on Databricks
Author: Wenqiang Feng

Set up SparkSession
from pyspark.sql import SparkSession
spark = SparkSession \
    .builder \
    .appName("python Spark Linear Regression Example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

2. Load dataset
df = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', \
            inferSchema='true') \
    .load("/FileStore/tables/05rmhuqv1489687378010/", header= True)

```

위의 5단계를 완료하면 Databricks 커뮤니티 클라우드에서 Spark 코드를 실행할 준비가 되었습니다. Databricks 커뮤니티 클라우드에서 다음 데모를 모두 실행하겠습니다. 여러분이 데모 코드를 실행했을 때 여러분은 다음과 같은 결과들을 얻게 될 것입니다.

```

+---+---+---+---+
| _c0 | TV | Radio | Newspaper | Sales |
+---+---+---+---+
| 1 | 230.1 | 37.8 | 69.2 | 22.1 |
| 2 | 44.5 | 39.3 | 45.1 | 10.4 |
| 3 | 17.2 | 45.9 | 69.3 | 9.3 |
| 4 | 151.5 | 41.3 | 58.5 | 18.5 |
| 5 | 180.8 | 10.8 | 58.4 | 12.9 |
+---+---+---+---+
only showing top 5 rows

root
|-- _c0: integer (nullable = true)
|-- TV: double (nullable = true)
|-- Radio: double (nullable = true)
|-- Newspaper: double (nullable = true)
|-- Sales: double (nullable = true)

```

3.2 Mac과 Ubuntu에서 Spark 환경설정

3.2.1 필수 구성 요소 설치

Anaconda를 설치하는 것을 강력히 추천합니다. Anaconda에는 대부분의 필수 구성 요소를 포함하고 있고 다양한 운영 시스템을 지원하기 때문입니다.

1. Python 설치

Ubuntu 소프트웨어 센터로 이동하여 다음 단계를 따릅니다:

- Ubuntu 소프트웨어 센터를 엽니다
- 파이썬을 검색합니다
- 설치를 클릭합니다

또는 터미널을 열고 다음 명령을 사용합니다:

```
sudo apt-get install build-essential checkinstall  
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev  
      libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev  
sudo apt-get install python  
sudo easy_install pip  
sudo pip install ipython
```

3.2.2 자바 설치

자바는 많은 다른 소프트웨어들에 의해 사용됩니다. 그래서 여러분은 이미 자바를 설치했을 가능성이 있습니다. 여러분은 명령 프롬프트에서 다음 명령을 사용할 수 있습니다:

```
java -version
```

그렇지 않으면 Mac에 Java를 설치하려면 어떻게 해야 합니까?의 단계에 따라 Java를 Mac에 설치하고 명령 프롬프트에서 Ubuntu를 설치하기 위해 다음 명령을 수행합니다:

```
sudo apt-add-repository ppa:webupd8team/java  
sudo apt-get update  
sudo apt-get install oracle-java8-installer
```

3.2.3 Java SE Runtime Environment 설치하기

저는 오라클 Java JDK를 설치했습니다.

경고: Java 및 Java SE Runtime Environment 단계를 설치하는 것은 매우 중요합니다, 왜냐하면 Spark는 Java로 작성된 특정 도메인(domain-specific) 언어이기 때문입니다.

명령 프롬프트에서 다음 명령을 사용하여 Java를 사용할 수 있는지 확인하고 해당 버전을 찾을 수 있습니다:

```
java -version
```

Java가 성공적으로 설치되면 다음과 비슷한 결과를 얻을 수 있습니다:

```
java version "1.8.0_131"
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)
```

3.2.4 아파치 스파크(Apache Spark) 설치

사실 Pre-build 버전은 설치할 필요가 없습니다. 여러분은 Pre-build 버전을 언팩(unpack)할 때 해당 버전을 사용할 수 있습니다.

- 다운로드: [Download Apache Spark™](#) 다운로드에서 사전에 구축된 Apache Spark™ 을(를) 얻을 수 있습니다.
- 언팩(Unpack): 여러분이 Spark를 설치하기 원하는 경로에 Apache Spark™를 언팩(Unpack) 합니다.
- 테스트: 필수 구성 요소를 테스트합니다: 위치를 spark-#.#.#+bin-hadoop#.#/bin로 변경하고 실행합니다.

```
./pyspark
```

```
Python 2.7.13 |Anaconda 4.4.0 (x86_64) | (default, Dec 20 2016, 23:05:08)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR,
use setLogLevel(newLevel).
17/08/30 13:30:12 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/08/30 13:30:17 WARN ObjectStore: Failed to get database global_temp,
returning NoSuchObjectException
Welcome to

    /__/
   _\ \_ \_ \_ `/_ /' _/ \
  /__ / .__/_\_,/_/_/ /_/\_\_ \
 /_/_/                                         version 2.1.1

Using Python version 2.7.13 (default, Dec 20 2016 23:05:08)
SparkSession available as 'spark'.
```

3.2.5 스파크(Spark) 환경 설정

a. Mac 운영 시스템 : 터미널에서 여러분의 bash_profile 을 열어보세요

```
vim ~/.bash_profile
```

여러분의 bash_profile에서 다음 라인들을 추가하세요(SPARK_HOME 경로를 여러분이 스파크를 설치한 경로로 변경하시길 바랍니다)

```
# add for spark
export SPARK_HOME=your_spark_installation_path
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PATH=$PATH:$SPARK_HOME/bin
export PYSPARK_DRIVER_PYTHON="jupyter"
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
```

마지막으로 수정된 bash_profile을 바로 적용하는 해야함을 명심하세요

```
source ~/.bash_profile
```

b. Ubuntu 운영 시스템 : 터미널에서 여러분의 bashrc 을 열어보세요

```
vim ~/.bashrc
```

여러분의 bashrc에서 다음 라인들을 추가하세요(SPARK_HOME 경로를 여러분이 스파크를 설치한 경로로 변경하시길 바랍니다)

```
# add for spark
export SPARK_HOME=your_spark_installation_path
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PATH=$PATH:$SPARK_HOME/bin
export PYSPARK_DRIVER_PYTHON="jupyter"
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
```

마지막으로 수정된 bashrc을 바로 적용하는 해야하는 것을 명심하세요

```
source ~/.bashrc
```

3.3 Windows에서 Spark 환경 설정

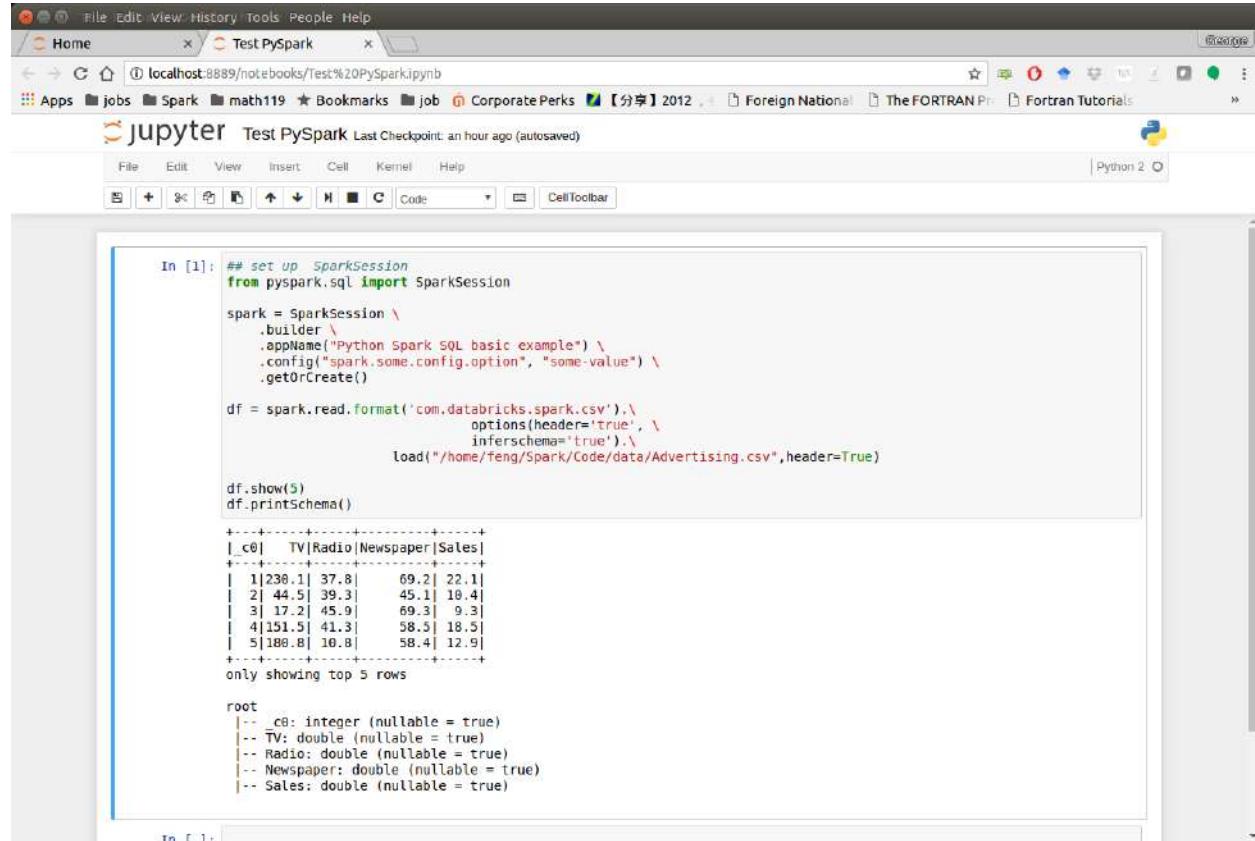
윈도우에 오픈 소스 소프트웨어를 설치하는 것은 항상 악몽입니다. 딜레시 맨들로이(Deelesh Mandloi)에게 감사합니다.

여러분은 Windows 운영 체제에서 Apache Spark™를 설치하기 위해 Windows에서 PySpark 시작하기 블로그의 자세한 절차에 따라 설치할 수 있습니다.

3.4 텍스트 편집기 혹은 IDE에서의 PySpark

3.4.1 주피터 노트북 (Jupyter Notebook)에서의 PySpark

*Mac 및 Ubuntu*에서 *스파크 구성*에서 위의 설치 단계를 완료한 후, 여러분은 주피터 노트북에서 PySpark 코드를 작성하고 실행해야 합니다.



The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** File Edit View History Tools People Help
- Address Bar:** Home Test PySpark localhost:8889/notebooks/Test%20PySpark.ipynb
- Toolbar:** File Edit View Insert Cell Kernel Help
- Cell Content:**

```
In [1]: ## set up SparkSession
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

df = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', \
            inferSchema='true') \
    .load("/home/feng/Spark/Code/data/Advertising.csv",header=True)

df.show(5)
df.printSchema()
```
- Output:**

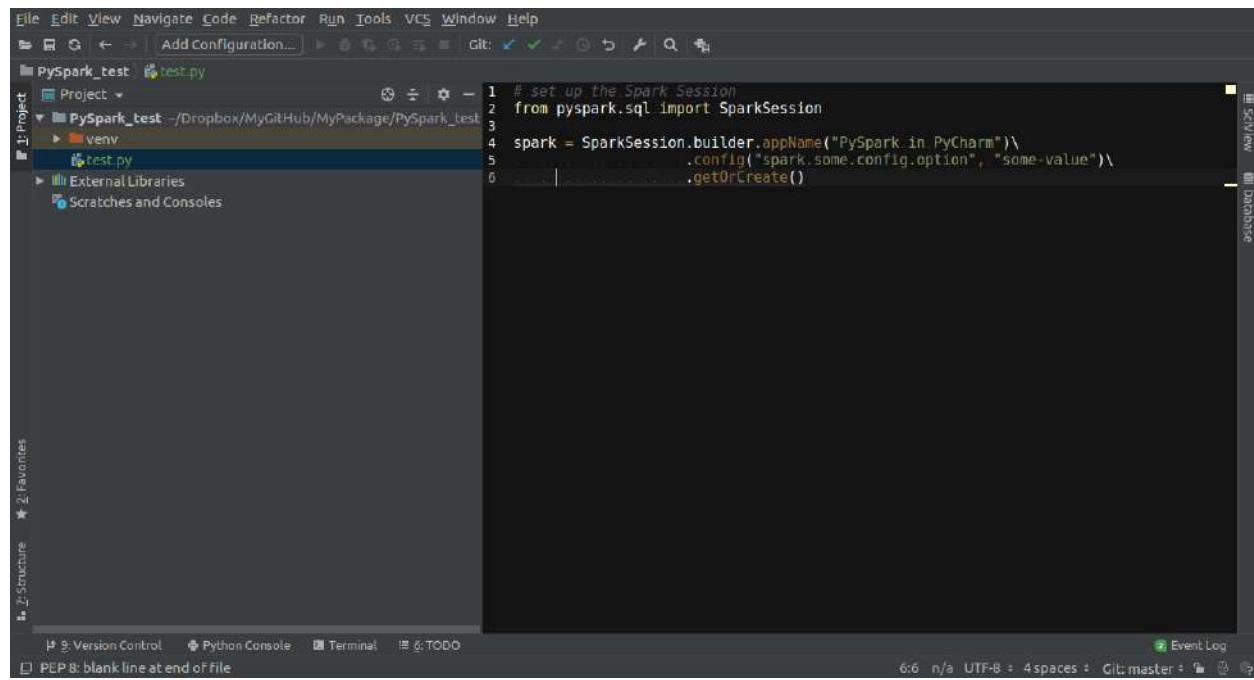
```
+---+---+---+---+
|_c0| TV|Radio|Newspaper|Sales|
+---+---+---+---+
| 1|230.1| 37.8|   69.2| 22.1|
| 2| 44.5| 39.3|   45.1| 10.4|
| 3| 17.2| 45.9|   69.3|  9.3|
| 4|151.5| 41.3|   58.5| 18.5|
| 5|180.8| 10.8|   58.4| 12.9|
+---+---+---+---+
only showing top 5 rows

root
 |-- _c0: integer (nullable = true)
 |-- TV: double (nullable = true)
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)
```

3.4.2 PyCharm에서 PySpark 사용하기

*Mac 및 Ubuntu*에서 *스파크 구성*에서 위의 설치 단계를 완료한 후, 여러분은 PyCharm 프로젝트에 PySpark를 추가해야 합니다.

1. 새로운 PyCharm 프로젝트를 생성하세요



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The toolbar has icons for file operations like Open, Save, and Run. The left sidebar shows the Project structure with a 'PySpark_test' project containing a 'venv' folder and a 'test.py' file. The main editor window displays the following Python code:

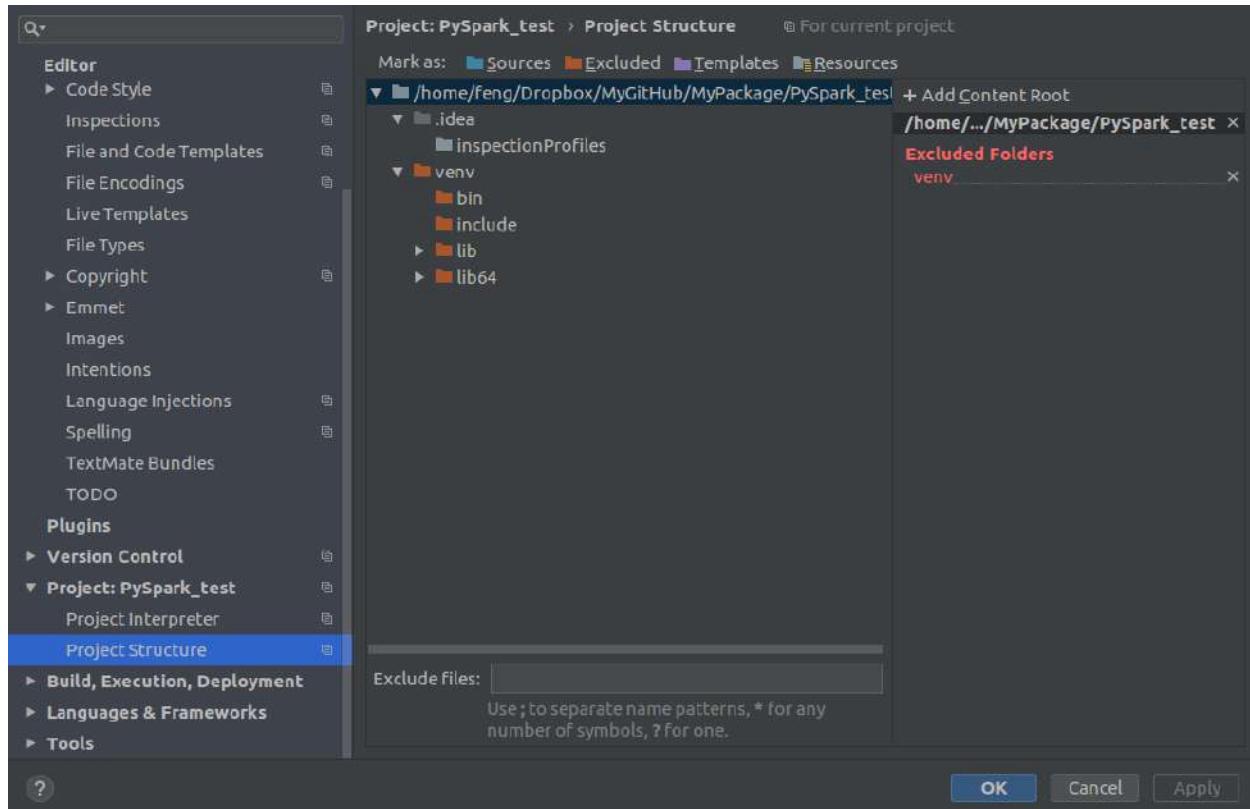
```
# set up the Spark Session
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("PySpark in PyCharm")\
    .config("spark.some.config.option", "some-value")\
    .getOrCreate()
```

The bottom status bar shows the current file is 'test.py', the encoding is UTF-8, there are 4 spaces, and the Git status is 'master'. Other tabs in the status bar include Version Control, Python Console, Terminal, and TODO.

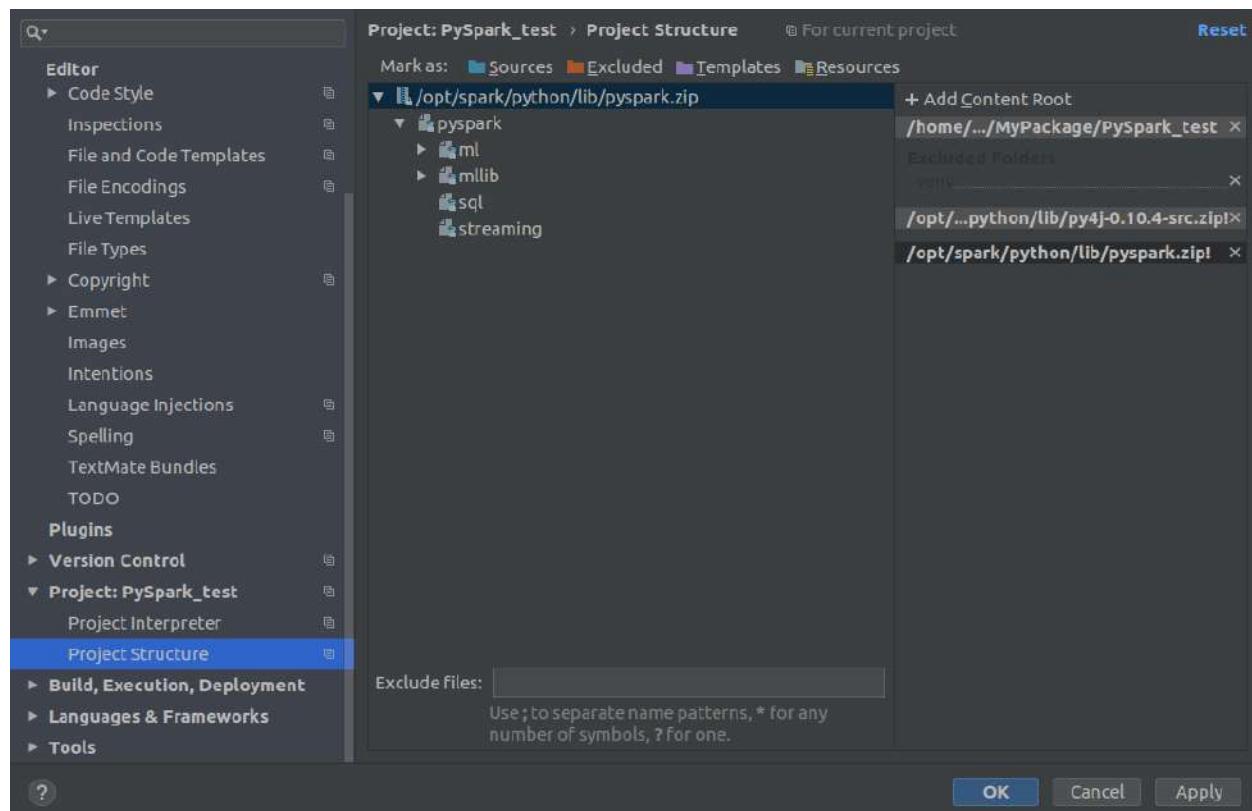
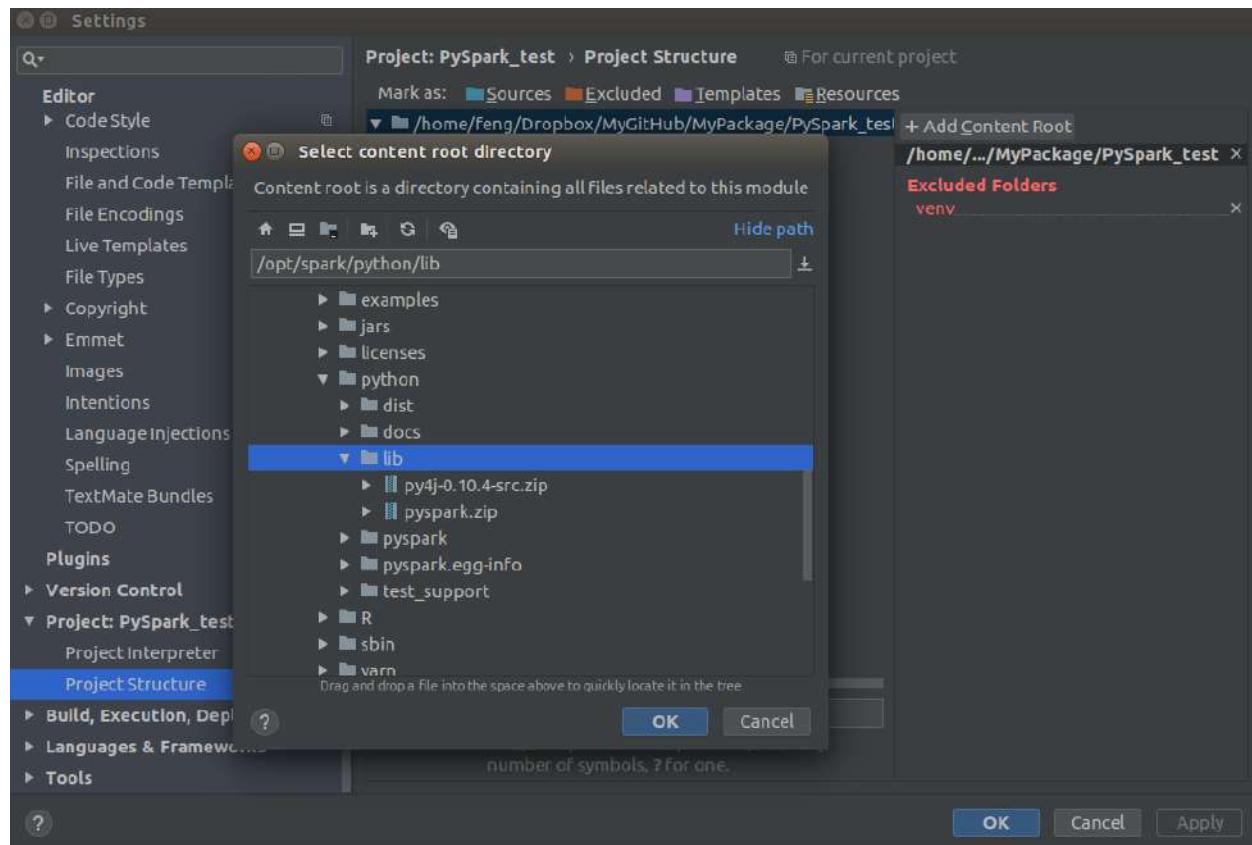
2. Project Structure로 이동합니다

옵션 1: File -> Settings -> Project: -> Project Structure

옵션 2: PyCharm -> Preferences -> Project: -> Project Structure



3. 컨텐츠 루트(Content Root)를 추가하세요: \$SPARK_HOME/python/lib의 모든 ZIP 파일



4. 여러분의 스크립트를 실행하세요

```

1 # set up the Spark Session
2 from pyspark.sql import SparkSession
3
4 spark = SparkSession.builder.appName("PySpark in PyCharm")\
5     .config("spark.some.config.option", "some-value")\
6     .getOrCreate()
7
8 # load data
9 data = spark.read.format('com.databricks.spark.csv')\
10    .options(header='true', inferSchema='true')\
11    .load("Heart.csv", header=True)
12
13 # show data
14 data.show(5)

```

The Run tab shows the output:

Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
63	1	typical	145	233	1	2	156	0	2.3	3	0	fixed	No
67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3	normal	Yes
67	1	asymptomatic	126	229	0	2	129	1	2.6	2	2	reversible	Yes
37	1	nonanginal	136	250	0	0	187	0	3.5	3	0	normal	No
41	0	nontypical	138	204	0	2	172	0	1.4	1	0	normal	No

only showing top 5 rows

Process finished with exit code 0

3.4.3 아파치 제플린(Apache Zeppelin)에서 PySpark 사용하기

*Mac 및 Ubuntu*에서 스파크 구성에서 위의 설치 단계를 완료한 후, 여러분은 Apache Zeppelin에서 PySpark 코드를 작성하고 실행해야 합니다.

The screenshot shows a Zeppelin Notebook interface running on localhost:8080. The notebook has a blue header bar with tabs for 'Notebook' and 'Job'. Below the header, there's a search bar and a dropdown for 'anonymous'. The main area contains several code cells and their corresponding visual outputs.

- Code Cell 1:** PySpark code to load a CSV file and show the first 4 rows.


```
df = spark.read.format('com.databricks.spark.csv')\
    .options(header='true', \
    inferSchema='true')\
    .load('/home/feng/Dropbox/MyTutorial/LearningApacheSpark/doc/data/bank.csv',header=True);
```

Output: FINISHED
Took 0 sec. Last updated by anonymous at September 24 2017, 4:03:16 PM. (updated)

```
%spark.pyspark
df.show(4)
```

age	job marital education default balance housing loan contact day month duration campaign pdays previous poutcome y
30 unemployed primary no no no cellular 19 act 79 1 -1 0 unknown no	
33 services married secondary yes yes cellular 11 may 230 1 339 4 failure no	
35 management single tertiary no 1350 yes no cellular 16 apr 185 1 330 1 failure no	
38 management married tertiary no 1476 yes yes unknown 3 jun 199 4 -1 0 unknown no	

only showing top 4 rows

Took 1 sec. Last updated by anonymous at September 24 2017, 4:14:43 PM.
- Code Cell 2:** PySpark code to register a temporary table named 'bank'.


```
%spark.pyspark
df.registerTempTable("bank")
```

Output: FINISHED
Took 0 sec. Last updated by anonymous at September 24 2017, 4:03:27 PM. (updated)
- Code Cell 3:** SQL query to find the count of people in each age group where age is less than or equal to 30.


```
%%sql
select age, count(*) value
from bank
where age <= 30
group by age
order by age
```

Output: FINISHED
Took 0 sec. Last updated by anonymous at September 24 2017, 4:11:05 PM.

maxAge: 30

age distribution pie chart (values: 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29)
- Code Cell 4:** SQL query to find the count of people in each age group where marital status is single, divorced, or married.


```
%%sql
select age, count(*) value
from bank
where marital in ("single","divorced","married")
group by age
order by age
```

Output: FINISHED
Took 1 sec. Last updated by anonymous at September 24 2017, 4:03:35 PM.

marital: single

marital status pie chart (values: 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29)

age distribution histogram (Stacked, Stream, Expanded, value)

3.4.4 Sublime text와 함께 사용하는 PySpark

Mac 및 Ubuntu에서 스파크 구성에서 위의 설치 단계를 완료한 후에, 여러분은 PySpark 코드를 작성하기 위해 Sublime Text를 사용하고 터미널에서 보통의 파이썬 코드처럼 여러분의 코드를 실행해야 합니다.

```
python test_pyspark.py
```

그런 다음 터미널에서 출력 결과를 얻어야 합니다.

```

feng@feng-ThinkPad:~/Spark/Code
to bind to another address
17/05/21 19:12:47 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
17/05/21 19:12:47 WARN Utils: Service 'sparkUI' could not bind on port 4041. Attempting port 4042.
+---+---+---+---+---+
|_c0| TV|Radio|Newspaper|Sales|
+---+---+---+---+---+
| 1|230.1| 37.8|   69.2| 22.1|
| 2| 44.5| 39.3|   45.1| 16.4|
| 3| 17.2| 45.9|   69.3|  9.3|
| 4|151.5| 41.3|   58.5| 18.5|
| 5|180.8| 10.8|   58.4| 12.9|
+---+---+---+---+
only showing top 5 rows

root
|-- _c0: integer (nullable = true)
|-- TV: double (nullable = true)
|-- Radio: double (nullable = true)
|-- Newspaper: double (nullable = true)
|-- Sales: double (nullable = true)

feng@feng-ThinkPad:~/spark/code$ 

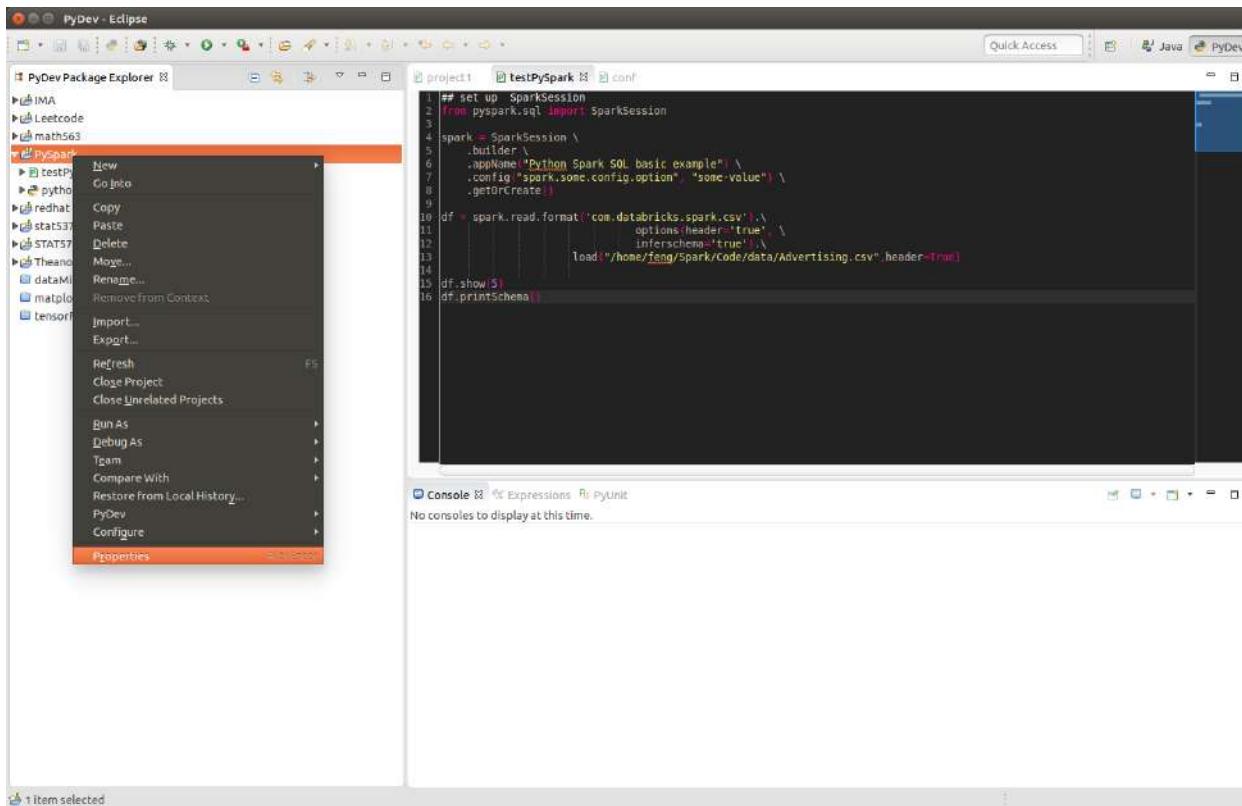
```

The screenshot shows a terminal window running on a ThinkPad. It displays the output of a PySpark session. The session starts with some warning messages about port binding. Then it shows the schema of a DataFrame named 'df' with columns '_c0', 'TV', 'Radio', 'Newspaper', and 'Sales'. Finally, it prints the top 5 rows of the DataFrame.

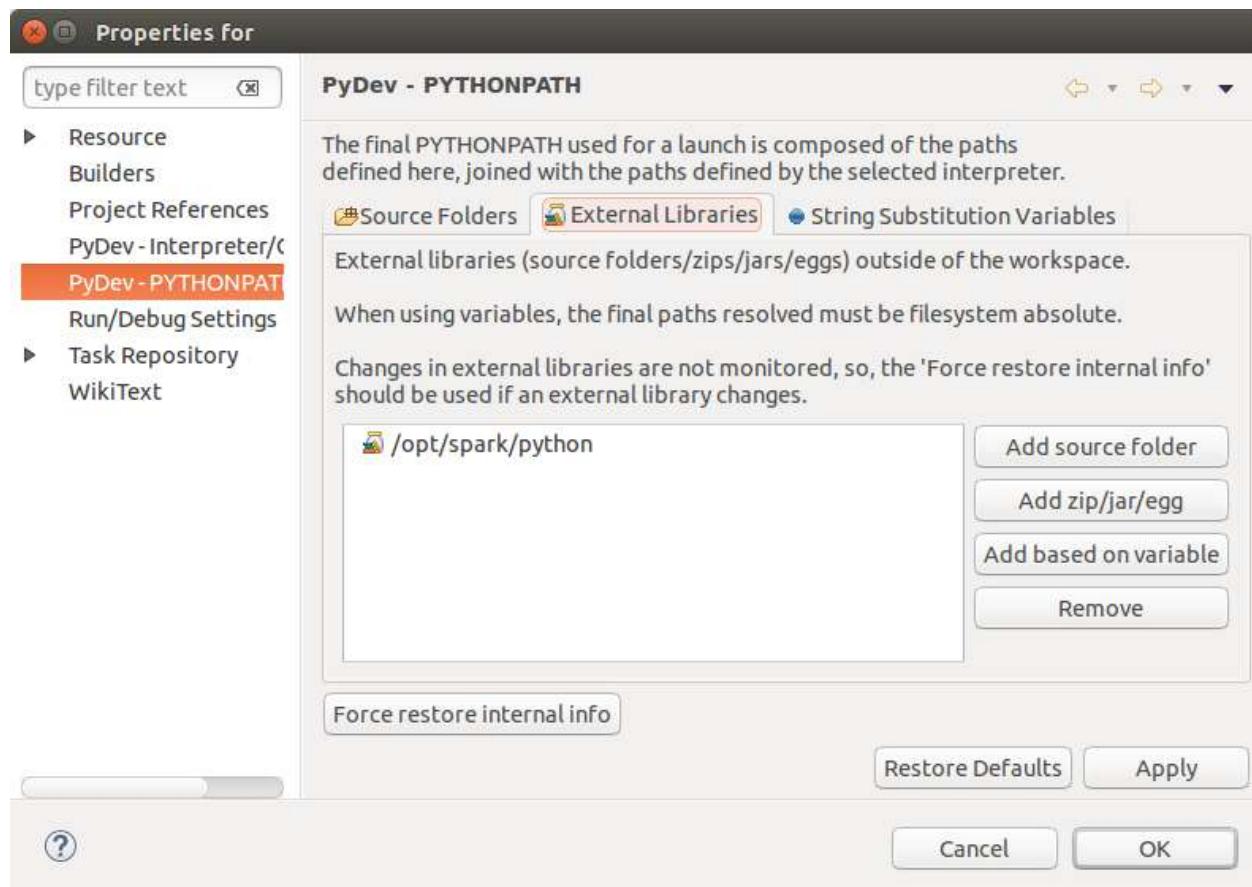
3.4.5 Eclipse에서 PySpark 사용하기

Eclipse에서 PySpark 코드를 실행하려면 다음과 같이 Current Project와 External Libraries에 대한 경로를 추가해야 합니다:

1. 프로젝트의 속성 (Properties) 열기



2. External Libraries에 경로 추가하기



그리고 나서 여러분은 이클립스에서 PyDev 플러그인을 통해 여러분의 코드를 실행해야 합니다.

```

## set up SparkSession
from pyspark.sql import SparkSession
spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

df = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', \
    inferSchema='true') \
    .load("/home/teg/Spark/Code/data/Advertising.csv".header=True)

df.show(5)
df.printSchema()

```

Console > PyConsole

```

<terminated> /home/teg/Eclipseworkspace/PySpark/testPySpark.py
+---+---+---+
| c0 | TV | Radio | Newspaper | Sales |
+---+---+---+
| 1 | 230.1 | 37.8 | 69.2 | 22.1 |
| 2 | 44.5 | 39.3 | 45.1 | 18.4 |
| 3 | 17.2 | 45.9 | 69.3 | 9.3 |
| 4 | 151.5 | 41.3 | 58.5 | 18.5 |
| 5 | 180.8 | 10.8 | 58.4 | 12.9 |
+---+---+---+
only showing top 5 rows

root
+- c0: integer (nullable = true)
+- TV: double (nullable = true)
+- Radio: double (nullable = true)
+- Newspaper: double (nullable = true)
+- Sales: double (nullable = true)

```

3.5 PySparkling Water: Spark + H2O

1. Sparkling Water를 다운로드하세요: <https://s3.amazonaws.com/h2o-release/sparkling-water/rel-2.4/5/index.html>

2. PySparkling을 테스트하세요

```

unzip sparkling-water-2.4.5.zip
cd ~/sparkling-water-2.4.5/bin
./pysparkling

```

만약 여러분이 PySpark를 올바르게 설치했다면, 다음과 같은 결과를 얻게 될 것입니다:

```

Using Spark defined in the SPARK_HOME=/Users/dt216661/spark environmental
property

Python 3.7.1 (default, Dec 14 2018, 13:28:58)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
2019-02-15 14:08:30 WARN NativeCodeLoader:62 - Unable to load native-hadoop_
library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.  
→properties  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use  
→setLogLevel(newLevel).  
2019-02-15 14:08:31 WARN  Utils:66 - Service 'SparkUI' could not bind on port  
→4040. Attempting port 4041.  
2019-02-15 14:08:31 WARN  Utils:66 - Service 'SparkUI' could not bind on port  
→4041. Attempting port 4042.  
17/08/30 13:30:12 WARN NativeCodeLoader: Unable to load native-hadoop  
library for your platform... using builtin-java classes where applicable  
17/08/30 13:30:17 WARN ObjectStore: Failed to get database global_temp,  
returning NoSuchObjectException  
Welcome to  
  
Using Python version 3.7.1 (default, Dec 14 2018 13:28:58)  
SparkSession available as 'spark'.
```

3. 주피터 노트북에서 pysparkling을 설치하세요

다음 alias를 bashrc (Linux 시스템) 혹은 bash_profile (Mac 시스템)에 추가합니다

```
alias sparkling="PYSPARK_DRIVER_PYTHON='ipython' PYSPARK_DRIVER_PYTHON_OPTS=  
→ "notebook" ~/~/sparkling-water-2.4.5/bin/pysparkling"
```

4. 터미널에서 pysparkling을 여세요

```
sparkling
```

3.6 클라우드에서 스파크 설치하기

Mac 및 Ubuntu에서 스파크 구성에 설치 단계에 따라 여러분은 AWS, 구글 클라우드 등 클라우드에 자체 클러스터를 설정할 수 있습니다. 실제로 해당 클라우드의 경우 자체 빅 데이터 도구가 있습니다. 데이터브릭스 커뮤니티 클라우드와 마찬가지로 설정 없이 직접 실행할 수 있습니다. 자세한 내용은 언제든지 제게 문의하세요.

3.7 Colaboratory에서 Pyspark 사용하기

Colaboratory는 무료 Jupyter 노트북 환경으로 설정이 필요하지 않고 전적으로 클라우드에서 실행됩니다.

3.7.1 설치

```
!pip install pyspark
```

3.7.2 테스트

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark create RDD example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

df = spark.sparkContext \
    .parallelize([(1, 2, 3, 'a b c'), \
                 (4, 5, 6, 'd e f'), \
                 (7, 8, 9, 'g h i')]) \
    .toDF(['col1', 'col2', 'col3', 'col4'])

df.show()
```

출력:

col1	col2	col3	col4
1	2	3	a b c
4	5	6	d e f
7	8	9	g h i

3.8 데모 코드

Colab에 설치로부터 주피터 노트북을 다운로드 받을 수 있습니다.

- 파이썬 소스 코드

```
## SparkSession 설정
from pyspark.sql import SparkSession

spark = SparkSession \
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
.builder \
.appName("Python Spark SQL basic example") \
.config("spark.some.config.option", "some-value") \
.getOrCreate()

df = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', \
             inferSchema='true') \
    .load("/home/feng/Spark/Code/data/Advertising.csv"
         , header=True)

df.show(5)
df.printSchema()
```

아파치 스파크 개요

중국 속담

자신을 알고 적을 알면 절대 지지 않을 것 이다 – 손자병법

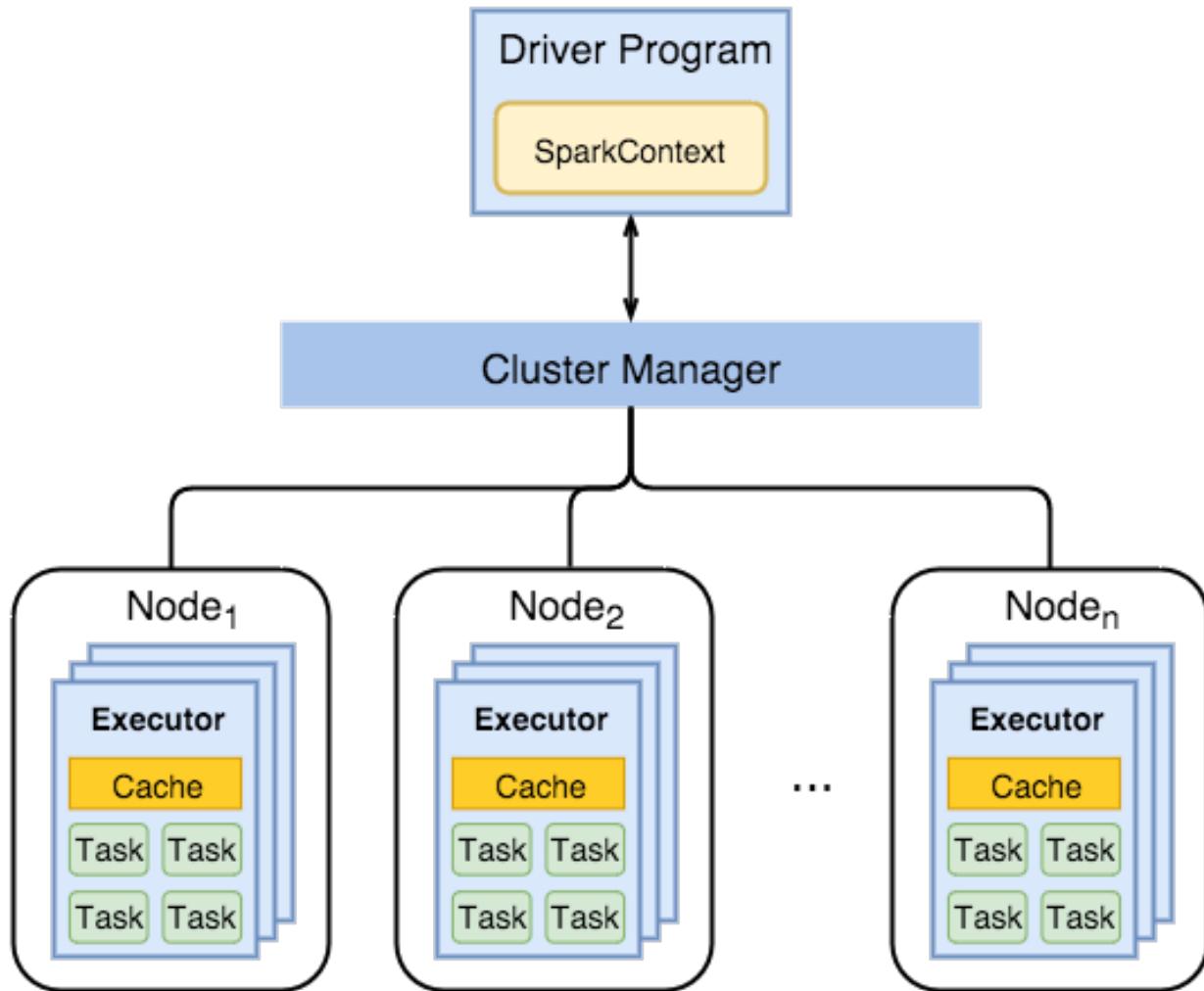
4.1 핵심 개념

다음 개념은 대부분 [Kirillov2016]에서 인용했습니다. 해당 저작권은 안톤 키릴로프(**Anton Kirillov**)에게 있습니다. 아파치 스파크의 핵심 개념, 아키텍처 및 내부로부터 더 자세한 정보를 얻을 수 있습니다.

아파치 스파크가 어떻게 작동하는지 자세히 살펴보기 전에 아파치 스파크의 전문 용어를 이해해 봅시다.

- 잡(Job): HDFS 또는 로컬에서 입력을 읽고 데이터에 대한 계산을 수행하고 출력 데이터를 쓰는 코드 조각입니다.
- 스테이지(Stages): 잡은 스테이지로 나뉩니다. 스테이지는 맵 혹은 리듀스 단계로 구분됩니다(여러분이 하둡에서 작업을 해본 적이 있고 하둡과 스파크의 연관성을 파악하고자 한다면 해당 개념을 더 쉽게 이해할 수 있습니다). 스테이지는 계산 경계를 기준으로 나뉘이며, 모든 계산(연산자)을 한 단계에서 업데이트할 수 없습니다. 이는 여러 스테이지에 걸쳐 이뤄지게 됩니다.
- 테스크(Tasks): 각 스테이지에는 파티션 당 하나의 테스크가 있습니다. 한 테스크는 하나의 익스큐터(머신)에서 한 개의 데이터 파티션에서 실행됩니다.
- DAG: DAG는 Directed Acyclic Graph의 약자이며, 현재 맥락에서는 연산자의 DAG를 의미합니다.
- 익스큐터(Executor): 테스크 실행을 담당하는 프로세스입니다.
- 마스터(Master): 드라이버 프로그램이 실행되는 머신을 뜻합니다.
- 슬레이브(Slave): 실행 프로그램이 실행되는 머신을 뜻합니다.

4.2 스파크 구성요소



1. 스파크 드라이버 (Spark Driver)

- 사용자 애플리케이션 실행을 위한 별도의 프로세스를 의미합니다.
- 작업 실행을 스케줄링하기 위해 `SparkContext` 객체를 생성하고 클러스터 매니저는 스파크와 애플리케이션의 리소스를 효율적으로 분배할 수 있도록 조율합니다.

2. 익스큐터 (Executors)

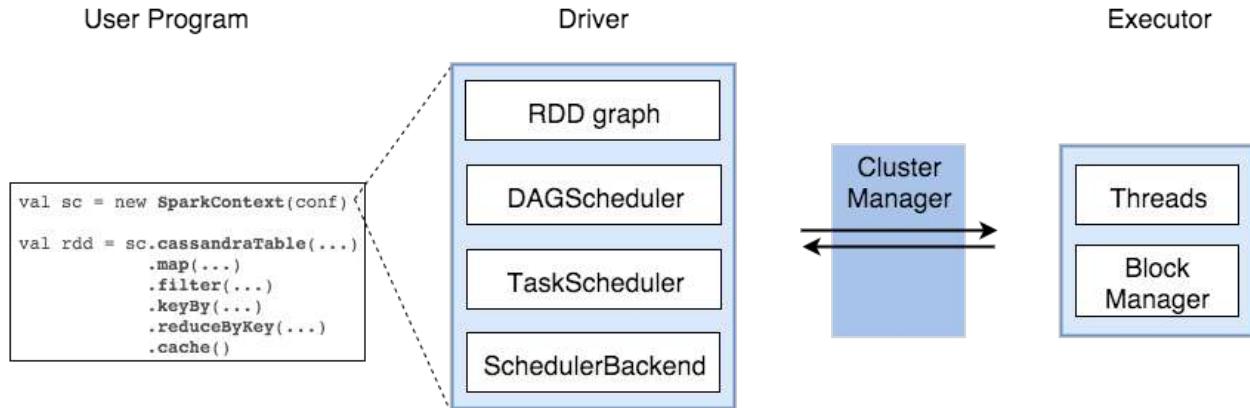
- 드라이버에서 예약된 태스크를 실행하는 역할을 합니다.
- 계산 결과를 메모리, 디스크 혹은 힙 메모리 밖에 (Off-heap) 저장합니다.
- 스토리지 시스템과 상호작용합니다.

3. 클러스터 매니저 (Cluster Manager)

- Mesos: 아파치 클러스터 매니저로 동적 리소스 공유 및 격리를 사용하여 여러 소스의 워크로드를 처리합니다. 마스터와 슬레이브 구조로 구성됩니다.

- YARN: 하둡 클러스터 매니저로 리소스와 노드 매니저로 구성됩니다.
- Spark Standalone: 스파크에서 자체적으로 제공하는 클러스터 매니저로 각 노드에 하나의 익스큐터만 실행 가능합니다.

스파크 드라이버는 사용자 코드를 클러스터에서 실제 실행되는 작업으로 변환하기 위해 더 많은 구성 요소를 포함하고 있습니다:



- 스파크 컨텍스트(SparkContext)
 - 스파크 클러스터에 연결해 해당 클러스터에 RDD, 누산기(accumulator) 및 브로드캐스트 변수를 만드는 데 사용할 수 있습니다.
- DAG스케줄러 (DAGScheduler)
 - 각 작업 스테이지의 DAG를 계산하여 테스크들의 선호되는 위치를 결정하고 (캐시 상태나 파일 위치 섞기를 기반으로) 작업을 실행하기 위한 최소스케줄을 찾는 테스크 스케줄러에게 이를 제출합니다.
- Task스케줄러 (TaskScheduler)
 - 클러스터에 작업을 보내고, 실행하고, 장애가 발생한 경우 재시도하고, 작업의 낙오를 완화하는 역할을 담당합니다.
- 스케줄러 백엔드 (SchedulerBackend)
 - 다양한 클러스터 관리 소프트웨어(Mesos, YARN, Standalone, 로컬)에 연결하여 기능을 확장하여 사용할 수 있는 스케줄링 시스템용 백엔드 인터페이스입니다.
- 블록매니저 (BlockManager)
 - 블록을 로컬 및 원격으로 다양한 저장소(메모리, 디스크 및 오프-힙)에 넣고 검색할 수 있는 인터페이스를 제공합니다.

4.3 아키텍처

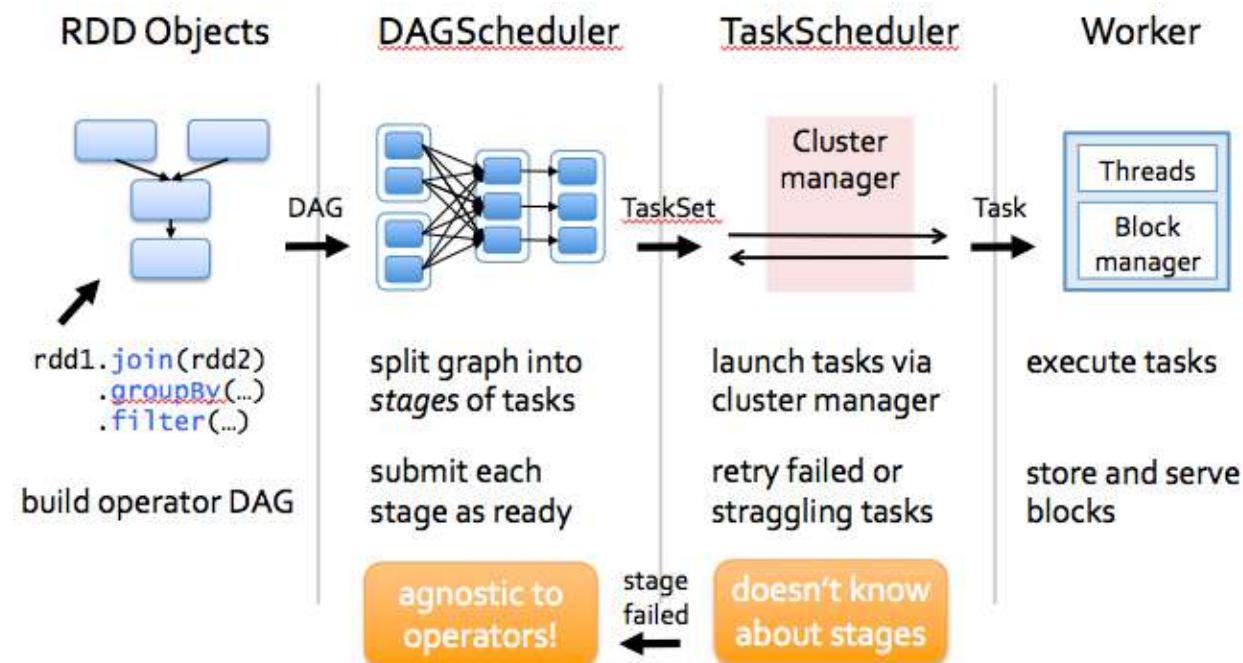
4.4 스파크는 어떻게 작동될까?

스파크는 최소한의 코드를 가지고 활용할 수 있으며 시스템이 여러 계층(layer)으로 나눠져 있습니다. 각 계층은 역할이 있습니다. 각 계층은 서로 독립적입니다.

첫 번째 계층은 인터프리터이고, 스파크는 스칼라 인터프리터를 일부 수정해 사용합니다. 스파크 콘솔에 코드를 입력하면(RDD 생성 및 연산자 적용), 스파크가 연산자 그래프를 생성합니다. 사용자가 수집과 같은 작업을 실행하면 그래프가 DAG 스케줄러에 제출됩니다. DAG 스케줄러는 연산자 그래프를 맵과 리듀스 단계로 나눕니다. 각 스테이지(Stage)는 입력 데이터의 파티션에 기반한 작업들로 구성됩니다.

DAG 스케줄러는 그래프를 최적화하기 위해 오퍼레이터(operators)들을 파이프라인으로 연결합니다. 예를 들어, 다수의 맵 오퍼레이터들은 단일 스테이지에 스케줄링 될 수 있습니다. 이 최적화는 Sparks 성능의 핵심입니다. DAG 스케줄러의 최종 결과는 스테이지들의 집합입니다.

스테이지들은 작업 스케줄러로 전달됩니다. 테스크 스케줄러는 클러스터 매니저(Spark Standalone/Yarn/Mesos) 통해 테스크를 시작합니다. 작업 스케줄러는 스테이지 간 종속성에 대해서는 알지 못합니다.



RDD 프로그래밍

중국 속담

상대가 아닌 자기 자신만 알면 이길 수도, 질 수도 있다. 만약 당신이 당신 자신이나 당신의 적을 알지 못한다면, 당신은 항상 위험에 처하게 될 것이다 – 손자병법

RDD는 복원 분산 데이터 집합(**Resilient Distributed Dataset**)을 의미합니다. 스파크의 RDD는 불변 객체 집합의 분산 컬렉션(collection)입니다. 각 RDD는 여러 개의 파티션(더 작은 집합과 비슷한 패턴)으로 나뉘며 클러스터의 다른 노드에서 계산될 수 있습니다.

5.1 RDD 생성

대개, RDD를 생성하는 방법은 외부 데이터 셋을 로드하거나 객체 모음의 한 집합을 배포하는 두 가지 방법이 있습니다. 다음은 프로그램의 기존에 존재하는 컬렉션을 가져와 스파크 컨텍스트에 이를 전달하는 `parallelize()` 함수를 사용해서 RDD를 생성하는 간단한 예시입니다.

1. `parallelize()` 함수를 사용하여

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark create RDD example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

df = spark.sparkContext.parallelize([(1, 2, 3, 'a b c'),
                                      (4, 5, 6, 'd e f'),
                                      (7, 8, 9, 'g h i')]).toDF(['col1', 'col2', 'col3', 'col4'])
```

생성된 RDD 데이터는 다음과 같습니다:

```
df.show()
+---+---+---+---+
| 1 | 2 | 3 | a b c |
| 4 | 5 | 6 | d e f |
| 7 | 8 | 9 | g h i |
```

(다음 페이지에 계속)

(o] 전 페이지에서 계속)

col1	col2	col3	col4
1	2	3	a b c
4	5	6	d e f
7	8	9	g h i

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark create RDD example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

myData = spark.sparkContext.parallelize([(1, 2), (3, 4), (5, 6), (7, 8), (9, 10)])
```

생성된 RDD 데이터는 다음과 같습니다:

```
myData.collect()
[(1, 2), (3, 4), (5, 6), (7, 8), (9, 10)]
```

2. `createDataFrame()` 함수를 사용하여

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark create RDD example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

Employee = spark.createDataFrame([
    ('1', 'Joe', '70000', '1'),
    ('2', 'Henry', '80000', '2'),
    ('3', 'Sam', '60000', '2'),
    ('4', 'Max', '90000', '1')],
    ['Id', 'Name', 'Salary', 'DepartmentId'])
```

생성된 RDD 데이터는 다음과 같습니다:

Id	Name	Salary	DepartmentId
1	Joe	70000	1
2	Henry	80000	2
3	Sam	60000	2
4	Max	90000	1

3. read 와 load 함수를 사용하여

a. .csv 파일로부터 데이터 셋을 읽는 경우

```
## set up SparkSession
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark create RDD example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

df = spark.read.format('com.databricks.spark.csv').\
    options(header='true', \
            inferschema='true') \
    .load("/home/feng/Spark/Code/data/Advertising.csv",
        header=True)

df.show(5)
df.printSchema()
```

생성된 RDD 데이터는 다음과 같습니다:

```
+---+---+---+---+
| _c0 | TV | Radio | Newspaper | Sales |
+---+---+---+---+
| 1 | 230.1 | 37.8 | 69.2 | 22.1 |
| 2 | 44.5 | 39.3 | 45.1 | 10.4 |
| 3 | 17.2 | 45.9 | 69.3 | 9.3 |
| 4 | 151.5 | 41.3 | 58.5 | 18.5 |
| 5 | 180.8 | 10.8 | 58.4 | 12.9 |
+---+---+---+---+
only showing top 5 rows

root
|-- _c0: integer (nullable = true)
|-- TV: double (nullable = true)
|-- Radio: double (nullable = true)
|-- Newspaper: double (nullable = true)
|-- Sales: double (nullable = true)
```

생성된 RDD는 변환과 액션 두 가지 유형의 작업을 제공합니다.

b. 데이터베이스로부터 데이터 셋을 읽는 경우

```
## set up SparkSession
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark create RDD example") \
    .config("spark.some.config.option", "some-value") \
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
.getOrCreate()

## User information
user = 'your_username'
pw   = 'your_password'

## Database information
table_name = 'table_name'
url = 'jdbc:postgresql://###.###.###.##:5432/dataset?user=' + user + '&
      password=' + pw
properties = {'driver': 'org.postgresql.Driver', 'password': pw, 'user':
      ': user'}

df = spark.read.jdbc(url=url, table=table_name,_
                     properties=properties)

df.show(5)
df.printSchema()
```

생성된 RDD data는 다음과 같습니다:

```
+---+---+---+---+
| _c0 | TV | Radio | Newspaper | Sales |
+---+---+---+---+
| 1 | 230.1 | 37.8 | 69.2 | 22.1 |
| 2 | 44.5 | 39.3 | 45.1 | 10.4 |
| 3 | 17.2 | 45.9 | 69.3 | 9.3 |
| 4 | 151.5 | 41.3 | 58.5 | 18.5 |
| 5 | 180.8 | 10.8 | 58.4 | 12.9 |
+---+---+---+---+
only showing top 5 rows

root
| -- _c0: integer (nullable = true)
| -- TV: double (nullable = true)
| -- Radio: double (nullable = true)
| -- Newspaper: double (nullable = true)
| -- Sales: double (nullable = true)
```

노트: 데이터베이스에서 테이블을 읽으려면 해당 데이터베이스에 적합한 드라이브가 필요합니다. 예를 들어 위의 데모에는 org.postgresql.Driver이 필요합니다. 여러분이 공식 웹사이트에서 postgresql-42.1.1.jar를 다운로드하여 이를 스파크 설치 경로 내 jars 폴더에 넣어야 합니다.

C. HDFS로부터 데이터 셋을 읽는 경우

```
from pyspark.conf import SparkConf
from pyspark.context import SparkContext
from pyspark.sql import HiveContext
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

sc= SparkContext('local','example')
hc = HiveContext(sc)
tf1 = sc.textFile("hdfs://cdhstltest/user/data/demo.CSV")
print(tf1.first())

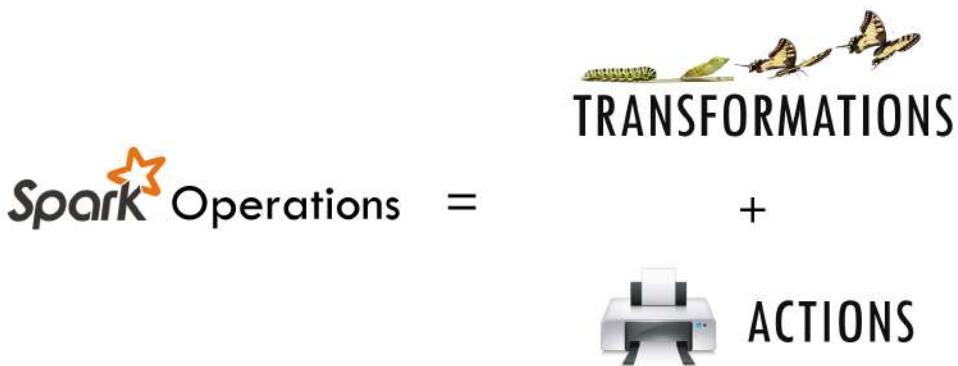
hc.sql("use intg_cme_w")
spf = hc.sql("SELECT * FROM spf LIMIT 100")
print(spf.show(5))

```

5.2 스파크 연산

경고: 아래 모든 그림은 Jeffrey Thompson로부터 인용하였습니다. 관심 있으신 독자분들은 [pyspark pictures](#)을 참고해 주세요.

스파크 연산에는 두 가지 주요 유형이 있습니다: 변형과 액션입니다[Karau2015].



노트: 어떤 사람들은 연산을 세가지 유형으로 정의하기도 합니다. 변형, 액션 그리고 셔플입니다.

5.2.1 스파크 변환

변환은 이전 RDD에서 새로운 RDD를 구성합니다. 예를 들어, 한 가지 일반적인 변환은 조건과 일치하는 데이터를 필터링하는 것입니다.



= easy = medium

Essential Core & Intermediate Spark Operations

General	Math / Statistical	Set Theory / Relational	Data Structure / I/O
<ul style="list-style-type: none"> map filter flatMap mapPartitions mapPartitionsWithIndex groupByKey sortBy 	<ul style="list-style-type: none"> sample randomSplit 	<ul style="list-style-type: none"> union intersection subtract distinct cartesian zip 	<ul style="list-style-type: none"> keyBy zipWithIndex zipWithUniqueId zipPartitions coalesce repartition repartitionAndSortWithinPartitions pipe



= easy = medium

Essential Core & Intermediate PairRDD Operations

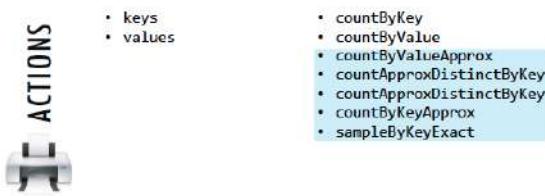
General	Math / Statistical	Set Theory / Relational	Data Structure
<ul style="list-style-type: none"> flatMapValues groupByKey reduceByKey reduceByKeyLocally foldByKey aggregateByKey sortByKey combineByKey 	<ul style="list-style-type: none"> sampleByKey 	<ul style="list-style-type: none"> cogroup (=groupWith) join subtractByKey fullOuterJoin leftOuterJoin rightOuterJoin 	<ul style="list-style-type: none"> partitionBy

5.2.2 스파크 액션

액션은 RDD를 기반으로 결과를 계산하고 드라이버 프로그램으로 반환하거나 외부 스토리지 시스템(예: HDFS)에 저장합니다.



<ul style="list-style-type: none"> reduce collect aggregate fold first take foreach top treeAggregate treeReduce foreachPartition collectAsMap 	<ul style="list-style-type: none"> count takeSample max min sum histogram mean variance stdev sampleVariance countApprox countApproxDistinct 	<ul style="list-style-type: none"> takeOrdered 	<ul style="list-style-type: none"> saveAsTextFile saveAsSequenceFile saveAsObjectFile saveAsHadoopDataset saveAsHadoopFile saveAsNewAPIHadoopDataset saveAsNewAPIHadoopFile
--	--	---	--



5.3 rdd.DataFrame vs pd.DataFrame

5.3.1 DataFrame 생성하기

1. 리스트(List)로 부터 생성하기

```
my_list = [['a', 1, 2], ['b', 2, 3], ['c', 3, 4]]
col_name = ['A', 'B', 'C']
```

:: 파이썬 코드:

```
# columns= 라고 작성해야 하니 주의하세요
pd.DataFrame(my_list, columns= col_name)
#
spark.createDataFrame(my_list, col_name).show()
```

:: 비교:

	A	B	C
0	a	1	2
1	b	2	3
2	c	3	4

주의: pd.DataFrame의 파라미터 columns=의 사용에 주의하세요. columns=(을)를 사용하지 않고 컬럼 명에 해당하는 리스트 값만 작성하면 이를 열이 아닌 행 이름으로 사용하기 때문입니다.

:: 파이썬 코드:

```
# # columns= 라고 작성해야 하니 주의하세요
pd.DataFrame(my_list, columns= col_name)
#
pd.DataFrame(my_list, col_name)
```

:: 비교:

A	B	C	0	1	2		
0	a	1	2	A	a	1	2
1	b	2	3	B	b	2	3
2	c	3	4	C	c	3	4

2. Dict로부터 생성하기

```
d = {'A': [0, 1, 0],
      'B': [1, 0, 1],
      'C': [1, 0, 0]}
```

:: 파이썬 코드:

```
pd.DataFrame(d)
# Tedious for PySpark
spark.createDataFrame(np.array(list(d.values()))).T.tolist(), list(d.keys()))
show()
```

:: 비교:

A B C			A B C		
			0	1	2
0	0	1	1	0	1
1	1	0	0	1	0
2	0	1	0	0	1

5.3.2 데이터 프레임 로드

1. 데이터 베이스로부터 로드하기

대부분 여러분은 동료들과 코드를 공유하거나 코드 리뷰 또는 품질 보증(QA)을 위해 여러분이 작성하신 코드를 공개해야 할 것입니다. 코드에 사용자 정보(User Information)를 넣는 것을 원치 않을 경우 login.txt에 사용자 정보를 따로 저장할 수 있습니다:

```
runawayhorse001
PythonTips
```

다음 코드를 사용하여 사용자 정보를 불러옵니다:

```
# 사용자 정보
try:
    login = pd.read_csv(r'login.txt', header=None)
    user = login[0][0]
    pw = login[0][1]
    print('사용자 정보가 준비되었습니다!')
except:
    print('로그인 정보가 없습니다!!!!')
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
# 데이터베이스 정보
host = '##.###.###.##'
db_name = 'db_name'
table_name = 'table_name'
```

:: 비교:

```
conn = psycopg2.connect(host=host, database=db_name, user=user, password=pw)
cur = conn.cursor()

sql = """
    select *
    from {table_name}
""".format(table_name=table_name)
dp = pd.read_sql(sql, conn)
```

```
# 데이터베이스 연결
url = 'jdbc:postgresql://'+host+':5432/' + db_name + '?user=' + user + '&password=' + pw
properties = {'driver': 'org.postgresql.Driver', 'password': pw, 'user': user}
ds = spark.read.jdbc(url=url, table=table_name, properties=properties)
```

주의: Pyspark에서 데이터베이스 상에 테이블을 읽기 위해선 데이터베이스와 대응되는 드라이버가 필요합니다. 예를 들어, 위의 데모에서는 org.postgresql.Driver가 필요해 여러분은 이를 spark가 설치된 경로에 있는 jars 폴더 안에 다운로드해야 합니다. 저는 공식 웹사이트로부터 postgresql-42.1.1.jar를 jar 폴더 안에 다운로드했습니다.

2. .csv로부터 로드하기

:: 비교:

```
# pd.DataFrame dp: DataFrame pandas
dp = pd.read_csv('Advertising.csv')
#rdd.DataFrame. dp: DataFrame spark
ds = spark.read.csv(path='Advertising.csv',
#                     sep=',',
#                     encoding='UTF-8',
#                     comment=None,
                     header=True,
                     inferSchema=True)
```

3. .json으로 부터 로드하기

Data는 <http://api.luftdaten.info/static/v1/data.json>에서 다운 받습니다

```
dp = pd.read_json("data/data.json")
ds = spark.read.json('data/data.json')
```

:: 파일 코드:

```
dp[['id','timestamp']].head(4)
#
ds[['id','timestamp']].show(4)
```

:: 비교:

```
+-----+-----+
|      id | 
+-----+-----+
| 2994551481 | 2019-02-28 |
| 2994551482 | 2019-02-28 |
| 2994551483 | 2019-02-28 |
| 2994551484 | 2019-02-28 |
+-----+
only showing top 4 rows
```

5.3.3 첫 n번째까지의 행

:: 파이썬 코드:

```
dp.head(4)
#
ds.show(4)
```

:: 비교:

```
+-----+-----+-----+-----+
|   TV | Radio | Newspaper | Sales |
+-----+-----+-----+-----+
| 230.1 | 37.8 | 69.2 | 22.1 |
| 44.5 | 39.3 | 45.1 | 10.4 |
| 17.2 | 45.9 | 69.3 | 9.3 |
| 151.5 | 41.3 | 58.5 | 18.5 |
+-----+-----+-----+-----+
only showing top 4 rows
```

5.3.4 컬럼 명

:: 파이썬 코드:

```
dp.columns
#
ds.columns
```

:: 비교:

```
Index(['TV', 'Radio', 'Newspaper', 'Sales'], dtype='object')
['TV', 'Radio', 'Newspaper', 'Sales']
```

5.3.5 데이터 타입

:: 파이썬 코드:

```
dp.dtypes
#
ds.dtypes
```

:: 비교:

```
TV          float64      [('TV', 'double'),
Radio        float64      ('Radio', 'double'),
Newspaper    float64      ('Newspaper', 'double'),
Sales         float64      ('Sales', 'double')]
dtype: object
```

5.3.6 널(NULL) 값 채우기

```
my_list = [['male', 1, None], ['female', 2, 3], ['male', 3, 4]]
dp = pd.DataFrame(my_list, columns=['A', 'B', 'C'])
ds = spark.createDataFrame(my_list, ['A', 'B', 'C'])
#
dp.head()
ds.show()
```

:: 비교:

	A	B	C		A	B	C
0	male	1	NaN	male	1	null	
1	female	2	3.0	female	2	3	
2	male	3	4.0	male	3	4	

:: 파이썬 코드:

```
dp.fillna(-99)
#
ds.fillna(-99).show()
```

:: 비교:

	A	B	C		A	B	C
0	male	1	-99		male	1	-99
1	female	2	3.0		female	2	3
2	male	3	4.0		male	3	4

5.3.7 값 대체하기(Replace)

:: 파이썬 코드:

```
# 주의: 특정 컬럼을 선택해야 합니다
dp.A.replace(['male', 'female'], [1, 0], inplace=True) dp
# 주의: na를 대체하면서 동시에 특정 열의 값을 대체하지 못합니다. 이 경우 특정 열의 대체 값만 진행됩니다.
ds.na.replace(['male', 'female'], ['1', '0']).show()
```

:: 비교:

	A	B	C		A	B	C
0	1	1	NaN		1	1	null
1	0	2	3.0		0	2	3
2	1	3	4.0		1	3	4

5.3.8 컬럼명 바꾸기

- 모든 열의 이름 바꾸기

:: 파이썬 코드:

```
dp.columns = ['a', 'b', 'c', 'd']
dp.head(4)
#
ds.toDF('a', 'b', 'c', 'd').show(4)
```

:: 비교:

	a	b	c	d
(다음 페이지에 계속)				

(이전 페이지에서 계속)

	a	b	c	d	
0	230.1	37.8	69.2	22.1	230.1 37.8 69.2 22.1
1	44.5	39.3	45.1	10.4	44.5 39.3 45.1 10.4
2	17.2	45.9	69.3	9.3	17.2 45.9 69.3 9.3
3	151.5	41.3	58.5	18.5	151.5 41.3 58.5 18.5
					+-----+-----+-----+-----+
					only showing top 4 rows

2. 하나 혹은 그 이상의 열 이름 바꾸기

```
mapping = {'Newspaper': 'C', 'Sales': 'D'}
```

:: 파이썬 코드:

```
dp.rename(columns=mapping).head(4)
#
new_names = [mapping.get(col, col) for col in ds.columns]
ds.toDF(*new_names).show(4)
```

:: 비교:

	TV	Radio	C	D	
0	230.1	37.8	69.2	22.1	230.1 37.8 69.2 22.1
1	44.5	39.3	45.1	10.4	44.5 39.3 45.1 10.4
2	17.2	45.9	69.3	9.3	17.2 45.9 69.3 9.3
3	151.5	41.3	58.5	18.5	151.5 41.3 58.5 18.5
					+-----+-----+-----+-----+
					only showing top 4 rows

노트: PySpark에서 한 칼럼의 이름을 바꾸기 위해 withColumnRenamed를 사용할 수도 있습니다.

:: 파이썬 코드:

```
ds.withColumnRenamed('Newspaper', 'Paper').show(4)
```

:: 비교:

	TV	Radio	Paper	Sales	
0	230.1	37.8	69.2	22.1	230.1 37.8 69.2 22.1
1	44.5	39.3	45.1	10.4	44.5 39.3 45.1 10.4
2	17.2	45.9	69.3	9.3	17.2 45.9 69.3 9.3
3	151.5	41.3	58.5	18.5	151.5 41.3 58.5 18.5
					+-----+-----+-----+-----+
					only showing top 4 rows

5.3.9 컬럼 삭제하기(Drop)

```
drop_name = ['Newspaper', 'Sales']
```

:: 파이썬 코드:

```
dp.drop(drop_name, axis=1).head(4)  
#  
ds.drop(*drop_name).show(4)
```

:: 표:

	TV	Radio		TV Radio	
0	230.1	37.8		230.1 37.8	
1	44.5	39.3		44.5 39.3	
2	17.2	45.9		17.2 45.9	
3	151.5	41.3		151.5 41.3	

only showing top 4 rows

5.3.10 추출(Filter)

```
dp = pd.read_csv('Advertising.csv')  
#  
ds = spark.read.csv(path='Advertising.csv',  
                     header=True,  
                     inferSchema=True)
```

:: 파이썬 코드:

```
dp[dp.Newspaper<20].head(4)  
#  
ds[ds.Newspaper<20].show(4)
```

:: 표:

	TV	Radio	Newspaper	Sales	TV Radio Newspaper Sales	
7	120.2	19.6	11.6	13.2	120.2 19.6 11.6 13.2	
8	8.6	2.1	1.0	4.8	8.6 2.1 1.0 4.8	
11	214.7	24.0	4.0	17.4	214.7 24.0 4.0 17.4	
13	97.5	7.6	7.2	9.7	97.5 7.6 7.2 9.7	

only showing top 4 rows

:: 파이썬 코드:

```
dp [ (dp.Newspaper<20) & (dp.TV>100) ].head(4)
#
ds [ (ds.Newspaper<20) & (ds.TV>100) ].show(4)
```

:: 비고:

	TV	Radio	Newspaper	Sales		TV	Radio	Newspaper	Sales
7	120.2	19.6	11.6	13.2		120.2	19.6	11.6	13.2
11	214.7	24.0	4.0	17.4		214.7	24.0	4.0	17.4
19	147.3	23.9	19.1	14.6		147.3	23.9	19.1	14.6
25	262.9	3.5	19.5	12.0		262.9	3.5	19.5	12.0

only showing top 4 rows

5.3.11 새 컬럼 추가하기

:: 파이썬 코드:

```
dp['tv_norm'] = dp.TV/sum(dp.TV)
dp.head(4)
import pyspark.sql.functions as F
ds.withColumn('tv_norm', ds.TV/ds.groupBy().agg(F.sum("TV")).collect()[0][0]).
    show(4)
```

:: 비고:

	TV	Radio	Newspaper	Sales	tv_norm		TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1	0.007824		230.1	37.8	69.2	22.
1	44.5	39.3	45.1	10.4	0.001513		44.5	39.3	45.1	10.
2	17.2	45.9	69.3	9.3	0.000585		17.2	45.9	69.3	9.
3	151.5	41.3	58.5	18.5	0.005152		151.5	41.3	58.5	18.

only showing top 4 rows

:: 파이썬 코드:

```
dp['cond'] = Ap.apply(lambda A: 1 if ((c.TV>100) & (c.Radio<40)) else 2 if A.
→Sales>10 else 3, axis=1)
#
ds.withColumn('cond', F.when((ds.TV>100) & (ds.Radio<40), 1) \
    .when(ds.Sales>10, 2) \
    .otherwise(3)).show(4)
```

:: 비고:

	TV	Radio	Newspaper	Sales	cond			
0	230.1	37.8	69.2	22.1	1	230.1 37.8	69.2 22.1	↳
1	44.5	39.3	45.1	10.4	2	44.5 39.3	45.1 10.4	↳
2	17.2	45.9	69.3	9.3	3	17.2 45.9	69.3 9.3	↳
3	151.5	41.3	58.5	18.5	2	151.5 41.3	58.5 18.5	↳
						+-----+-----+-----+-----+	+-----+-----+-----+-----+	
								only showing top 4 rows

:: 파이썬 코드:

```
dp['log_tv'] = np.log(dp.TV)
dp.head(4)
#
import pyspark.sql.functions as F
ds.withColumn('log_tv', F.log(ds.TV)).show(4)
```

:: 비고:

	TV	Radio	Newspaper	Sales	log_tv			
0	230.1	37.8	69.2	22.1	5.438514	230.1 37.8	69.2 22.1	↳
1	44.5	39.3	45.1	10.4	3.795489	44.5 39.3	45.1 10.	↳
2	17.2	45.9	69.3	9.3	2.844909	17.2 45.9	69.3 9.	↳
3	151.5	41.3	58.5	18.5	5.020586	151.5 41.3	58.5 18.5	↳
						+-----+-----+-----+-----+	+-----+-----+-----+-----+	
								(다음 페이지에 계속)

(이전 페이지에서 계속)

only showing top 4 rows

:: 파일 코드:

```
dp['tv+10'] = dp.TV.apply(lambda x: x+10)
dp.head(4)
#
ds.withColumn('tv+10', ds.TV+10).show(4)
```

:: 비교:

	TV	Radio	Newspaper	Sales	tv+10				
0	230.1	37.8	69.2	22.1	240.1	230.1	37.8	69.2	22.
1	44.5	39.3	45.1	10.4	54.5	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3	27.2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5	161.5	151.5	41.3	58.5	18.

only showing top 4 rows

5.3.12 조인(Join)

```
leftp = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                      'B': ['B0', 'B1', 'B2', 'B3'],
                      'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']},
                      index=[0, 1, 2, 3])

rightp = pd.DataFrame({'A': ['A0', 'A1', 'A6', 'A7'],
                       'F': ['B4', 'B5', 'B6', 'B7'],
                       'G': ['C4', 'C5', 'C6', 'C7'],
                       'H': ['D4', 'D5', 'D6', 'D7']},
                       index=[4, 5, 6, 7])

lefts = spark.createDataFrame(leftp)
rights = spark.createDataFrame(rightp)
```

	A	B	C	D		A	F	G	H	
0	A0	B0	C0	D0		4	A0	B4	C4	D4
1	A1	B1	C1	D1		5	A1	B5	C5	D5

(다음 페이지에 계속)

(이전 페이지에서 계속)

2 A2 B2 C2 D2	6 A6 B6 C6 D6
3 A3 B3 C3 D3	7 A7 B7 C7 D7

1. Left Join

:: 파이썬 코드:

```
leftp.merge(rightp, on='A', how='left')
#
lefts.join(rights, on='A', how='left')
    .orderBy('A', ascending=True).show()
```

:: 비교:

	A	B	C	D	F	G	H	A	B	C	D	F	
0	A0	B0	C0	D0	B4	C4	D4	A0	B0	C0	D0	B4	
1	A1	B1	C1	D1	B5	C5	D5	A1	B1	C1	D1	B5	
2	A2	B2	C2	D2	Nan	Nan	Nan	A2	B2	C2			
3	A3	B3	C3	D3	Nan	Nan	Nan	A3	B3	C3			
					D2 null null null								
						D3 null null null							

2. Right Join

:: 파이썬 코드:

```
leftp.merge(rightp, on='A', how='right')
#
lefts.join(rights, on='A', how='right')
    .orderBy('A', ascending=True).show()
```

:: 비교:

	A	B	C	D	F	G	H	A	B	C	D	F	
0	A0	B0	C0	D0	B4	C4	D4	A0	B0	C0	D0	B4	
1	A1	B1	C1	D1	B5	C5	D5	A1	B1	C1	D1	B5	
2	A2	B2	C2	D2	Nan	Nan	Nan	A2	B2	C2			
3	A3	B3	C3	D3	Nan	Nan	Nan	A3	B3	C3			
					D2 null null null								
						D3 null null null							

(다음 페이지에 계속)

(이전 페이지에서 계속)

2 A6 NaN NaN NaN B6 C6 D6 →C6 D6	A6 null null null B6 ↴
3 A7 NaN NaN NaN B7 C7 D7 →C7 D7	A7 null null null B7 ↴
+---+---+---+---+---+---+	
↳---+---+	

3. Inner Join

:: 파이썬 코드:

```
leftp.merge(rightp, on='A', how='inner')
#
lefts.join(rights, on='A', how='inner')
    .orderBy('A', ascending=True).show()
```

:: 비교:

	A	B	C	D	F	G	H	A	B	C	D	F	G	H
0	A0	B0	C0	D0	B4	C4	D4	A0	B0	C0	D0	B4	C4	D4
1	A1	B1	C1	D1	B5	C5	D5	A1	B1	C1	D1	B5	C5	D5

4. Full Join

:: 파이썬 코드:

```
leftp.merge(rightp, on='A', how='outer')
#
lefts.join(rights, on='A', how='full')
    .orderBy('A', ascending=True).show()
```

:: 비교:

	A	B	C	D	F	G	H	A	B	C	D
0	A0	B0	C0	D0	B4	C4	D4	A0	B0	C0	D0
1	A1	B1	C1	D1	B5	C5	D5	A1	B1	C1	D1
2	A2	B2	C2	D2	Nan	Nan	Nan	A2	B2	C2	
3	A3	B3	C3	D3	Nan	Nan	Nan	A3	B3	C3	
4	A6	Nan	Nan	Nan	B6	C6	D6	A6	null	null	
	↳B6	C6	D6								

(다음 페이지에 계속)

(이전 페이지에서 계속)

5	A7	NaN	NaN	NaN	B7	C7	D7		A7 null null null ↴
↪	B7	C7	D7						+---+---+---+---+---+
↪	+	---+---+---+							

5.3.13 열 결합하기 (Concat)

```
my_list = [('a', 2, 3),
           ('b', 5, 6),
           ('c', 8, 9),
           ('a', 2, 3),
           ('b', 5, 6),
           ('c', 8, 9)]
col_name = ['col1', 'col2', 'col3']
#
dp = pd.DataFrame(my_list, columns=col_name)
ds = spark.createDataFrame(my_list, schema=col_name)
```

	col1	col2	col3
0	a	2	3
1	b	5	6
2	c	8	9
3	a	2	3
4	b	5	6
5	c	8	9

:: 파이썬 코드:

```
dp['concat'] = dp.apply(lambda x:'%s%s' %(x['col1'],x['col2']),axis=1)
dp
#
ds.withColumn('concat',F.concat('col1','col2')).show()
```

:: 비교:

	col1	col2	col3	concat		col1 col2 col3 concat
0	a	2	3	a2	a 2 3 a2	
1	b	5	6	b5	b 5 6 b5	
2	c	8	9	c8	c 8 9 c8	
3	a	2	3	a2	a 2 3 a2	
4	b	5	6	b5	b 5 6 b5	
5	c	8	9	c8	c 8 9 c8	

5.3.14 GroupBy

:: 파이썬 코드:

```
dp.groupby(['col1']).agg({'col2':'min','col3':'mean'})
#
ds.groupBy(['col1']).agg({'col2': 'min', 'col3': 'avg'}).show()
```

:: 비교:

	col2	col3	
col1			+-----+-----+-----+
a	2	3	col1 min(col2) avg(col3)
b	5	6	+-----+-----+-----+
c	8	9	c 8 9.0
			b 5 6.0
			a 2 3.0
			+-----+-----+-----+

5.3.15 피벗(Pivot)

:: 파이썬 코드:

```
import numpy as np
pd.pivot_table(dp, values='col3', index='col1', columns='col2', aggfunc=np.sum)
ds.groupBy(['col1']).pivot('col2').sum('col3').show()
```

:: 비교:

col2	2	5	8	
col1				+-----+-----+-----+
a	6.0	NaN	NaN	col1 2 5 8
b	NaN	12.0	NaN	+-----+-----+-----+
c	NaN	NaN	18.0	b null 12 null
				a 6 null null
				+-----+-----+-----+

5.3.16 Window

```
d = {'A':['a','b','c','d'], 'B':['m','m','n','n'], 'C':[1,2,3,6]}
dp = pd.DataFrame(d)
ds = spark.createDataFrame(dp)
```

:: 파이썬 코드:

```
dp['rank'] = dp.groupby('B')['C'].rank('dense', ascending=False)
#
from pyspark.sql.window import Window
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
w = Window.partitionBy('B').orderBy(ds.C.desc())
ds = ds.withColumn('rank', F.rank().over(w))
```

:: 비교:

	A	B	C	rank
0	a	m	1	2.0
1	b	m	2	1.0
2	c	n	3	2.0
3	d	n	6	1.0

	b	m	2	1
0	b	m	2	1
1	a	m	1	2
2	d	n	6	1
3	c	n	3	2

5.3.17 rank vs dense_rank

```
d = {'Id': [1, 2, 3, 4, 5, 6],
      'Score': [4.00, 4.00, 3.85, 3.65, 3.65, 3.50]}
#
data = pd.DataFrame(d)
dp = data.copy()
ds = spark.createDataFrame(data)
```

	Id	Score
0	1	4.00
1	2	4.00
2	3	3.85
3	4	3.65
4	5	3.65
5	6	3.50

:: 파이썬 코드:

```
dp['Rank_dense'] = dp['Score'].rank(method='dense', ascending=False)
dp['Rank'] = dp['Score'].rank(method='min', ascending=False)
dp
#
import pyspark.sql.functions as F
from pyspark.sql.window import Window
w = Window.orderBy(ds.Score.desc())
ds = ds.withColumn('Rank_spark_dense', F.dense_rank().over(w))
ds = ds.withColumn('Rank_spark', F.rank().over(w))
ds.show()
```

:: 비교:

	Id	Score	Rank_dense	Rank	Rank_spark_dense	Rank_spark
0	1	4.00	1	1	1	1

(다음 페이지에 계속)

(◎ 전 페이지에서 계속)

0	1	4.00	1.0	1.0		1	4.0	1	1
1	2	4.00	1.0	1.0		2	4.0	1	1
2	3	3.85	2.0	3.0		3	3.85	2	3
3	4	3.65	3.0	4.0		4	3.65	3	4
4	5	3.65	3.0	4.0		5	3.65	3	4
5	6	3.50	4.0	6.0		6	3.5	4	6

통계학 및 선형대수 예비

중국 속담

상대가 아닌 자기 자신만 알면 이길 수도, 질 수도 있다. 당신 자신이나 당신의 적 둘 다 모른다면, 당신 자신이 위험에 처하게 될 것이다 - 손자병법

6.1 표기

- m : 샘플 수
- n : 특징(features) 수
- y_i : i 번째 라벨
- \hat{y}_i : i 번째 예측 라벨
- $\bar{\mathbf{y}} = \frac{1}{m} \sum_{i=1}^m y_i$: \mathbf{y} 의 평균
- \mathbf{y} : 라벨(label) 벡터
- $\hat{\mathbf{y}}$: 예측된 라벨 벡터

6.2 선형대수 예비

수치 분석을 위한 기초 시험 노트에 선형 대수 사전 지식을 작성하였으니 자세한 내용은 [Feng2014]를 참조하시길 바랍니다(그림. [선형대수 예비](#)).

1 Preliminaries

1.1 Linear Algebra Preliminaries

1.1.1 Common Properties

Properties 1.1. (Structure of Matrices) Let $A = [A_{ij}]$ be a square or rectangular matrix, A is called

- **diagonal** : if $a_{ij} = 0, \forall i \neq j,$
- **upper triangular** : if $a_{ij} = 0, \forall i > j,$
- **upper Hessenberg** : if $a_{ij} = 0, \forall i > j + 1,$
- **block diagonal** : $A = \text{diag}(A_{11}, A_{22}, \dots, A_{nn}),$
- **tridiagonal** : if $a_{ij} = 0, \forall |i - j| > 1,$
- **lower triangular** : if $a_{ij} = 0, \forall i < j,$
- **lower Hessenberg** : if $a_{ij} = 0, \forall j > i + 1,$
- **block diagonal** : $A = \text{diag}(A_{i,i-1}, A_{ii}, \dots, A_{i,i+1}).$

Properties 1.2. (Type of Matrices) Let $A = [A_{ij}]$ be a square or rectangular matrix, A is called

- **Hermitian** : if $A^* = A,$
- **symmetric** : if $A^T = A,$
- **normal** : if $A^T A = AA^T, \text{when } A \in \mathbb{R}^{n \times n},$
if $A^* A = AA^*, \text{when } A \in \mathbb{C}^{n \times n},$
- **skew hermitian** : if $A^* = -A,$
- **skew symmetric** : if $A^T = -A,$
- **orthogonal** : if $A^T A = I, \text{when } A \in \mathbb{R}^{n \times n},$
unitary : if $A^* A = I, \text{when } A \in \mathbb{C}^{n \times n}.$

Properties 1.3. (Properties of invertible matrices) Let A be $n \times n$ square matrix. If A is **invertible**, then

- $\det(A) \neq 0,$
- $\text{rank}(A) = n,$
- $Ax = b$ has a unique solution for every $b \in \mathbb{R}^n$
- the row vectors are **linearly independent**,
- the row vectors of A form a basis for $\mathbb{R}^n.$
- $\text{nullity}(A) = 0,$
- $\lambda_i \neq 0, (\lambda_i \text{ eigenvalues}),$
- $Ax = 0$ has only trivial solution,
- the column vectors are **linearly independent**,
- the column vectors of A form a basis for $\mathbb{R}^n,$
- the column vectors of A span $\mathbb{R}^n.$

Properties 1.4. (Properties of conjugate transpose) Let A, B be $n \times n$ square matrix and γ be a complex constant, then

- $(A^*)^* = A,$
- $(AB)^* = B^* A^*,$
- $(A + B)^* = A^* + B^*,$
- $\det(A^*) = \det(A)$
- $\text{tr}(A^*) = \text{tr}(A)$
- $(\gamma A)^* = \gamma^* A^*.$

Properties 1.5. (Properties of similar matrices) If $A \sim B$, then

- $\det(A) = \det(B),$
- $\text{eig}(A) = \text{eig}(B),$
- $A \sim A,$
- $\text{rank}(A) = \text{rank}(B),$
- if $B \sim C$, then $A \sim C$
- $B \sim A$

6.3 측정 공식

6.3.1 평균 절대 오차(Mean absolute error)

통계학에서 **MAE**(평균 절대 오차)는 두 연속 변수 사이의 차이를 측정하는 것입니다. 평균 절대 오차는 다음과 같습니다.:

$$\text{MAE} = \frac{1}{m} \sum_{i=1}^m |\hat{y}_i - y_i|.$$

6.3.2 평균 제곱 오차(Mean squared error)

통계학에서 (관측되지 않은 수량을 추정하는 절차의) 추정량의 **평균 제곱 오차(MSE, Mean Squared Error)**는 오차 또는 편차의 평균을 의미합니다. 즉, 관측값과 예측값의 차이를 측정합니다.

$$\text{MSE} = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2$$

6.3.3 평균 제곱근 오차(Root Mean squared error)

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2}$$

6.3.4 총 제곱합(Total sum of squares)

통계 데이터 분석에서 **총 제곱합(TSS, Total Sum of Squares)**은 분석 결과를 나타내는 표준적인 방법 중 하나로 각 관측값과 전체 평균 간의 차이를 제곱하여 모두 합한 것입니다.

$$\text{TSS} = \sum_{i=1}^m (y_i - \bar{y})^2$$

6.3.5 설명된 제곱합(Explained Sum of Squares)

통계학에서 **설명된 제곱합(ESS, Explained sum of squares)**는 모형 제곱합 또는 회귀 제곱합으로 알려져 있습니다.

ESS는 예측값과 반응 변수의 평균값 차이의 제곱의 합으로 다음과 같습니다:

$$\text{ESS} = \sum_{i=1}^m (\hat{y}_i - \bar{y})^2$$

6.3.6 잔차 제곱합(Residual Sum of Squares)

통계학에서 잔차 제곱합(RSS, Residual sum of squares) 또는 예측오차의 제곱합(SSE)으로 알려져 있습니다. RSS(잔차 제곱합)은 잔차들의 제곱합을 의미합니다:

$$\text{RSS} = \sum_{i=1}^m (\hat{y}_i - y_i)^2$$

6.3.7 결정 계수 R^2

$$R^2 := \frac{\text{ESS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}}.$$

노트: 일반적으로 $(\mathbf{y}^T \bar{\mathbf{y}} = \hat{\mathbf{y}}^T \bar{\mathbf{y}})$, 총 제곱합 = 설명된 제곱합 + 잔차 제곱합입니다, 즉:

$$\text{TSS} = \text{ESS} + \text{RSS}, \quad \mathbf{y}^T \bar{\mathbf{y}} = \hat{\mathbf{y}}^T \bar{\mathbf{y}} \text{ 인 경우에만}$$

자세한 내용은 일반 최소 제곱 모형에서 분할을 참조하세요.

6.4 혼동 행렬(Confusion Matrix)

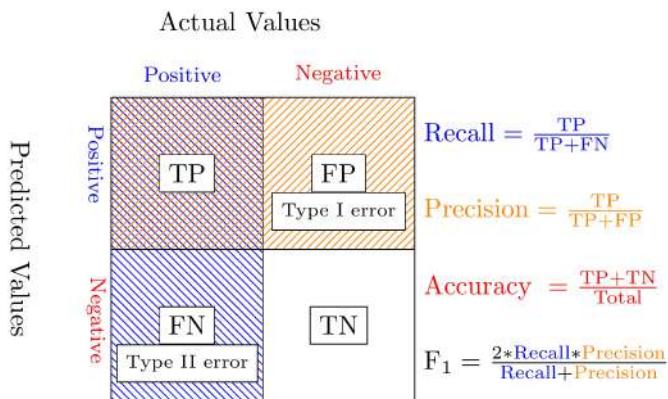


그림. 2: 혼동 행렬

6.4.1 재현율(Recall)

$$\text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}$$

6.4.2 정밀도(Precision)

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

6.4.3 정확도(Accuracy)

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total}}$$

6.4.4 F_1 -스코어

$$F_1 = \frac{2 * \text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}$$

6.5 통계 검정

6.5.1 상관성 검정

- 피어슨 상관관계(Pearson correlation): 두 연속형 변수 사이의 연관성의 강도를 검정합니다.
- 스피어먼 상관관계(Spearman correlation): 두 순서형 변수 사이의 연관성의 강도를 검정합니다(정규 분포 데이터의 가정에 의존하지 않습니다).
- 카이-제곱 검정(Chi-square): 두 범주형 변수 사이의 연관성의 강도를 검정합니다.

6.5.2 평균 검정의 비교

- 대응 표본 T 검정(Paired T-test): 관련된 두 변수 간의 차이를 검정합니다.
- 독립 표본 T-검정(Independent T-test): 두 독립 변수 간의 차이를 검정합니다.
- 분산 분석(ANOVA): 두 개 이상의 집단들이 있을 때, 각 집단 내와 집단 간 변량의 분산을 계산한 후 집단들의 평균 간의 차이를 검정합니다.

6.5.3 비모수 검정

- Wilcoxon 순위 합 검정: 두 독립 변수 간의 차이 검정은 차이의 크기와 방향을 고려합니다.
- Wilcoxon 부호 순위 검정: 두 관련 변수 간의 차이 검정은 차이의 크기와 방향을 고려합니다.
- 부호 검정(Sign test): 두 관련 변수가 다른지 검정합니다 – 변화의 크기를 무시하고 방향만 고려합니다.

데이터 탐색

중국 속담

천 리의 여정은 한 걸음으로 시작된다 – 노자 명언

데이터 과학에서 데이터 셋을 이해하는 것이 가장 어려운 일이라고는 할 수 없지만, 매우 중요하고 시간이 많이 걸립니다. 데이터 탐색은 데이터 분석 절차 중 하나로 통계 및 시각화 기법을 통해 데이터를 설명하는 단계입니다. 특징(features)을 이해하고 모델에 중요한 특징(features)을 제공하기 위해 데이터를 탐색합니다.

7.1 단변량 분석

수학에서, 단변량은 한 변수의 식, 방정식, 함수 또는 다항식을 나타냅니다. "Uni"는 "하나의"를 의미하므로, 다시 말해 데이터는 하나의 변수만 가집니다. 따라서 원인이나 관계를 다룰 필요가 없습니다. 단변량 분석은 데이터를 가져와 변수(속성)를 하나씩 요약하고 데이터에서 패턴을 찾습니다.

단변량 데이터에서 발견되는 패턴을 설명할 수 있는 여러 가지 방법에는 중심 경향성(평균, 최빈값 및 중위수)과 산포도: 범위, 분산, 최대, 최소, 사분위수(사분위수 범위 포함), 변동 계수 및 표준 편차가 있습니다. 또한 단변량 데이터를 사용하여 데이터를 시각화하고 설명하는 몇 가지 방법이 있습니다. 예를 들어, 도수 분포표, 막대 차트, 히스토그램, 도수 다각형, 원형 차트 등이 있습니다.

변수는 범주형 또는 수치형일 수 있습니다. 서로 다른 통계 및 시각화 기법들을 활용해 각각의 변수 유형을 살펴보겠습니다.

- 주피터 노트북은 데이터 탐색에서 다운로드할 수 있습니다.
- 데이터는 독일의 신용 데이터에서 다운로드할 수 있습니다.

7.1.1 수치형 변수

Describe

pandas와 spark의 describe 함수는 최소값, 중위수, 최대값, 사분위수 및 표준 편차와 같은 대부분의 통계치를 제공합니다. 사용자 정의 함수로 더 많은 통계치를 얻을 수 있습니다.

```
# 설명을 위해 선택된 변수들
num_cols = ['Account Balance', 'No of dependents']
df.select(num_cols).describe().show()
```

summary	Account Balance	No of dependents
count	1000	1000
mean	2.577	1.155
stddev	1.2576377271108936	0.36208577175319395
min	1	1
max	4	2

PySpark의 기본 함수에 사분위수가 포함되어 있지 않을 수 있습니다. 다음 함수를 통해 Pandas에서 describe 함수를 실행한 것과 동일한 결과를 얻을 수 있겠습니다.

```
def describe_pd(df_in, columns, deciles=False):
    """
    기본 통계 결과와 사분위수를 합치는 함수
    :param df_in: 입력 데이터 프레임
    :param columns: 수치형 변수의 열 이름 목록
    :param deciles: 사분위수 출력

    :반환 : 입력 데이터 프레임의 수치형 설명 정보

    :저자: Ming Chen and Wenqiang Feng
    :이메일: von198@gmail.com
    """

    if deciles:
        percentiles = np.array(range(0, 110, 10))
    else:
        percentiles = [25, 50, 75]

    percs = np.transpose([np.percentile(df_in.select(x).collect(), ↪
    percentiles) for x in columns])
    percs = pd.DataFrame(percs, columns=columns)
    percs['summary'] = [str(p) + '%' for p in percentiles]

    spark_describe = df_in.describe().toPandas()
    new_df = pd.concat([spark_describe, percs], ignore_index=True)
    new_df = new_df.round(2)
    return new_df[['summary']] + columns
```

```
describe_pd(df, num_cols)
```

summary	Account Balance	No of dependents
count	1000.0	1000.0
mean	2.577	1.155
stddev	1.2576377271108936	0.362085771753194
min	1.0	1.0
max	4.0	2.0
25%	1.0	1.0
50%	2.0	1.0
75%	4.0	1.0

때로는 데이터 보안 문제로 실제 데이터를 고객사에 전송할 수 없어 고객이 여러분에게 십분위수(deciles)와 같은 더 많은 통계 결과를 요청할 수 있습니다. 여러분이 이러한 요청에 대응하기 위해 다음 함수를 적용할 수 있습니다.

```
describe_pd(df, num_cols, deciles=True)
```

summary	Account Balance	No of dependents
count	1000.0	1000.0
mean	2.577	1.155
stddev	1.2576377271108936	0.362085771753194
min	1.0	1.0
max	4.0	2.0
0%	1.0	1.0
10%	1.0	1.0
20%	1.0	1.0
30%	2.0	1.0
40%	2.0	1.0
50%	2.0	1.0
60%	3.0	1.0
70%	4.0	1.0
80%	4.0	1.0
90%	4.0	2.0
100%	4.0	2.0

왜도 및 첨도

이 섹션은 위키피디아의 왜도(Skewness)에서 비롯되었습니다.

학률 이론과 통계학에서 왜도는 실수를 치역으로 가지는 랜덤 변수의 평균에 대한 학률 분포의 비대칭을 측정하는 것이다. 왜도의 값은 양수 또는 음수이거나 정의되지 않은 값일 수 있습니다. 단봉 분포(unimodal distribution)의 경우 일반적으로 음의 치우침은 꼬리가 분포의 왼쪽에 있음을 나타내고 양의 치우침은 꼬리가 오른쪽에 있음을 나타냅니다.

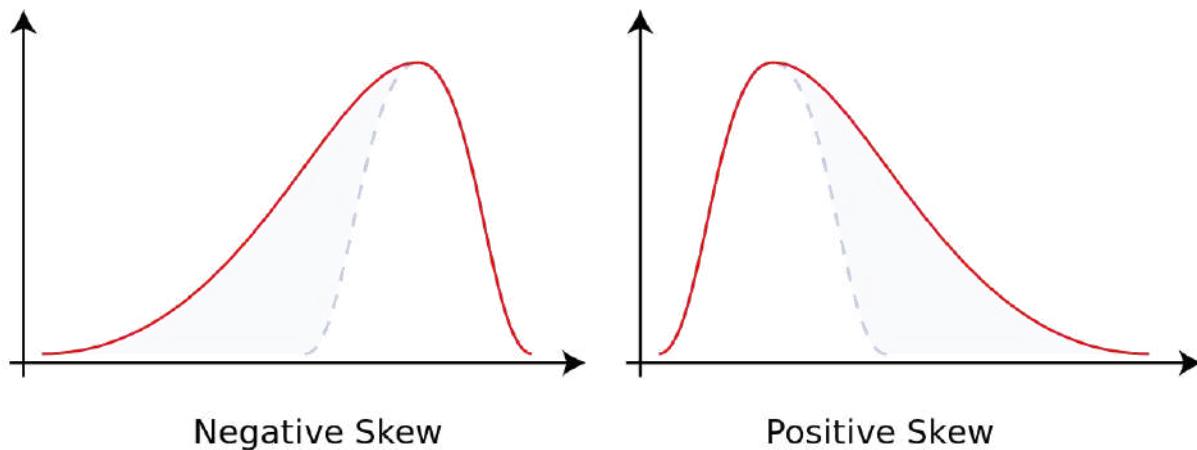
아래 그림의 두 분포를 고려해봅시다. 각 그래프 내에서 분포의 오른쪽에 있는 값은 왼쪽에 있는

값과 다르게 가늘어집니다. 이러한 가늘어지는 면을 꼬리라고 하며, 분포가 다음과 같은 두 가지 종류의 왜도를 갖는지 시각적으로 확인할 수 있습니다.

1. 음의 값을 갖는 왜도: 왼쪽 꼬리는 더 길고, 분포가 그림의 오른쪽에 치우쳐 있습니다. 분포는 곡선 자체가 오른쪽으로 치우쳐 있거나 기울어져 있음에도 불구하고 왼쪽으로 치우쳐 있다고 합니다. 왼쪽 대신에 왼쪽 꼬리가 길게 늘여져 있고 종종 평균이 데이터의 일반적인 중심에서 왼쪽으로 치우쳐 있는 것을 의미합니다. 좌편향 분포는 보통 우편향 곡선으로 나타납니다.
2. 양의 값을 갖는 왜도: 오른쪽 꼬리는 더 길고, 분포가 그림의 왼쪽에 집중되어 있습니다. 분포는 곡선 자체가 왼쪽으로 치우쳐 있거나 기울어져 있음에도 불구하고 오른쪽으로 치우쳐 있다고 합니다. 오른쪽 대신에 오른쪽 꼬리가 길게 늘어져 있고 종종 평균이 데이터의 일반적인 중심에서 오른쪽으로 치우쳐 있는 것을 의미합니다. 우편향 분포는 보통 좌편향 곡선으로 나타냅니다.

이 부분은 위키피디아 첨도(Kurtosis)로부터 작성되었습니다.

확률론과 통계학에서 첨도("약간 굽은, 아치"라는 뜻의 kyrtos 혹은 kurtos)는 실수를 치역으로 가지는 랜덤 변수의 확률분포의 꼬리가 두꺼운 정도를 나타내는 척도입니다. 왜도 개념과 유사한 방식으로 첨도는 확률 분포의 모양을 설명하는 것이고 왜도와 같이 이론적 분포에 대해 첨도를 정량화하는 여러 방법과 모집단의 표본으로부터 추정하는 방법이 있습니다.



```
from pyspark.sql.functions import col, skewness, kurtosis
df.select(skewness(df['Age (years)']), kurtosis(df['Age (years)'])).show()
```

skewness(Age (years))	kurtosis(Age (years))
1.0231743160548064	0.6114371688367672

경고: 때로는 통계가 오해를 불러일으킬 수 있습니다!

F. J. Anscombe는 통계 계산과 그래프를 모두 만들라고 말한 적이 있습니다. 두 종류의 결과물을 모두 함께 고려해야 한다는 뜻이기도 합니다. 데이터를 이해하는 데 있어 각각이 기여하는 부분이 있습니다. 그럼 [동일한 통계량, 다른 그래프](#)(Datasaurus 및 기타 12개)의 13개 데이터 셋들은 각각 소수점 두 자리까지 동일한 요약 통계량을(x/y 평균, x/y 표준 편차, 및 피어슨 상관관계) 가지면서도 데이터의 겉모습은 크게 다릅니다. 자세한 내용과 흥미로운 결과는 [동일한 통계량, 다른 그래프](#)에서 확인할 수 있습니다. 서로 다른 겉모양을 갖는 13개 데이터 집합을 만들기 위해 개발한 기술 등을 설명합니다.

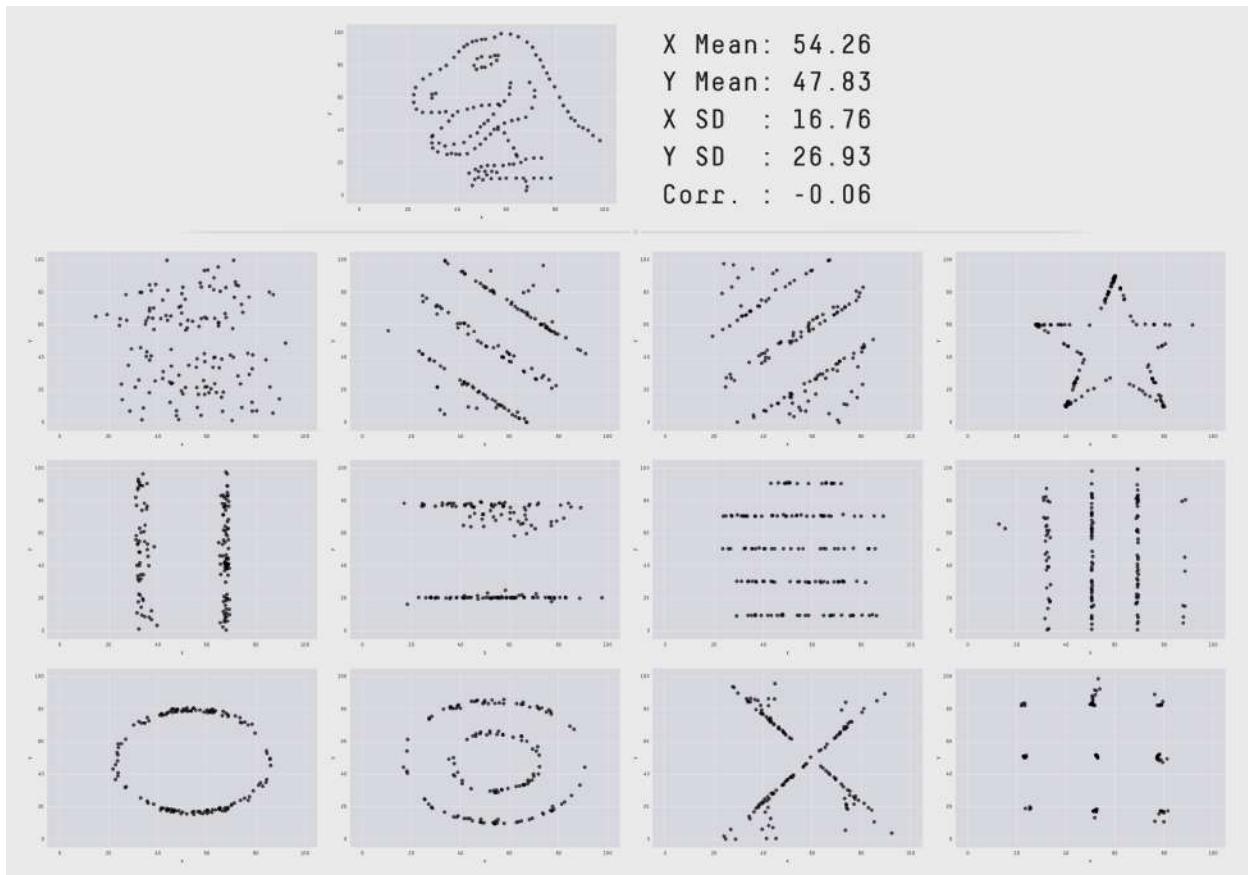


그림. 1: 같은 통계량, 다른 그래프

히스토그램

경고: 히스토그램은 종종 막대 그래프와 혼동되곤 합니다!

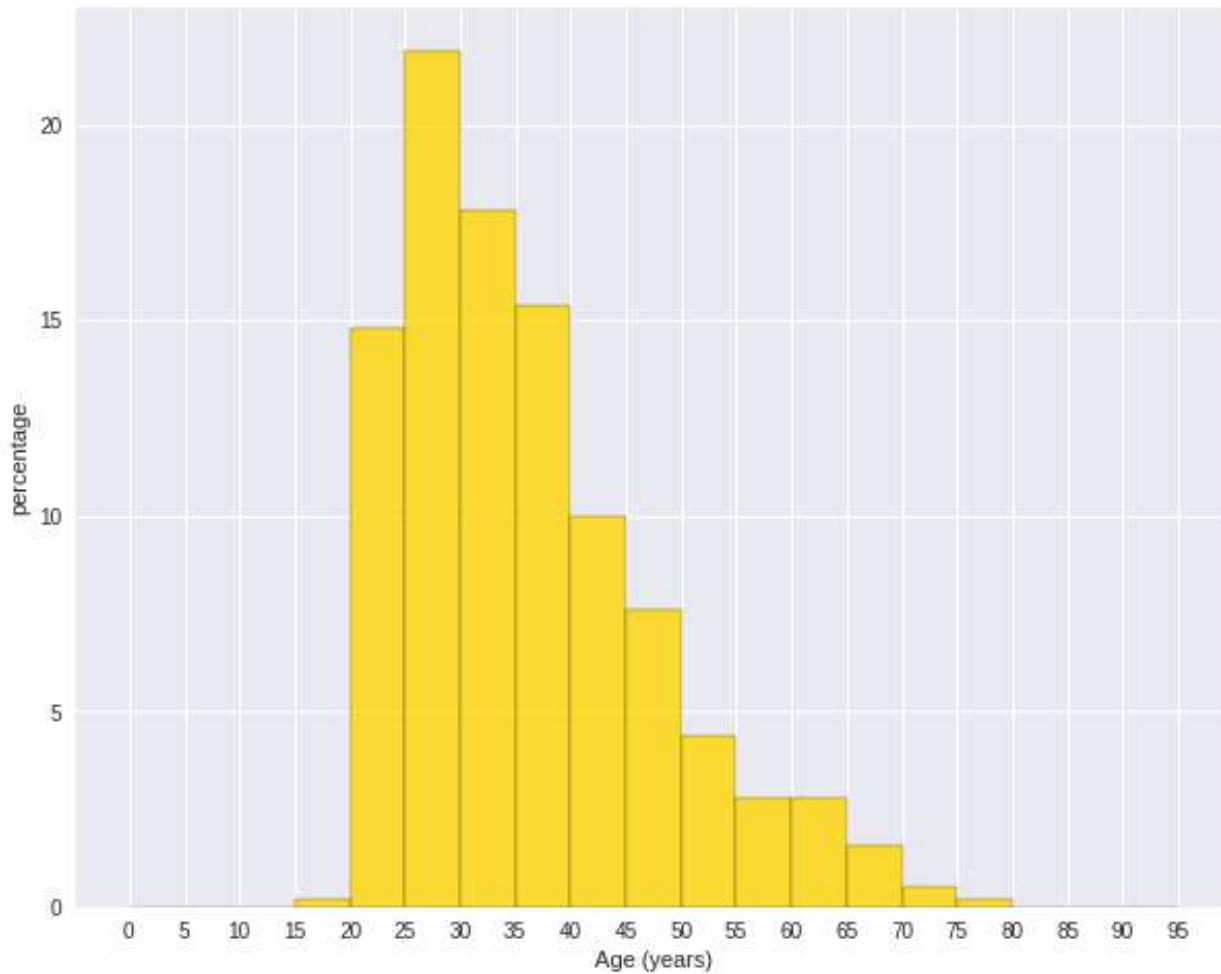
히스토그램과 막대 그래프의 근본적인 차이는 막대 그래프에서는 막대 사이에 간격이 있지만 히스토그램에서는 막대가 서로 인접해 있다는 것입니다. 관심 있는 독자는 히스토그램과 막대 그래프의 차이를 참조하길 바랍니다.

```
import numpy as np
import matplotlib.pyplot as plt
var = 'Age (years)'
```

(다음 페이지에 계속)

(○] 전 페이지에서 계속)

```
x = data1.select(var).rdd.flatMap(lambda x:x).collect()
bins = np.arange(0, 100, 5.0)
plt.figure(figsize=(10,8))
# 데이터의 히스토그램
plt.hist(x, bins, alpha=0.8, histtype='bar', color='gold',
          ec='black', weights=np.zeros_like(x) + 100. / len(x))
plt.xlabel(var)
plt.ylabel('percentage')
plt.xticks(bins)
plt.show()
fig.savefig(var+".pdf", bbox_inches='tight')
```



```
var = 'Age (years)'
x = data1.select(var).rdd.flatMap(lambda x:x).collect()
bins = np.arange(0, 100, 5.0)
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
#####
hist, bin_edges = np.histogram(x,bins,
                               weights=np.zeros_like(x) + 100. / len(x))
# 히스토그램 생성하기

fig = plt.figure(figsize=(20, 8))
ax = fig.add_subplot(1, 2, 1)

# x 축의 정수에 대해 히스토그램 높이 표시
ax.bar(range(len(hist)),hist,width=1,alpha=0.8,ec ='black', color='gold') #
# 눈금을 막대 가운데로 설정
ax.set_xticks([0.5+i for i,j in enumerate(hist)])
# bin의 가장자리가 무엇인지 알려주는 문자열에 xticklabels를 설정
labels =['{}'.format(int(bins[i+1])) for i,j in enumerate(hist)]
labels.insert(0,'0')
ax.set_xticklabels(labels)
plt.xlabel(var)
plt.ylabel('percentage')

#####
hist, bin_edges = np.histogram(x,bins) # 히스토그램 생성하기

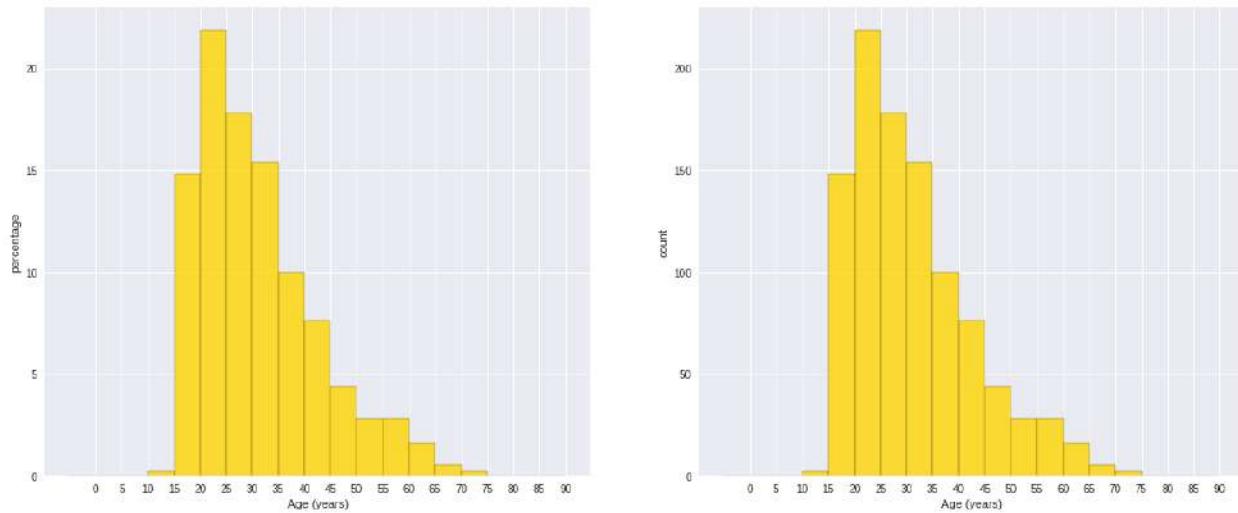
ax = fig.add_subplot(1, 2, 2)
# x 축의 정수에 대해 히스토그램의 높이 표시
ax.bar(range(len(hist)),hist,width=1,alpha=0.8,ec ='black', color='gold')

# # 눈금을 막대 가운데로 설정
ax.set_xticks([0.5+i for i,j in enumerate(hist)])

# bin의 가장자리가 무엇인지 알려주는 문자열로 xticklabels를 설정
labels =['{}'.format(int(bins[i+1])) for i,j in enumerate(hist)]
labels.insert(0,'0')
ax.set_xticklabels(labels)
plt.xlabel(var)
plt.ylabel('count')
plt.suptitle('Histogram of {}: Left with percentage output; Right with count
             output'.format(var), size=16)
plt.show()

fig.savefig(var+".pdf", bbox_inches='tight')
```

Histogram of Age (years): Left with percentage output; Right with count output



때로는 막대가 서로 다른 폭(histogram의 잘못된 인수)을 가지는 히스토그램을 요청하는 사람도 있을 것 입니다. 여러분은 다음과 같은 요령으로 이를 달성할 수 있습니다.

```

var = 'Credit Amount'
plot_data = df.select(var).toPandas()
x= plot_data[var]

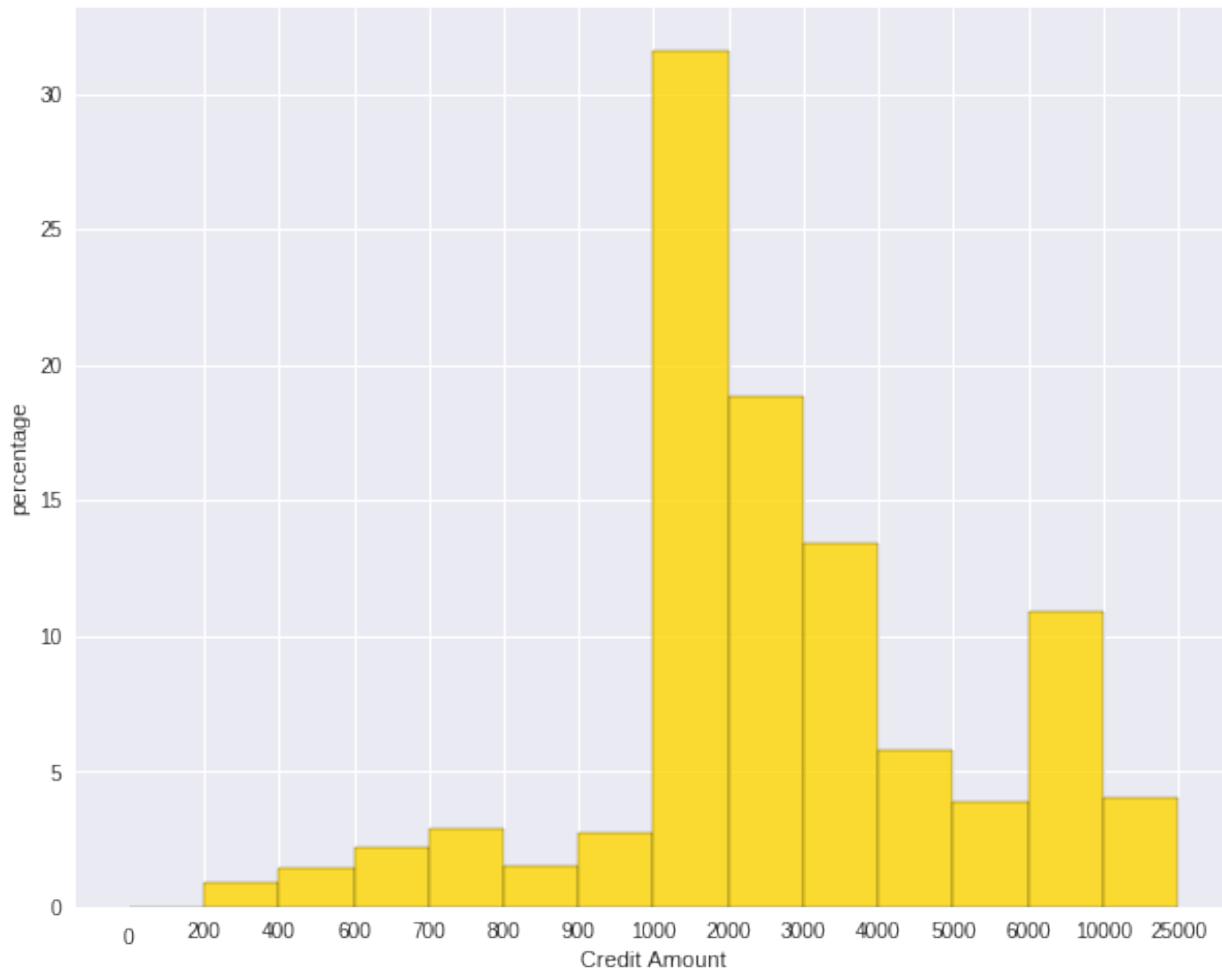
bins =[0, 200, 400, 600, 700, 800, 900, 1000, 2000, 3000, 4000, 5000, 6000, 10000, 25000]

hist, bin_edges = np.histogram(x,bins,weights=np.zeros_like(x) + 100./len(x))
# 히스토그램 생성하기
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(1, 1, 1)
# x 축의 정수에 대해 히스토그램의 높이 표시
ax.bar(range(len(hist)),hist,width=1,alpha=0.8,ec ='black',color = 'gold')

# # 눈금을 막대 가운데로 설정
ax.set_xticks([0.5+i for i,j in enumerate(hist)])

# bin의 가장자리가 무엇인지 알려주는 문자열로 xticklabels를 설정
#labels =['{}k'.format(int(bins[i+1]/1000)) for i,j in enumerate(hist)]
labels =['{}'.format(int(bins[i+1])) for i,j in enumerate(hist)]
labels.insert(0,'0')
ax.set_xticklabels(labels)
# plt.text(-0.6, -1.4, '0')
plt.xlabel(var)
plt.ylabel('percentage')
plt.show()

```



박스 플롯과 바이올린 플롯(Box plot and violin plot)

바이올린 플롯은 튜키의 박스 플롯(1977)과 밀접한 관련이 있지만, 바이올린 플롯은 박스 플롯보다 더 많은 정보를 나타냅니다. 탐색적 분석을 할 때 우리는 표본에 대해 알 수 없습니다. 따라서 표본의 분포를 정규 분포로 가정할 수 없으며 일반적으로 여러분이 가진 빅데이터는 박스플롯 상에서 정규분포가 아님을 확인하실 수 있을 것입니다.

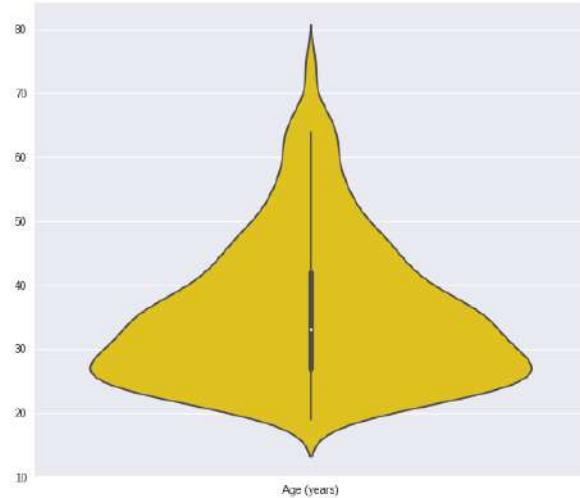
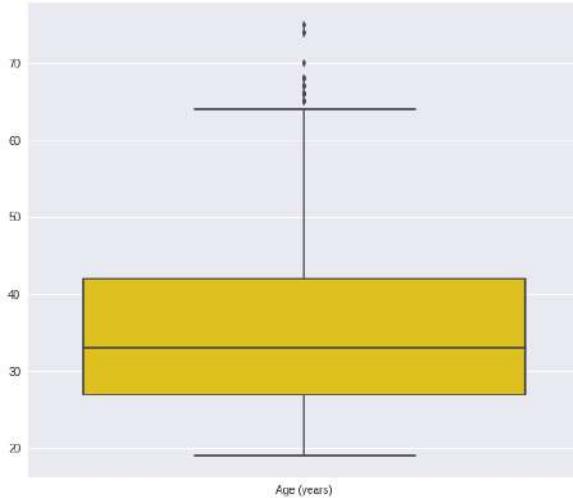
하지만 바이올린 플롯은 심지어 표준 정규 데이터로부터 생성되었어도 확률 밀도함수가 흥미로운 특징을 (그리고 그룹 내 차이) 보여주는 작은 표본 크기에 대해 잠재적으로 오해를 불러일으킬 수 있습니다. 어떤 이는 표본 크기가 250보다 커야 한다고 제안하기도 합니다. 커널 밀도 그래프에서 이상적으로 정확한 분포의 재현을 제공하는 표본 크기는 (250보다 크거나 이상적으로 더 큰)박스 플롯에서 불 명확하거나 보이지 않을 수 있는 비정규성의 다른 형태나 이원 양상과 같은 미묘한 차이를 잠재적으로 보여줄 수 있습니다. 자세한 내용은 [박스 플롯과 바이올린 플롯의 간단한 비교](#)에서 확인할 수 있습니다.

```
import seaborn as sns
x = df.select(var).toPandas()
fig = plt.figure(figsize=(20, 8))
ax = fig.add_subplot(1, 2, 1)
ax = sns.boxplot(data=x)
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
ax = fig.add_subplot(1, 2, 1)
ax = sns.violinplot(data=x)
```



7.1.2 범주형 변수

수치형 변수에 비해 범주형 변수가 탐색하기 훨씬 쉽습니다.

도수분포표(Frequency table)

```
from pyspark.sql import functions as F
from pyspark.sql.functions import rank,sum,col
from pyspark.sql import Window

window = Window.rowsBetween(Window.unboundedPreceding,Window.
    unboundedFollowing)
# withColumn('Percent %',F.format_string("%5.0f%%\n",col('Credit_num')*100/
    col('total'))).\
tab = df.select(['age_class','Credit Amount']).\
    groupBy('age_class').\
    agg(F.count('Credit Amount').alias('Credit_num'),
        F.mean('Credit Amount').alias('Credit_avg'),
        F.min('Credit Amount').alias('Credit_min'),
        F.max('Credit Amount').alias('Credit_max')).\
    withColumn('total',F.sum(col('Credit_num')).over(window)).\
    withColumn('Percent',col('Credit_num')*100/col('total')).\
    drop(col('total'))
```

age_class	Credit_num	Credit_avg	Credit_min	Credit_max	Percent
1	100	100	100	100	100%
2	200	200	200	200	200%
3	300	300	300	300	300%
4	400	400	400	400	400%
5	500	500	500	500	500%
6	600	600	600	600	600%
7	700	700	700	700	700%
8	800	800	800	800	800%
9	900	900	900	900	900%
10	1000	1000	1000	1000	1000%

(다음 페이지에 계속)

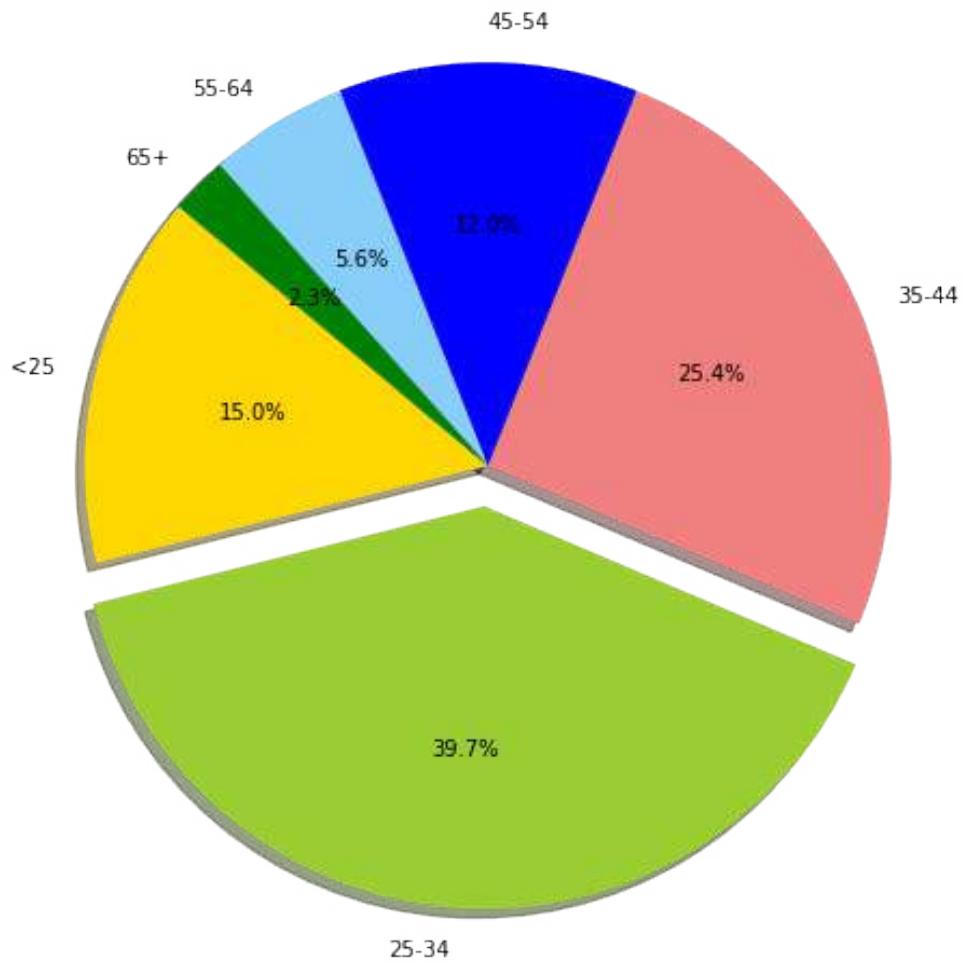
(이전 페이지에서 계속)

45-54 120 3183.066666666666 338 12612 12.0
<25 150 2970.733333333333 276 15672 15.0
55-64 56 3493.660714285714 385 15945 5.6
35-44 254 3403.771653543307 250 15857 25.4
25-34 397 3298.823677581864 343 18424 39.7
65+ 23 3210.1739130434785 571 14896 2.3

파이 플롯(Pie plot)

```
# 차트에 그릴 데이터
import matplotlib.pyplot as plt
labels = tab.select('age_class').rdd.flatMap(lambda x:x).collect()
sizes = tab.select('Percent').rdd.flatMap(lambda x:x).collect()
colors = ['gold', 'yellowgreen', 'lightcoral', 'blue', 'lightskyblue', 'green',
         'red']
explode = (0, 0.1, 0, 0, 0, 0) # explode 1st slice

# 그리기
plt.figure(figsize=(10,8))
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%.1f%',
        shadow=True, startangle=140)
plt.axis('equal')
plt.show()
```



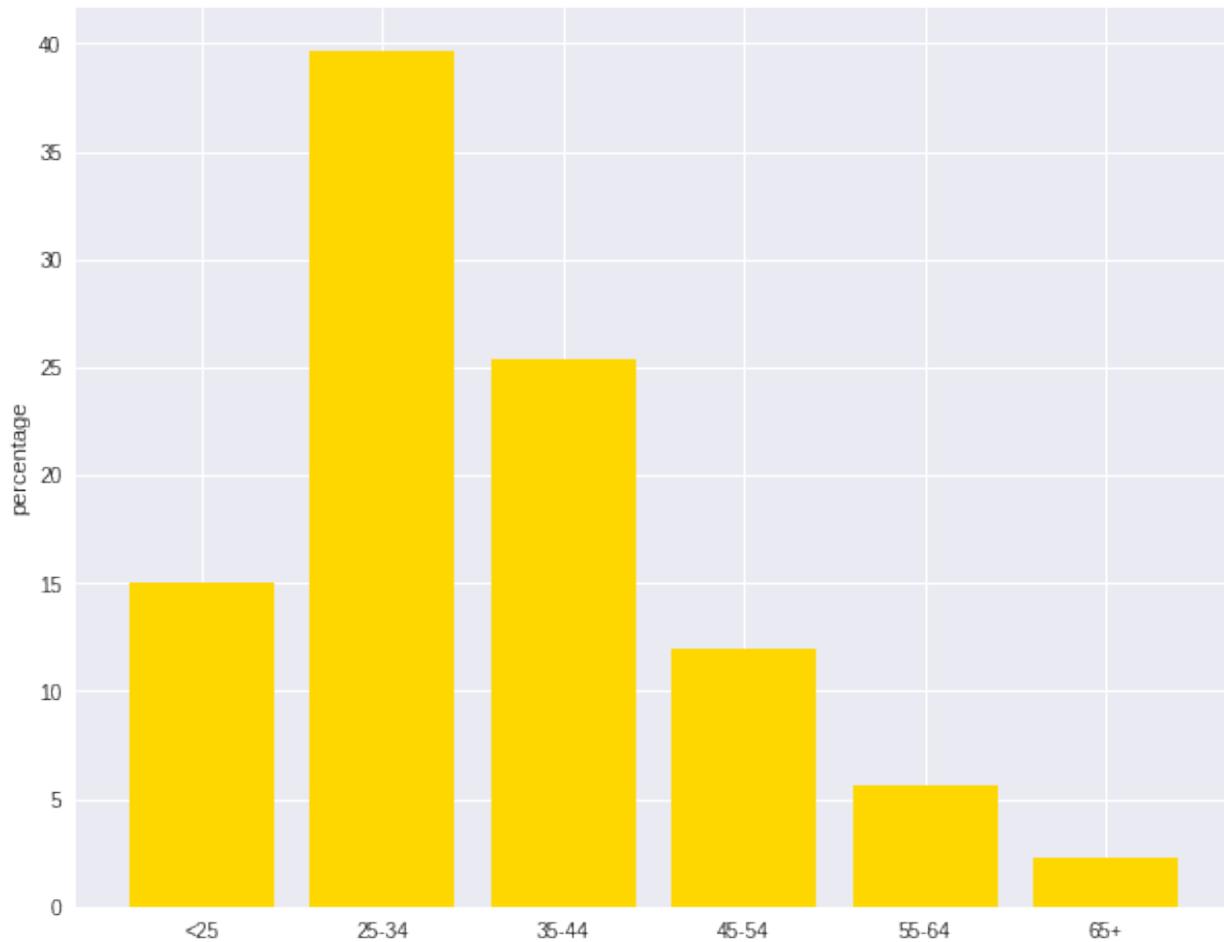
바 플롯(Bar plot)

```
labels = plot_data.age_class
missing = plot_data.Percent
ind = [x for x, _ in enumerate(labels)]

plt.figure(figsize=(10,8))
plt.bar(ind, missing, width=0.8, label='missing', color='gold')

plt.xticks(ind, labels)
plt.ylabel("percentage")

plt.show()
```



```

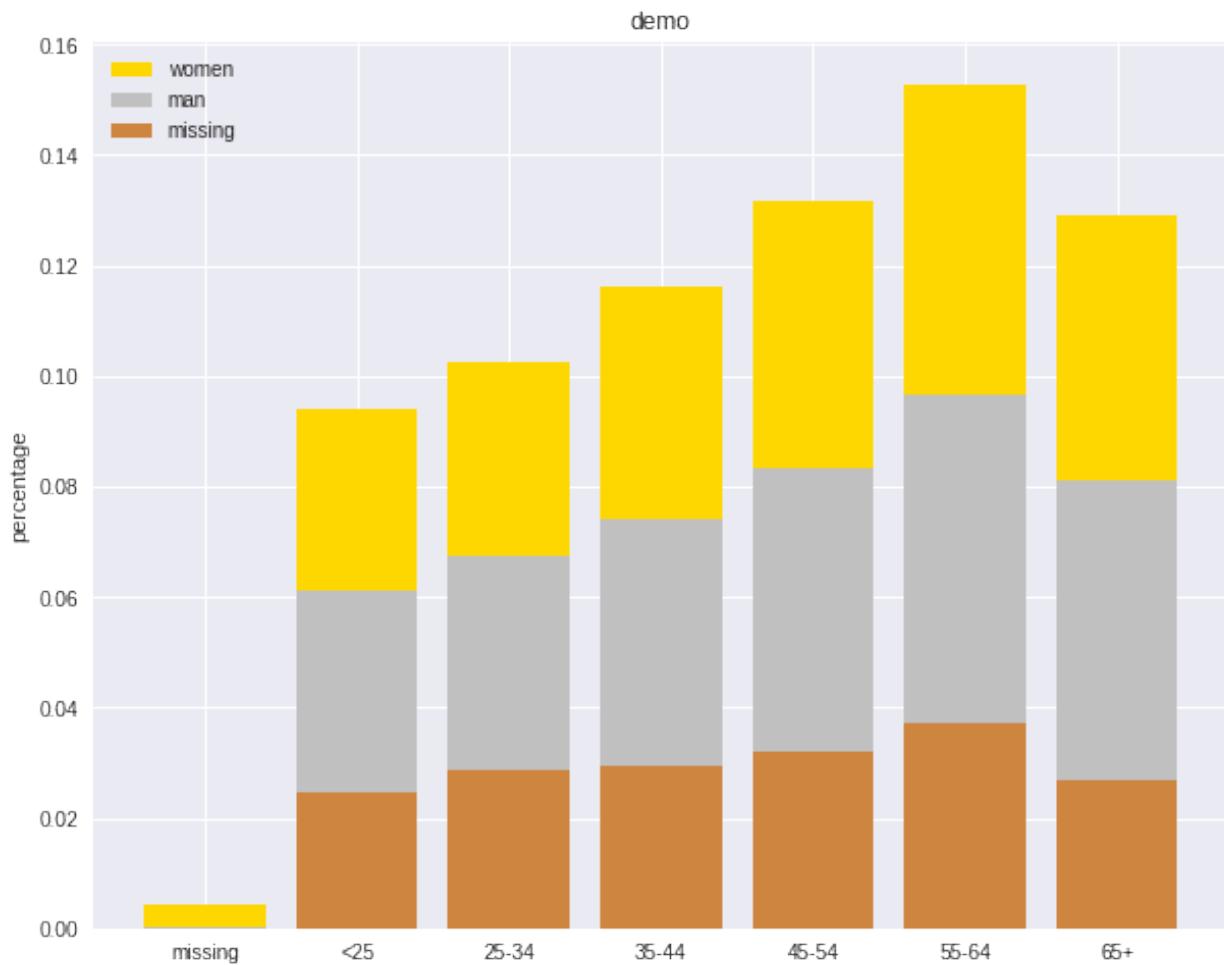
labels = ['missing', '<25', '25-34', '35-44', '45-54', '55-64', '65+']
missing = np.array([0.000095, 0.024830, 0.028665, 0.029477, 0.031918, 0.037073,
                   ↪0.026699])
man = np.array([0.000147, 0.036311, 0.038684, 0.044761, 0.051269, 0.059542, 0.
                   ↪0.054259])
women = np.array([0.004035, 0.032935, 0.035351, 0.041778, 0.048437, 0.056236,
                   ↪0.048091])
ind = [x for x, _ in enumerate(labels)]

plt.figure(figsize=(10,8))
plt.bar(ind, women, width=0.8, label='women', color='gold', ↴
        bottom=man+missing)
plt.bar(ind, man, width=0.8, label='man', color='silver', bottom=missing)
plt.bar(ind, missing, width=0.8, label='missing', color='#CD853F')

plt.xticks(ind, labels)
plt.ylabel("percentage")
plt.legend(loc="upper left")
plt.title("demo")

plt.show()

```



7.2 다변량 분석

이 절에서는 이변량 분석만 설명하겠습니다. 다변량 분석은 변량이 두개일때 부터 해당하기 때문입니다.

7.2.1 수치형 변수 V.S. 수치형 변수

상관 행렬

```
from pyspark.mllib.stat import Statistics
import pandas as pd

corr_data = df.select(num_cols)

col_names = corr_data.columns
features = corr_data.rdd.map(lambda row: row[0:])
corr_mat=Statistics.corr(features, method="pearson")
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
corr_df = pd.DataFrame(corr_mat)
corr_df.index, corr_df.columns = col_names, col_names

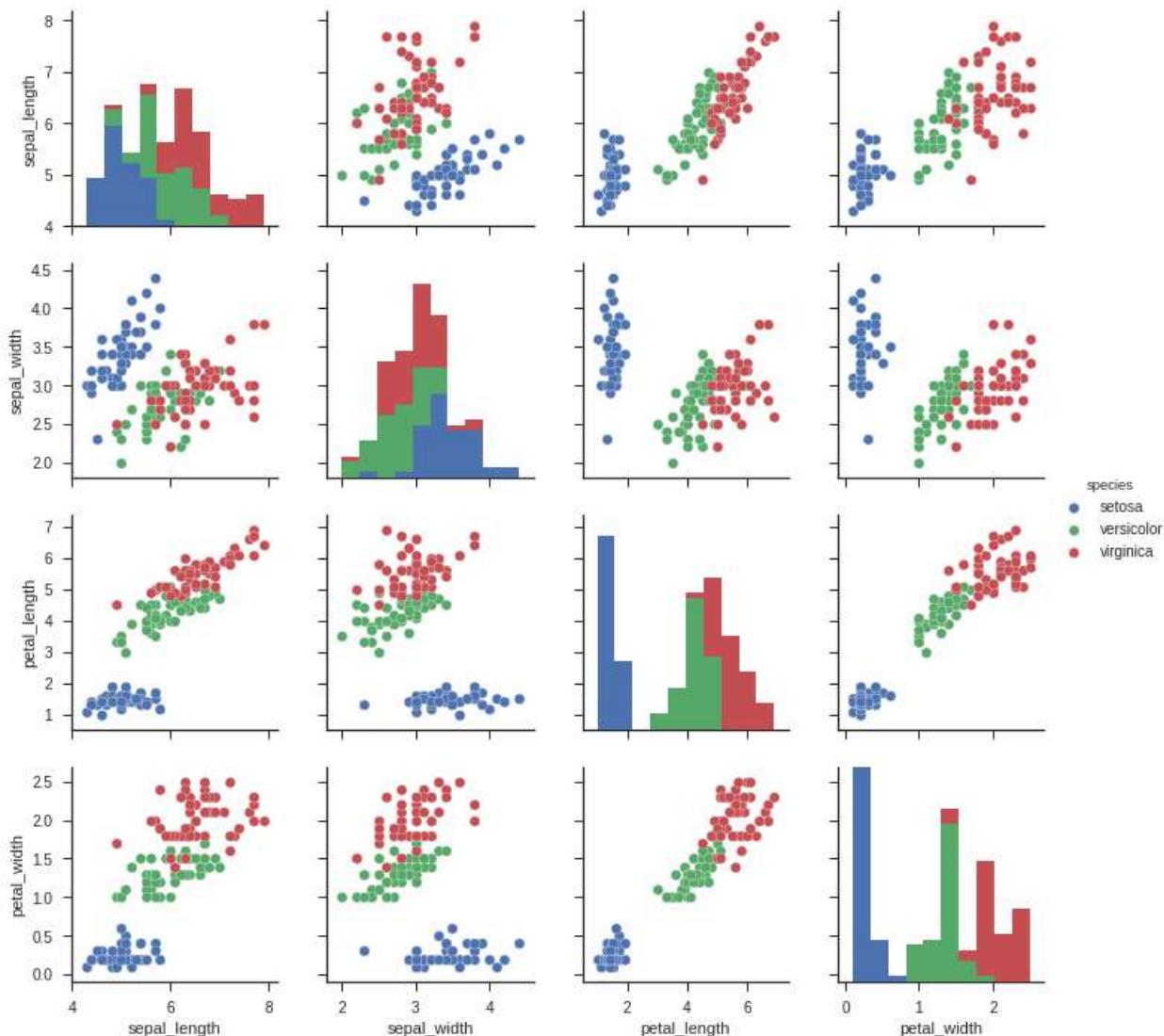
print(corr_df.to_string())
```

Account Balance	No of dependents
1.0	-0.01414542650320914
-0.01414542650320914	1.0

산점도(Scatter Plot)

```
import seaborn as sns
sns.set(style="ticks")

df = sns.load_dataset("iris")
sns.pairplot(df, hue="species")
plt.show()
```



7.2.2 범주형 변수 V.S. 범주형 변수

피어슨 카이제곱 검정 (Pearson's Chi-squared test)

경고: pyspark.ml.stat 은 Spark 2.4.0버전에서만 가능합니다.

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.stat import ChiSquareTest

data = [(0.0, Vectors.dense(0.5, 10.0)),
        (0.0, Vectors.dense(1.5, 20.0)),
        (1.0, Vectors.dense(1.5, 30.0)),
        (0.0, Vectors.dense(3.5, 30.0)),
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
(0.0, Vectors.dense(3.5, 40.0)),
(1.0, Vectors.dense(3.5, 40.0)])
df = spark.createDataFrame(data, ["label", "features"])

r = ChiSquareTest.test(df, "features", "label").head()
print("pValues: " + str(r.pValues))
print("degreesOfFreedom: " + str(r.degreesOfFreedom))
print("statistics: " + str(r.statistics))
```

```
pValues: [0.687289278791, 0.682270330336]
degreesOfFreedom: [2, 3]
statistics: [0.75, 1.5]
```

교차 표(Cross table)

```
df.stat.crosstab("age_class", "Occupation").show()
```

age_class_Occupation	1	2	3	4
<25	4	34	108	4
55-64	1	15	31	9
25-34	7	61	269	60
35-44	4	58	143	49
65+	5	3	6	9
45-54	1	29	73	17

누적 플롯(Stacked plot)

```
labels = ['missing', '<25', '25-34', '35-44', '45-54', '55-64', '65+']
missing = np.array([0.000095, 0.024830, 0.028665, 0.029477, 0.031918, 0.037073,
    ↪0.026699])
man = np.array([0.000147, 0.036311, 0.038684, 0.044761, 0.051269, 0.059542, 0.
    ↪0.054259])
women = np.array([0.004035, 0.032935, 0.035351, 0.041778, 0.048437, 0.056236,
    ↪0.048091])
ind = [x for x, _ in enumerate(labels)]

plt.figure(figsize=(10,8))
plt.bar(ind, women, width=0.8, label='women', color='gold', ↪
    bottom=man+missing)
plt.bar(ind, man, width=0.8, label='man', color='silver', bottom=missing)
plt.bar(ind, missing, width=0.8, label='missing', color='#CD853F')

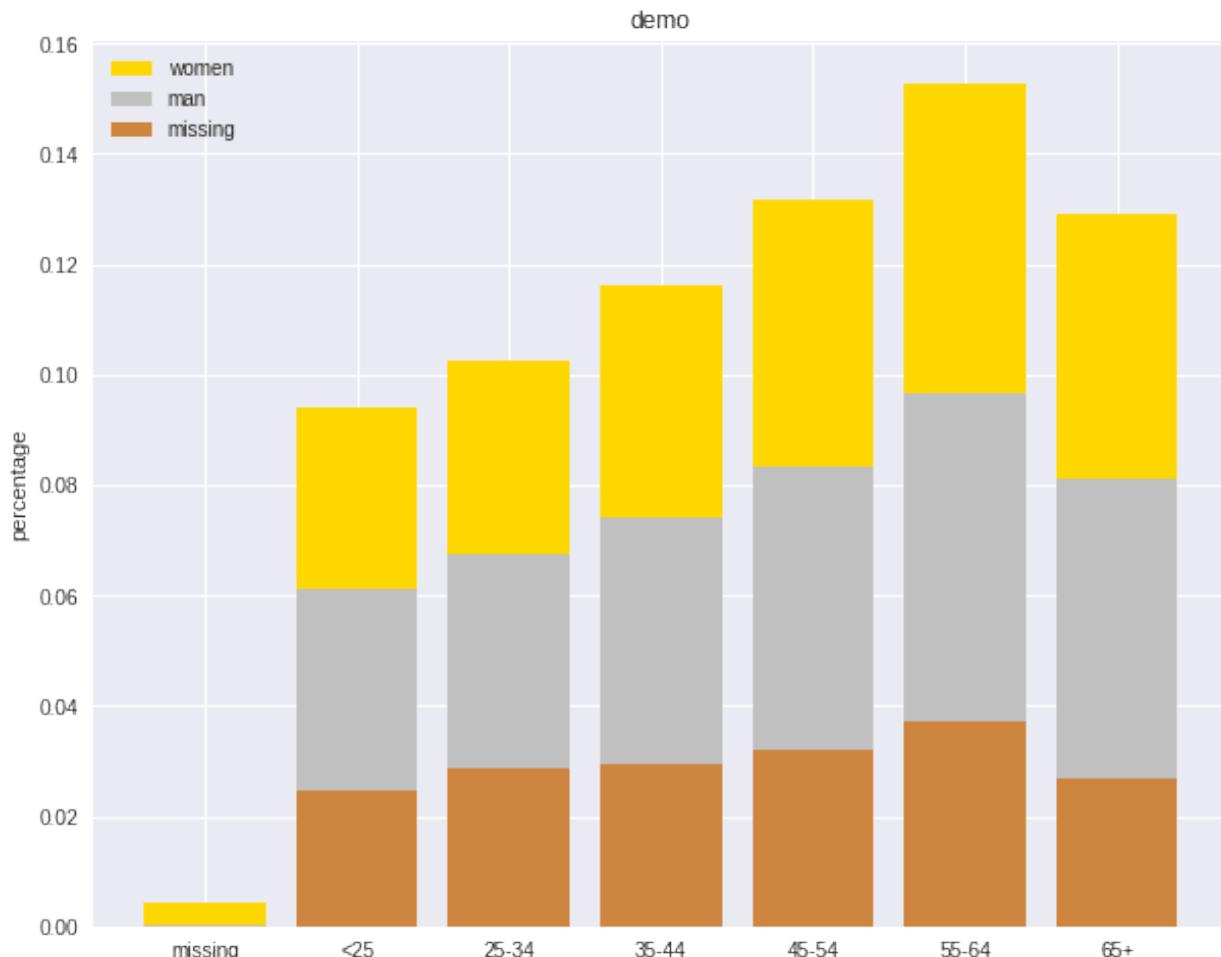
plt.xticks(ind, labels)
plt.ylabel("percentage")
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
plt.legend(loc="upper left")
plt.title("demo")

plt.show()
```



7.2.3 수치형 변수 V.S. 범주형 변수

오차 막대가 있는 선 차트(Line Chart with Error Bars)

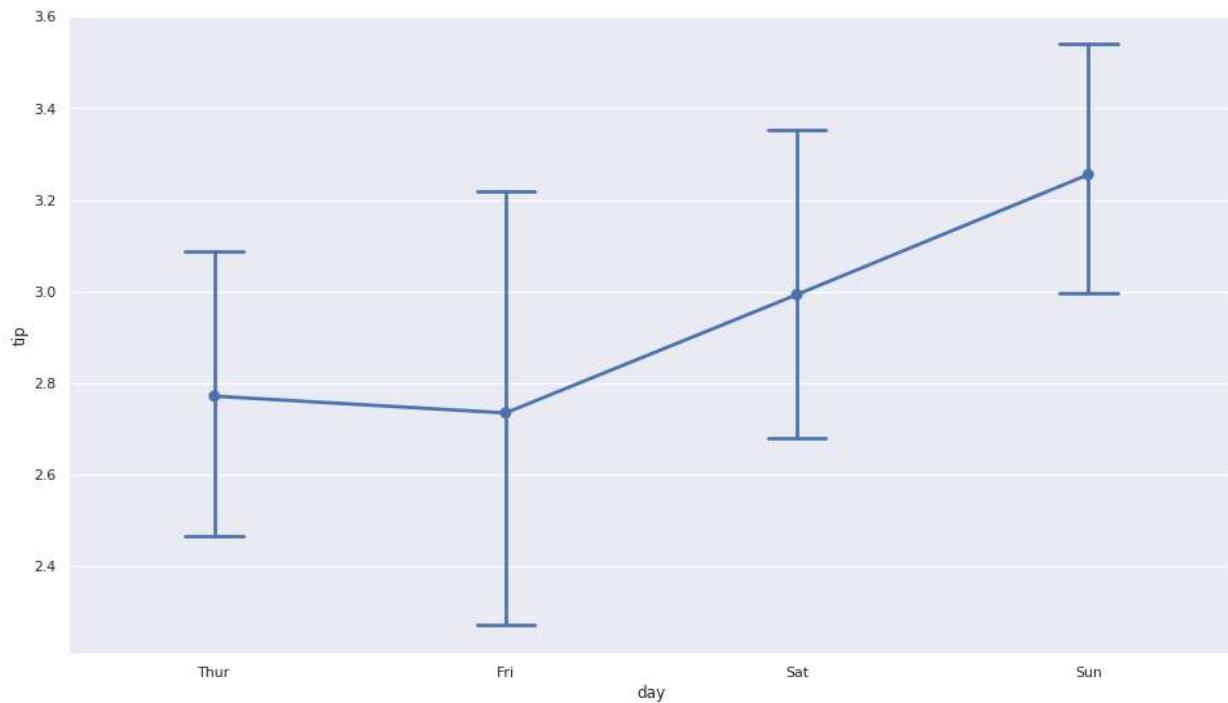
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt import
seaborn as sns
from scipy import stats
%matplotlib inline
tips = sns.load_dataset("tips")
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
sns.set()

ax = sns.pointplot(x="day", y="tip", data=tips, capsize=.2)
plt.show()
```



콤비네이션 차트(Combination Chart)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
%matplotlib inline

plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')
sns.set()

# 월(months) 목록 생성하기
Month = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'June',
         'July', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
# 구성된 평균 온도에 대한 목록 생성하기
Avg_Temp = [35, 45, 55, 65, 75, 85, 95, 100, 85, 65, 45, 35]
# 구성된 평균 강수량 %에 대한 목록 생성하기
Avg_Precipitation_Perc = [.90, .75, .55, .10, .35, .05, .05, .08, .20, .45, .
                           ↪.65, .80]
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

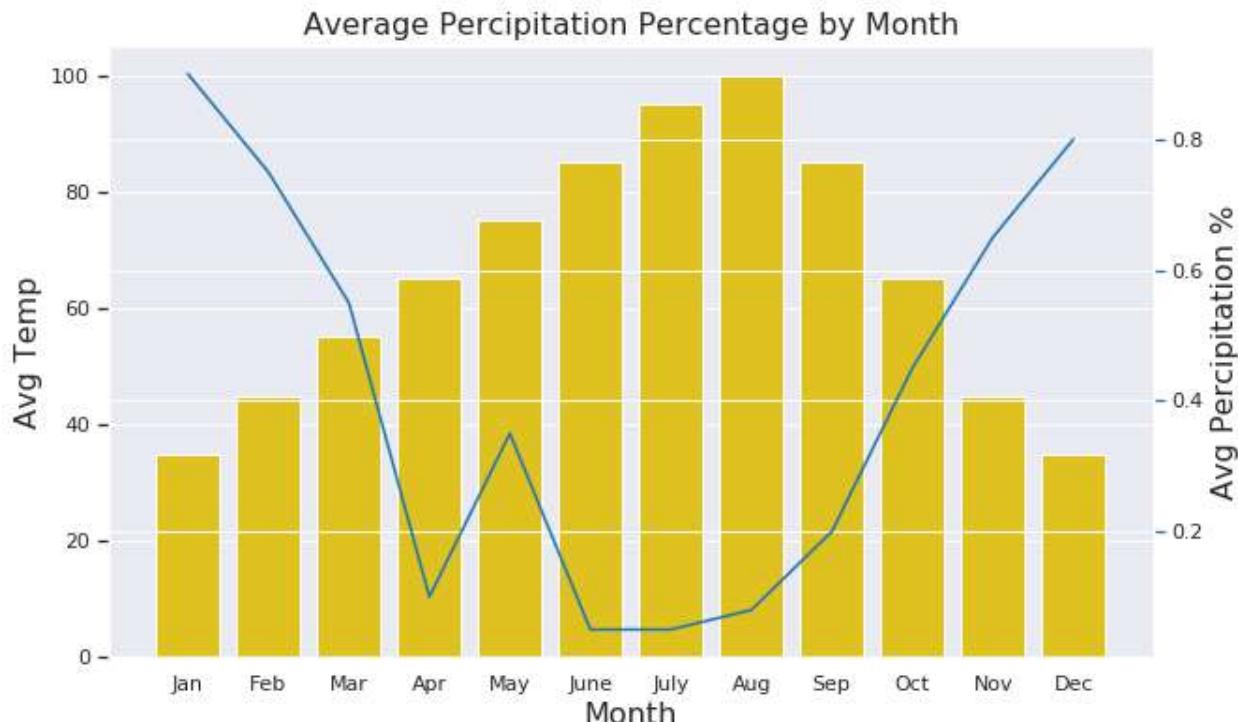
```
#값에 목록들을 할당하기
data = {'Month': Month, 'Avg_Temp': Avg_Temp, 'Avg_Percipitation_Perc': Avg_
→Percipitation_Perc}
#딕셔너리를 데이터 프레임으로 변환하기
df = pd.DataFrame(data)

fig, ax1 = plt.subplots(figsize=(10, 6))
ax1.set_title('Average Percipitation Percentage by Month', fontsize=16)
ax1.tick_params(axis='y')

ax2 = sns.barplot(x='Month', y='Avg_Temp', data = df, color = 'gold')
ax2 = ax1.twinx()
ax2 = sns.lineplot(x='Month', y='Avg_Percipitation_Perc', data = df,_
→sort=False, color='blue')

ax1.set_xlabel('Month', fontsize=16)
ax1.set_ylabel('Avg Temp', fontsize=16)

ax2.tick_params(axis='y', color=color)
ax2.set_ylabel('Avg Percipitation %', fontsize=16)
plt.show()
```



데이터 처리: 특징 (FEATURES)

중국 속담

모든 것은 쉽기 전에 어렵다!

특징 생성은 모델의 성공 또는 실패를 결정하는 모델링에 매우 중요한 단계입니다. 그렇지 않으면, 여러분은 쓰레기 반입; 쓰레기 배출! 을 얻게될 것입니다. 이 기법들은 다음 장에서 다루어질 것이며, 이번 장은 이에 대해 간략히 요약해보겠습니다. 저는 최근에 스파크 공식 웹사이트에서 튜토리얼에 대한 문서화를 정말 잘 해냈다는 것을 알게 되었습니다. 이 장은 특징추출, 변환, 선택에 대해 다뤄보겠습니다.

8.1 특징 추출

8.1.1 TF-IDF

단어 빈도 역 문서 빈도(TF-IDF)는 텍스트 마이닝에서 각 단어의 중요성을 말뭉치 내 한 문서에 반영하기 위해 널리 사용되는 특징 벡터화 방법입니다. 자세한 내용은 다음 웹 사이트에서 확인하실 수 있습니다: <https://spark.apache.org/docs/latest/ml-features#feature-extractors>

스택오버플로우 TF: 해싱TF 및 CountVectorizer를 사용하여 단어 빈도 벡터를 생성할 수 있습니다. 몇 가지 중요한 차이점이 있습니다:

- a. **부분 복원(CountVectorizer) vs 비복원(해싱TF)** - 해싱은 복원이 안되므로 해시 벡터로부터 원래 입력을 복원할 수 없습니다. 모델(색인)이 있는 셀 수 있는 다른 카운트 벡터를 사용하여 정렬되지 않은 입력을 복원할 수 있습니다. 따라서 해시 입력을 사용하여 만든 모델은 해석하고 모니터링하기 훨씬 어려울 수 있습니다.
- b. **메모리 및 계산 오버헤드** - 해싱TF는 단일 데이터 스캔만을 필요로 하며 본래 입력과 벡터 이외의 추가적인 메모리가 필요하지 않습니다. CountVectorizer는 모델을 구축하기 위해 데이터를 추가로 스캔해야 하며 어휘(색인)를 저장하기 위한 추가 메모리가 필요합니다. 유니그램 언어 모델의 경우 일반적으로 문제가 되지 않지만 큰 n-그램일 수록 런타임과 메모리의 관점에서 엄청 비싸거나 실행 불가능할 수 있습니다.
- c. **해싱은 벡터, 해싱 함수 및 문서의 크기에 따라 달라집니다.** 계산은 벡터, 훈련 말뭉치 및 문서의 크기에 따라 달라집니다.

d. 정보 손실의 원천 - 해싱TF의 경우 충돌 가능성 있는 차원 축소입니다. CountVectorizer는 자주 사용하지 않는 토큰을 삭제합니다. 다운스트림 모델에 영향을 미치는 방법은 특정 사용 사례와 데이터에 따라 달라집니다.

해싱TF와 CountVectorizer는 단어 빈도 벡터를 생성하는 데 사용되는 두 가지 일반적인 알고리즘입니다. 두 알고리즘은 기본적으로 문서를 직접 또는 추가처리를 통해 LDA, MinHash for Jaccard Distance, Cosine Distance와 같은 다른 알고리즘에 반영될 수 있는 숫자 표현으로 변환합니다.

- t : 용어(term)
- d : 문서(document)
- D : 말뭉치(corpus)
- $|D|$: 말뭉치 내 원소 개수
- $TF(t, d)$: 단어 빈도: 문서 d 에서 용어 t 의 출현 빈도
- $DF(t, D)$: 문서 빈도: 용어 t 를 포함하는 문서의 개수
- $IDF(t, D)$: 역문서 빈도: 용어가 제공하는 정보의 양을 나타내는 수치적 측도

$$IDF(t, D) = \log \frac{|D| + 1}{DF(t, D) + 1}$$

- $TFIDF(t, d, D)$ TF와 IDF의 곱

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D)$$

예제를 살펴보겠습니다:

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import HashingTF, IDF, Tokenizer

sentenceData = spark.createDataFrame([
    (0, "Python python Spark Spark"),
    (1, "Python SQL")],
    ["document", "sentence"])
```

```
sentenceData.show(truncate=False)
+-----+-----+
| document | sentence           |
+-----+-----+
| 0       | Python python Spark Spark |
| 1       | Python SQL               |
+-----+-----+
```

그러면:

- $TF(python, document1) = 1, TF(spark, document1) = 2$
- $DF(Spark, D) = 2, DF(sql, D) = 1$
- IDF:

$$IDF(python, D) = \log \frac{|D| + 1}{DF(t, D) + 1} = \log\left(\frac{2 + 1}{2 + 1}\right) = 0$$

$$IDF(spark, D) = \log \frac{|D| + 1}{DF(t, D) + 1} = \log\left(\frac{2 + 1}{1 + 1}\right) = 0.4054651081081644$$

$$IDF(sql, D) = \log \frac{|D| + 1}{DF(t, D) + 1} = \log\left(\frac{2 + 1}{1 + 1}\right) = 0.4054651081081644$$

- TFIDF

$$TFIDF(python, document1, D) = 3 * 0 = 0$$

$$TFIDF(spark, document1, D) = 2 * 0.4054651081081644 = 0.8109302162163288$$

$$TFIDF(sql, document1, D) = 1 * 0.4054651081081644 = 0.4054651081081644$$

Countvectorizer

[스택오버플로우 \(Stackoverflow\)](#) TF: CountVectorizer 및 CountVectorizerModel은 텍스트 문서 모음을 토큰 카운트 벡터로 변환하는 것을 목표로 합니다. 사전에 딕셔너리를 사용할 수 없는 경우, CountVectorizer는 어휘를 추출하는 추정기로 사용될 수 있으며 CountVectorizerModel을 생성합니다. 이 모델은 어휘를 통해 문서에 대한 희소 표현을 생성하며 이는 LDA와 같은 다른 알고리즘에 반영될 수 있습니다..

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import CountVectorizer
from pyspark.ml.feature import HashingTF, IDF, Tokenizer

sentenceData = spark.createDataFrame([
    (0, "Python python Spark Spark"),
    (1, "Python SQL")),
    ["document", "sentence"])

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
vectorizer = CountVectorizer(inputCol="words", outputCol="rawFeatures")

idf = IDF(inputCol="rawFeatures", outputCol="features")

pipeline = Pipeline(stages=[tokenizer, vectorizer, idf])

model = pipeline.fit(sentenceData)
```

```
import numpy as np

total_counts = model.transform(sentenceData) \
    .select('rawFeatures') .rdd \
    .map(lambda row: row['rawFeatures'].toArray()) \
    .reduce(lambda x,y: [x[i]+y[i] for i in range(len(y))])

vocabList = model.stages[1].vocabulary
d = {'vocabList':vocabList, 'counts':total_counts}
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
spark.createDataFrame(np.array(list(d.values()))).T.tolist(),list(d.keys())).
    show()
```

```
counts = model.transform(sentenceData).select('rawFeatures').collect()
counts

[Row(rawFeatures=SparseVector(8, {0: 1.0, 1: 1.0, 2: 1.0})),
 Row(rawFeatures=SparseVector(8, {0: 1.0, 1: 1.0, 4: 1.0})),
 Row(rawFeatures=SparseVector(8, {0: 1.0, 3: 1.0, 5: 1.0, 6: 1.0, 7: 1.0}))]
```

```
+-----+-----+
| vocabList | counts |
+-----+-----+
|    python |    3.0 |
|     spark |    2.0 |
|      sql |    1.0 |
+-----+-----+
```

```
model.transform(sentenceData).show(truncate=False)
```

```
+-----+-----+-----+-----+
|-----+-----+-----+-----+
| document | sentence           | words          | ↵
|-----+-----+-----+-----+
|-----+-----+-----+-----+
| rawFeatures | features           |               | ↵
|-----+-----+-----+-----+
|-----+-----+-----+-----+
| 0   | Python python Spark Spark | [python, python, spark, spark] | (3, [0, 1],
|-----+-----+-----+-----+
| 1   | Python SQL             | [python, sql]    | (3, [0, 2],
|-----+-----+-----+-----+
```

```
from pyspark.sql.types import ArrayType, StringType

def termsIdx2Term(vocabulary):
    def termsIdx2Term(termIndices):
        return [vocabulary[int(index)] for index in termIndices]
    return udf(termsIdx2Term, ArrayType(StringType()))

vectorizerModel = model.stages[1]
vocabList = vectorizerModel.vocabulary
vocabList
```

```
['python', 'spark', 'sql']
```

```
rawFeatures = model.transform(sentenceData).select('rawFeatures')
rawFeatures.show()
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
+-----+
|      rawFeatures |
+-----+
| (3, [0,1],[2.0,2.0]) |
| (3, [0,2],[1.0,1.0]) |
+-----+
```

```
from pyspark.sql.functions import udf
import pyspark.sql.functions as F
from pyspark.sql.types import StringType, DoubleType, IntegerType

indices_udf = udf(lambda vector: vector.indices.tolist(),  

    ↪ArrayType(IntegerType()))
values_udf = udf(lambda vector: vector.toArray().tolist(),  

    ↪ArrayType(DoubleType()))

rawFeatures.withColumn('indices', indices_udf(F.col('rawFeatures')))\  

    .withColumn('values', values_udf(F.col('rawFeatures')))\  

    .withColumn("Terms", termsIdx2Term(vocabList) ("indices")).show()
```

```
+-----+-----+-----+-----+
|      rawFeatures | indices |      values |      Terms |
+-----+-----+-----+-----+
| (3, [0,1],[2.0,2.0]) | [0, 1] | [2.0, 2.0, 0.0] | [python, spark] |
| (3, [0,2],[1.0,1.0]) | [0, 2] | [1.0, 0.0, 1.0] | [python, sql] |
+-----+-----+-----+-----+
```

해싱TF

스택오버플로우 TF: 해싱TF는 용어 집합을 가져와 고정 길이의 특징(feature) 벡터로 바꾸는 변환자입니다. 텍스트 처리에서 "용어 집합"은 단어 모음(bag of words) 일 수 있습니다. 해싱TF는 해싱 트리를 활용합니다. 원시 특징은(raw feature) 해시 함수를 적용하여 인덱스(용어)로 매핑됩니다. 여기서 사용되는 해시 함수는 MurmurHash 3입니다. 그런 다음 단어 빈도는 매핑된 인덱스를 기반으로 계산됩니다. 이 접근 방식은 대규모 말뭉치에서 비용이 많이 들 수 있는 전역적 단어 대 인덱스 맵(global term-to-index map)을 계산할 필요가 없지만, 다른 원시 특징이 해싱 후 동일한 단어가 될 수 있는 잠재적 해싱 충돌로 어려움을 겪습니다.

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import HashingTF, IDF, Tokenizer

sentenceData = spark.createDataFrame([
    (0, "Python python Spark Spark"),
    (1, "Python SQL")],
    ["document", "sentence"])

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
vectorizer = HashingTF(inputCol="words", outputCol="rawFeatures", ↳ numFeatures=5)

idf = IDF(inputCol="rawFeatures", outputCol="features")

pipeline = Pipeline(stages=[tokenizer, vectorizer, idf])

model = pipeline.fit(sentenceData)
model.transform(sentenceData).show(truncate=False)
```

document	sentence	words	
rawFeatures		features	
0	Python python Spark Spark	[python, python, spark, spark]	(5, [0, 4], [2.0, 2.0]) (5, [0, 4], [0.8109302162163288, 0.0])
1	Python SQL	[python, sql]	(5, [1, 4], [1.0, 1.0]) (5, [1, 4], [0.4054651081081644, 0.0])

8.1.2 Word2Vec

워드 임베딩 (Word Embeddings)

Word2Vec는 워드 임베딩을 구현하는 인기 있는 방법 중 하나입니다. 워드 임베딩은(지금까지 제가 읽은 튜토리얼 중 최고입니다. 다음의 단어와 이미지 내용은 듀크 대학의 크리스 베일 박사로부터 받은 것입니다. 따라서 저작권은 크리스 베일 박사(Duke University, Chris Bail)가 소유하고 있습니다) 자동 텍스트 분석의 세계에서 유사성을 식별하는 데 사용될 수 있다는 것이 입증되었을 쯤 명성을 얻었습니다. 그림 1은 모델에 의해 생성된 3차원 공간에 개별 단어가 표시되는 워드 임베딩 모델의 결과를 보여줍니다. 워드 내장 모델은 이 공간에서 단어의 인접성을 조사함으로써 "남자가 여자에게 왕처럼 여왕에게"와 같은 유추를 완성할 수 있습니다. 다수 워드 임베딩 모델의 결과에 대해 자세히 알아보고 싶다면 GloVe라는 워드 임베딩 모델을 사용하여 만든 영단어의 대부분을 구현한 환상적인 시각화 결과를 확인해보시길 바랍니다.

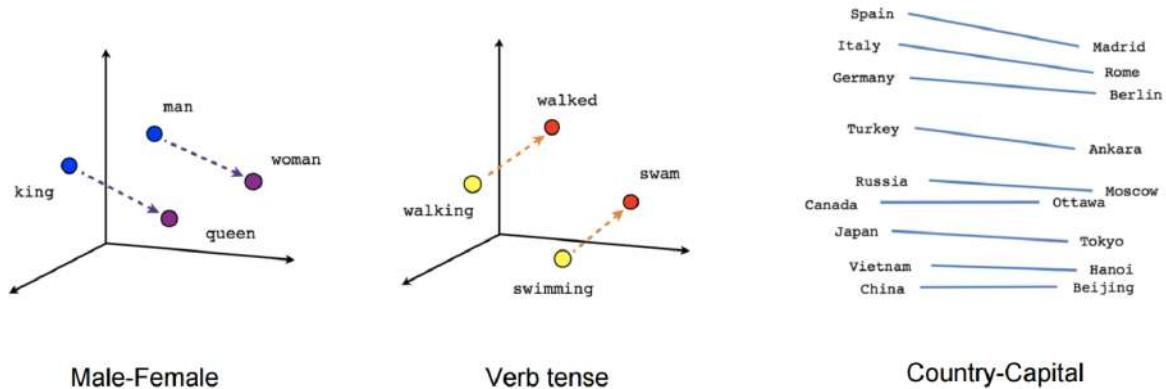


그림. 1: 워드 임베딩 모델 결과

문맥 윈도우(The Context Window)

워드 임베딩은 "문맥 윈도우"이라고 불리는 것 안에서 발생하는 단어들을 식별함으로써 만들어집니다. 아래 그림은 한 문장에 대해 다양한 길이의 문맥 윈도들을 보여줍니다. 문맥 윈도는 워드 임베딩 모델을 훈련시키는 데 사용될 중점 또는 "중심" 단어의 앞 뒤에 있는 단어 문자열로 정의됩니다. 각 중심 단어와 문맥 단어는 데이터 집합 내에서 고유한 단어의 유무를 설명하는 수치형 벡터로 표현될 수 있으며, 이것이 단어 내장 모델이 종종 "word 벡터" 모델 또는 "word2vec" 모델로 설명되는 이유일 것입니다.

: Center Word

: Context Word

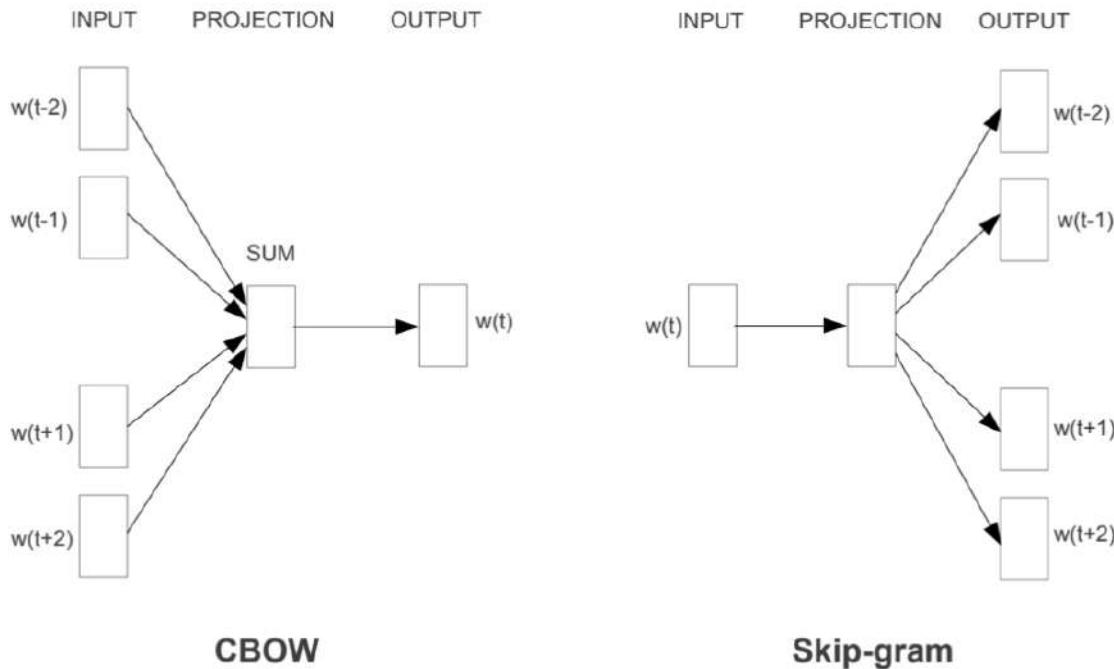
c=0 The cute cat jumps over the lazy dog.

c=1 The cute cat jumps over the lazy dog.

c=2 The cute cat jumps over the lazy dog.

두 가지 유형의 임베딩 모델

워드 임베딩은 일반적으로 CBOW(Continuous Bag of Words) 또는 스kip-gram 모형 두 가지 방법 중 하나로 수행됩니다. 아래 그림은 두 모델 간의 차이를 보여줍니다. CBOW 모델은 문맥 원도 단어들을 읽고 가장 가능성이 높은 중심 단어를 예측하려 합니다. 스kip-gram 모형은 주어진 중심 단어에서 문맥 단어들을 예측합니다. 위의 예는 텍스트 내에서 패턴을 식별하여 다차원 공간에서 나타내려는 사람들에게 가장 유용한 스kip-gram 모델을 사용하여 만들어졌습니다. 반면, CBOW 모델은 웹 검색 예측과 같은 실제 응용 분야에서 더 유용합니다.



PySpark에서 워드 임베딩 모형

```
from pyspark.ml.feature import Word2Vec, Tokenizer

from pyspark.ml import Pipeline

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
word2Vec = Word2Vec(vectorSize=3, minCount=0, inputCol="words", outputCol=
                     "feature")

pipeline = Pipeline(stages=[tokenizer, word2Vec])

model = pipeline.fit(sentenceData)
result = model.transform(sentenceData)
```

label	sentence	words	feature
0.0	I love Spark	[i, love, spark]	[0.05594437588782...]
0.0	I love python	[i, love, python]	[-0.0350368790871...]
1.0	I think ML is awesome	[i, think, ml, is ...]	[0.01242086507845...]

```
w2v = model.stages[1]
w2v.getVectors().show()
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
+-----+-----+
|word    |vector
+-----+-----+
|is      | [0.13657838106155396, 0.060924094170331955, -0.03379475697875023] |
|awesome| [0.037024181336164474, -0.023855900391936302, 0.0760037824511528] |
|i       | [-0.0014482572441920638, 0.049365971237421036, 0.12016955763101578] |
|ml     | [-0.14006119966506958, 0.01626444421708584, 0.042281970381736755] |
|spark   | [0.1589149385690689, -0.10970081388950348, -0.10547549277544022] |
|think   | [0.030011219903826714, -0.08994936943054199, 0.16471518576145172] |
|love    | [0.01036644633859396, -0.017782460898160934, 0.08870164304971695] |
|python  | [-0.11402882635593414, 0.045119188725948334, -0.029877422377467155] |
+-----+-----+
```

```
from pyspark.sql.functions import format_number as fmt
w2v.findSynonyms("could", 2).select("word", fmt("similarity", 5).alias(
    "similarity")).show()
```

```
+-----+-----+
| word|similarity|
+-----+-----+
| classes| 0.90232|
|      i| 0.75424|
+-----+-----+
```

8.1.3 FeatureHasher

```
from pyspark.ml.feature import FeatureHasher

dataset = spark.createDataFrame([
    (2.2, True, "1", "foo"),
    (3.3, False, "2", "bar"),
    (4.4, False, "3", "baz"),
    (5.5, False, "4", "foo")
], ["real", "bool", "stringNum", "string"])

hasher = FeatureHasher(inputCols=["real", "bool", "stringNum", "string"],
                       outputCol="features")

featurized = hasher.transform(dataset)
featurized.show(truncate=False)
```

```
+----+----+----+----+
|real|bool |stringNum|string|features
|    |    |
+----+----+----+----+
| 2.2 |true | 1          | foo    | (262144, [174475, 247670, 257907, 262126], [2.2, 1.0, 1.0, 1.0]) |
+----+----+----+----+
```

(다음 페이지에서 계속)

(이전 페이지에서 계속)

```
| 3.3 | false|2          |bar    |(262144,[70644,89673,173866,174475],[1.0,1.0,1.0,
˓→3.3]) |
| 4.4 | false|3          |baz    |(262144,[22406,70644,174475,187923],[1.0,1.0,4.4,
˓→1.0]) |
| 5.5 | false|4          |foo    |(262144,[70644,101499,174475,257907],[1.0,1.0,5.
˓→5,1.0]) |
+----+----+----+----+-----+
˓→-----+
```

8.1.4 RFormula

```
from pyspark.ml.feature import RFormula

dataset = spark.createDataFrame(
    [(7, "US", 18, 1.0),
     (8, "CA", 12, 0.0),
     (9, "CA", 15, 0.0)],
    ["id", "country", "hour", "clicked"])

formula = RFormula(
    formula="clicked ~ country + hour",
    featuresCol="features",
    labelCol="label")

output = formula.fit(dataset).transform(dataset)
output.select("features", "label").show()
```

```
+----+----+
|  features|label|
+----+----+
|[0.0,18.0]| 1.0|
|[1.0,12.0]| 0.0|
|[1.0,15.0]| 0.0|
+----+----+
```

8.2 특징 변환

8.2.1 Tokenizer

```
from pyspark.ml.feature import Tokenizer, RegexTokenizer
from pyspark.sql.functions import col, udf
from pyspark.sql.types import IntegerType

sentenceDataFrame = spark.createDataFrame([
    (0, "Hi I heard about Spark"),
    (1, "I wish Java could use case classes"),
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
(2, "Logistic, regression, models, are, neat")
], ["id", "sentence"])

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")

regexTokenizer = RegexTokenizer(inputCol="sentence", outputCol="words", 
    pattern="\w+")
# 또는 pattern="\w+", gaps=False)

countTokens = udf(lambda words: len(words), IntegerType())

tokenized = tokenizer.transform(sentenceDataFrame)
tokenized.select("sentence", "words") \
    .withColumn("tokens", countTokens(col("words"))).show(truncate=False)

regexTokenized = regexTokenizer.transform(sentenceDataFrame)
regexTokenized.select("sentence", "words") \
    .withColumn("tokens", countTokens(col("words"))).show(truncate=False)
```

sentence	words
Hi I heard about Spark	[hi, i, heard, about, spark]
I wish Java could use case classes	[i, wish, java, could, use, case,
Logistic, regression, models, are, neat	[logistic, regression, models, are, neat]
sentence	words
Hi I heard about Spark	[hi, i, heard, about, spark]
I wish Java could use case classes	[i, wish, java, could, use, case,
Logistic, regression, models, are, neat	[logistic, regression, models, are,
neat]	[neat]

8.2.2 불용어 제거

```
from pyspark.ml.feature import StopWordsRemover

sentenceData = spark.createDataFrame([
    (0, ["I", "saw", "the", "red", "balloon"]),
    (1, ["Mary", "had", "a", "little", "lamb"])
], ["id", "raw"])

remover = StopWordsRemover(inputCol="raw", outputCol="removed")
remover.transform(sentenceData).show(truncate=False)
```

id	raw	removed
0	[I, saw, the, red, balloon]	[saw, red, balloon]
1	[Mary, had, a, little, lamb]	[Mary, little, lamb]

8.2.3 N 그램

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import CountVectorizer
from pyspark.ml.feature import HashingTF, IDF, Tokenizer

from pyspark.ml.feature import NGram

sentenceData = spark.createDataFrame([
    (0.0, "I love Spark"),
    (0.0, "I love python"),
    (1.0, "I think ML is awesome")],
    ["label", "sentence"])

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
ngram = NGram(n=2, inputCol="words", outputCol="ngrams")

idf = IDF(inputCol="rawFeatures", outputCol="features")

pipeline = Pipeline(stages=[tokenizer, ngram])

model = pipeline.fit(sentenceData)

model.transform(sentenceData).show(truncate=False)
```

label	sentence	words	ngrams

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
| 0.0 | I love Spark           | [i, love, spark]           | [i love, love spark]_
|    ↓                         |                               |
| 0.0 | I love python          | [i, love, python]          | [i love, love_
|    ↓python]                  |                               |
| 1.0 | I think ML is awesome | [i, think, ml, is, awesome] | [i think, think ml,_
|    ↓ml is, is awesome]      |                               |
+-----+-----+-----+-----+
|    ↓
```

8.2.4 ⓠ 진화기 (Binarizer)

```
from pyspark.ml.feature import Binarizer

continuousDataFrame = spark.createDataFrame([
    (0, 0.1),
    (1, 0.8),
    (2, 0.2),
    (3, 0.5)
], ["id", "feature"])

binarizer = Binarizer(threshold=0.5, inputCol="feature", outputCol="binarized_"
                     ↓feature")

binarizedDataFrame = binarizer.transform(continuousDataFrame)

print("Binarizer output with Threshold = %f" % binarizer.getThreshold())
binarizedDataFrame.show()
```

```
Binarizer output with Threshold = 0.500000
+---+-----+
| id|feature|binarized_feature|
+---+-----+
|  0|    0.1|        0.0|
|  1|    0.8|        1.0|
|  2|    0.2|        0.0|
|  3|    0.5|        0.0|
+---+-----+
```

8.2.5 Bucketizer

[Bucketizer](<https://spark.apache.org/docs/latest/ml-features.html#bucketizer>) 연속 값을 갖는 특징 열을 사용자가 지정하는 특징 버킷 열로 변환합니다.

```
from pyspark.ml.feature import QuantileDiscretizer, Bucketizer

data = [(0, 18.0), (1, 19.0), (2, 8.0), (3, 5.0), (4, 2.0)]
df = spark.createDataFrame(data, ["id", "age"])
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
print(df.show())

splits = [-float("inf"), 3, 10, float("inf")]
result_bucketizer = Bucketizer(splits=splits, inputCol="age", outputCol="result"
                                .transform(df)
result_bucketizer.show()
```

```
+---+----+
| id| age|
+---+----+
| 0|18.0|
| 1|19.0|
| 2| 8.0|
| 3| 5.0|
| 4| 2.0|
+---+----+
```

None

```
+---+----+-----+
| id| age|result|
+---+----+-----+
| 0|18.0|    2.0|
| 1|19.0|    2.0|
| 2| 8.0|    1.0|
| 3| 5.0|    1.0|
| 4| 2.0|    0.0|
+---+----+-----+
```

8.2.6 QuantileDiscretizer

QuantileDiscretizer는 연속형 특징 열을 가져와 구간화한 범주형 특징 열을 출력합니다. 구간의 수는 numBuckets 파라미터로 설정됩니다. 예를 들어, 뚜렷이 구분되는 분위수들을 생성하기에 입력의 고유 값들이 너무 적으면 사용되는 버킷 수가 구간의 수보다 적을 수 있습니다.

```
from pyspark.ml.feature import QuantileDiscretizer, Bucketizer

data = [(0, 18.0), (1, 19.0), (2, 8.0), (3, 5.0), (4, 2.0)]
df = spark.createDataFrame(data, ["id", "age"])
print(df.show())

qds = QuantileDiscretizer(numBuckets=5, inputCol="age", outputCol="buckets",
                           relativeError=0.01, handleInvalid="error")
bucketizer = qds.fit(df)
bucketizer.transform(df).show()
bucketizer.setHandleInvalid("skip").transform(df).show()
```

```
+---+----+
| id| age|
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
+---+----+
| 0 | 18.0 |
| 1 | 19.0 |
| 2 | 8.0 |
| 3 | 5.0 |
| 4 | 2.0 |
+---+----+
```

None

```
+---+----+-----+
| id | age | buckets |
+---+----+-----+
| 0 | 18.0 | 3.0 |
| 1 | 19.0 | 3.0 |
| 2 | 8.0 | 2.0 |
| 3 | 5.0 | 2.0 |
| 4 | 2.0 | 1.0 |
+---+----+-----+
```

```
+---+----+-----+
| id | age | buckets |
+---+----+-----+
| 0 | 18.0 | 3.0 |
| 1 | 19.0 | 3.0 |
| 2 | 8.0 | 2.0 |
| 3 | 5.0 | 2.0 |
| 4 | 2.0 | 1.0 |
+---+----+-----+
```

데이터에 NULL 값이 있으면 다음과 같은 결과가 나타납니다:

```
from pyspark.ml.feature import QuantileDiscretizer, Bucketizer

data = [(0, 18.0), (1, 19.0), (2, 8.0), (3, 5.0), (4, None)]
df = spark.createDataFrame(data, ["id", "age"])
print(df.show())

splits = [-float("inf"), 3, 10, float("inf")]
result_bucketizer = Bucketizer(splits=splits,
                                inputCol="age", outputCol="result").
    transform(df)
result_bucketizer.show()

qds = QuantileDiscretizer(numBuckets=5, inputCol="age", outputCol="buckets",
                           relativeError=0.01, handleInvalid="error")
bucketizer = qds.fit(df)
bucketizer.transform(df).show()
bucketizer.setHandleInvalid("skip").transform(df).show()
```

```
+---+----+
| id | age |
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
+---+----+
| 0 | 18.0 |
| 1 | 19.0 |
| 2 | 8.0 |
| 3 | 5.0 |
| 4 | null |
+---+----+
```

None

```
+---+----+-----+
| id | age | result |
+---+----+-----+
| 0 | 18.0 | 2.0 |
| 1 | 19.0 | 2.0 |
| 2 | 8.0 | 1.0 |
| 3 | 5.0 | 1.0 |
| 4 | null | null |
+---+----+-----+
```

```
+---+----+-----+
| id | age | buckets |
+---+----+-----+
| 0 | 18.0 | 3.0 |
| 1 | 19.0 | 4.0 |
| 2 | 8.0 | 2.0 |
| 3 | 5.0 | 1.0 |
| 4 | null | null |
+---+----+-----+
```

```
+---+----+-----+
| id | age | buckets |
+---+----+-----+
| 0 | 18.0 | 3.0 |
| 1 | 19.0 | 4.0 |
| 2 | 8.0 | 2.0 |
| 3 | 5.0 | 1.0 |
+---+----+-----+
```

8.2.7 StringIndexer

```
from pyspark.ml.feature import StringIndexer

df = spark.createDataFrame(
    [(0, "a"), (1, "b"), (2, "c"), (3, "a"), (4, "a"), (5, "c")],
    ["id", "category"])

indexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
indexed = indexer.fit(df).transform(df)
indexed.show()
```

id	category	categoryIndex
0	a	0.0
1	b	2.0
2	c	1.0
3	a	0.0
4	a	0.0
5	c	1.0

8.2.8 라벨 변환기 (labelConverter)

```
from pyspark.ml.feature import IndexToString, StringIndexer

df = spark.createDataFrame([
    (0, "Yes"), (1, "Yes"), (2, "Yes"), (3, "No"), (4, "No"), (5, "No")],
    ["id", "label"])

indexer = StringIndexer(inputCol="label", outputCol="labelIndex")
model = indexer.fit(df)
indexed = model.transform(df)

print("Transformed string column '%s' to indexed column '%s'" %
      (indexer.getInputCol(), indexer.getOutputCol()))
indexed.show()

print("StringIndexer will store labels in output column metadata\n")

converter = IndexToString(inputCol="labelIndex", outputCol="originalLabel")
converted = converter.transform(indexed)

print("Transformed indexed column '%s' back to original string column '%s' using "
      "labels in metadata" % (converter.getInputCol(), converter.
      getOutputCol()))
converted.select("id", "labelIndex", "originalLabel").show()
```

Transformed string column 'label' to indexed column 'labelIndex'		
id	label	labelIndex
0	Yes	1.0
1	Yes	1.0
2	Yes	1.0
3	No	0.0
4	No	0.0
5	No	0.0

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
StringIndexer will store labels in output column metadata
```

```
Transformed indexed column 'labelIndex' back to original string column
```

```
↳ 'originalLabel' using labels in metadata
```

id	labelIndex	originalLabel
0	1.0	Yes
1	1.0	Yes
2	1.0	Yes
3	0.0	No
4	0.0	No
5	0.0	No

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import IndexToString, StringIndexer

df = spark.createDataFrame([
    (0, "Yes"), (1, "Yes"), (2, "Yes"), (3, "No"), (4, "No"), (5, "No")],
    ["id", "label"])

indexer = StringIndexer(inputCol="label", outputCol="labelIndex")
converter = IndexToString(inputCol="labelIndex", outputCol="originalLabel")

pipeline = Pipeline(stages=[indexer, converter])

model = pipeline.fit(df)
result = model.transform(df)

result.show()
```

id	label	labelIndex	originalLabel
0	Yes	1.0	Yes
1	Yes	1.0	Yes
2	Yes	1.0	Yes
3	No	0.0	No
4	No	0.0	No
5	No	0.0	No

8.2.9 VectorIndexer

```

from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

from pyspark.ml.feature import RFormula

df = spark.createDataFrame([
    (0, 2.2, True, "1", "foo", 'CA'),
    (1, 3.3, False, "2", "bar", 'US'),
    (0, 4.4, False, "3", "baz", 'CHN'),
    (1, 5.5, False, "4", "foo", 'AUS')
], ['label', "real", "bool", "stringNum", "string", "country"])

formula = RFormula(
    formula="label ~ real + bool + stringNum + string + country",
    featuresCol="features",
    labelCol="label")

# 범주형 피처를 자동으로 식별하고 인덱싱합니다
# maxCategory를 지정하여 4보다 큰 고유 값들을 가지는 피처들은 연속형으로 취급합니다

featureIndexer = VectorIndexer(inputCol="features", \
                                outputCol="indexedFeatures", \
                                maxCategories=2)

pipeline = Pipeline(stages=[formula, featureIndexer])

model = pipeline.fit(df)
result = model.transform(df)

result.show()

```

```

+----+----+----+----+----+----+
|label|real| bool|stringNum|string|country|          features|      |
+----+----+----+----+----+----+
|   0| 2.2| true|        1|    foo|      CA| (10, [0,1,5,7],[2....|(10,[0,1,5,7],
|   [2....|
|   1| 3.3|false|        2|    bar|      US| (10, [0,3,8],[3.3,...|(10,[0,3,8],
|   [3.3,...|
|   0| 4.4|false|        3|    baz|      CHN| (10, [0,4,6,9],[4....|(10,[0,4,6,9],
|   [4....|
|   1| 5.5|false|        4|    foo|      AUS| (10, [0,2,5],[5.5,...|(10,[0,2,5],
|   [5.5,...|
+----+----+----+----+----+----+
|   2| 6.6|false|        5|    bar|      CHN| (10, [0,6,8,9],[6....|(10,[0,6,8,9],
|   [6....|
|   3| 7.7|false|        6|    baz|      AUS| (10, [0,7,9,0],[7....|(10,[0,7,9,0],
|   [7....|
|   4| 8.8|false|        7|    foo|      CHN| (10, [0,8,0,9],[8....|(10,[0,8,0,9],
|   [8....|
|   5| 9.9|false|        8|    bar|      AUS| (10, [0,9,0,8],[9....|(10,[0,9,0,8],
|   [9....|
|   6|10.0|false|        9|    baz|      CHN| (10, [0,10,0,9],[10....|(10,[0,10,0,9],
|   [10....|
|   7|11.1|false|        10|    foo|      AUS| (10, [0,11,0,8],[11....|(10,[0,11,0,8],
|   [11....|
|   8|12.2|false|        11|    bar|      CHN| (10, [0,12,0,7],[12....|(10,[0,12,0,7],
|   [12....|
|   9|13.3|false|        12|    baz|      AUS| (10, [0,13,0,6],[13....|(10,[0,13,0,6],
|   [13....|
|   10|14.4|false|        13|    foo|      CHN| (10, [0,14,0,5],[14....|(10,[0,14,0,5],
|   [14....|
|   11|15.5|false|        14|    bar|      AUS| (10, [0,15,0,4],[15....|(10,[0,15,0,4],
|   [15....|
|   12|16.6|false|        15|    baz|      CHN| (10, [0,16,0,3],[16....|(10,[0,16,0,3],
|   [16....|
|   13|17.7|false|        16|    foo|      AUS| (10, [0,17,0,2],[17....|(10,[0,17,0,2],
|   [17....|
|   14|18.8|false|        17|    bar|      CHN| (10, [0,18,0,1],[18....|(10,[0,18,0,1],
|   [18....|
|   15|19.9|false|        18|    baz|      AUS| (10, [0,19,0,0],[19....|(10,[0,19,0,0],
|   [19....|
|   16|20.0|false|        19|    foo|      CHN| (10, [0,20,0,9],[20....|(10,[0,20,0,9],
|   [20....|
|   17|21.1|false|        20|    bar|      AUS| (10, [0,21,0,8],[21....|(10,[0,21,0,8],
|   [21....|
|   18|22.2|false|        21|    baz|      CHN| (10, [0,22,0,7],[22....|(10,[0,22,0,7],
|   [22....|
|   19|23.3|false|        22|    foo|      AUS| (10, [0,23,0,6],[23....|(10,[0,23,0,6],
|   [23....|
|   20|24.4|false|        23|    bar|      CHN| (10, [0,24,0,5],[24....|(10,[0,24,0,5],
|   [24....|
|   21|25.5|false|        24|    baz|      AUS| (10, [0,25,0,4],[25....|(10,[0,25,0,4],
|   [25....|
|   22|26.6|false|        25|    foo|      CHN| (10, [0,26,0,3],[26....|(10,[0,26,0,3],
|   [26....|
|   23|27.7|false|        26|    bar|      AUS| (10, [0,27,0,2],[27....|(10,[0,27,0,2],
|   [27....|
|   24|28.8|false|        27|    baz|      CHN| (10, [0,28,0,1],[28....|(10,[0,28,0,1],
|   [28....|
|   25|29.9|false|        28|    foo|      AUS| (10, [0,29,0,0],[29....|(10,[0,29,0,0],
|   [29....|
|   26|30.0|false|        29|    bar|      CHN| (10, [0,30,0,9],[30....|(10,[0,30,0,9],
|   [30....|
|   27|31.1|false|        30|    baz|      AUS| (10, [0,31,0,8],[31....|(10,[0,31,0,8],
|   [31....|
|   28|32.2|false|        31|    foo|      CHN| (10, [0,32,0,7],[32....|(10,[0,32,0,7],
|   [32....|
|   29|33.3|false|        32|    bar|      AUS| (10, [0,33,0,6],[33....|(10,[0,33,0,6],
|   [33....|
|   30|34.4|false|        33|    baz|      CHN| (10, [0,34,0,5],[34....|(10,[0,34,0,5],
|   [34....|
|   31|35.5|false|        34|    foo|      AUS| (10, [0,35,0,4],[35....|(10,[0,35,0,4],
|   [35....|
|   32|36.6|false|        35|    bar|      CHN| (10, [0,36,0,3],[36....|(10,[0,36,0,3],
|   [36....|
|   33|37.7|false|        36|    baz|      AUS| (10, [0,37,0,2],[37....|(10,[0,37,0,2],
|   [37....|
|   34|38.8|false|        37|    foo|      CHN| (10, [0,38,0,1],[38....|(10,[0,38,0,1],
|   [38....|
|   35|39.9|false|        38|    bar|      AUS| (10, [0,39,0,0],[39....|(10,[0,39,0,0],
|   [39....|
|   36|40.0|false|        39|    baz|      CHN| (10, [0,40,0,9],[40....|(10,[0,40,0,9],
|   [40....|
|   37|41.1|false|        40|    foo|      AUS| (10, [0,41,0,8],[41....|(10,[0,41,0,8],
|   [41....|
|   38|42.2|false|        41|    bar|      CHN| (10, [0,42,0,7],[42....|(10,[0,42,0,7],
|   [42....|
|   39|43.3|false|        42|    baz|      AUS| (10, [0,43,0,6],[43....|(10,[0,43,0,6],
|   [43....|
|   40|44.4|false|        43|    foo|      CHN| (10, [0,44,0,5],[44....|(10,[0,44,0,5],
|   [44....|
|   41|45.5|false|        44|    bar|      AUS| (10, [0,45,0,4],[45....|(10,[0,45,0,4],
|   [45....|
|   42|46.6|false|        45|    baz|      CHN| (10, [0,46,0,3],[46....|(10,[0,46,0,3],
|   [46....|
|   43|47.7|false|        46|    foo|      AUS| (10, [0,47,0,2],[47....|(10,[0,47,0,2],
|   [47....|
|   44|48.8|false|        47|    bar|      CHN| (10, [0,48,0,1],[48....|(10,[0,48,0,1],
|   [48....|
|   45|49.9|false|        48|    baz|      AUS| (10, [0,49,0,0],[49....|(10,[0,49,0,0],
|   [49....|
|   46|50.0|false|        49|    foo|      CHN| (10, [0,50,0,9],[50....|(10,[0,50,0,9],
|   [50....|
|   47|51.1|false|        50|    bar|      AUS| (10, [0,51,0,8],[51....|(10,[0,51,0,8],
|   [51....|
|   48|52.2|false|        51|    baz|      CHN| (10, [0,52,0,7],[52....|(10,[0,52,0,7],
|   [52....|
|   49|53.3|false|        52|    foo|      AUS| (10, [0,53,0,6],[53....|(10,[0,53,0,6],
|   [53....|
|   50|54.4|false|        53|    bar|      CHN| (10, [0,54,0,5],[54....|(10,[0,54,0,5],
|   [54....|
|   51|55.5|false|        54|    baz|      AUS| (10, [0,55,0,4],[55....|(10,[0,55,0,4],
|   [55....|
|   52|56.6|false|        55|    foo|      CHN| (10, [0,56,0,3],[56....|(10,[0,56,0,3],
|   [56....|
|   53|57.7|false|        56|    bar|      AUS| (10, [0,57,0,2],[57....|(10,[0,57,0,2],
|   [57....|
|   54|58.8|false|        57|    baz|      CHN| (10, [0,58,0,1],[58....|(10,[0,58,0,1],
|   [58....|
|   55|59.9|false|        58|    foo|      AUS| (10, [0,59,0,0],[59....|(10,[0,59,0,0],
|   [59....|
|   56|60.0|false|        59|    bar|      CHN| (10, [0,60,0,9],[60....|(10,[0,60,0,9],
|   [60....|
|   57|61.1|false|        60|    baz|      AUS| (10, [0,61,0,8],[61....|(10,[0,61,0,8],
|   [61....|
|   58|62.2|false|        61|    foo|      CHN| (10, [0,62,0,7],[62....|(10,[0,62,0,7],
|   [62....|
|   59|63.3|false|        62|    bar|      AUS| (10, [0,63,0,6],[63....|(10,[0,63,0,6],
|   [63....|
|   60|64.4|false|        63|    baz|      CHN| (10, [0,64,0,5],[64....|(10,[0,64,0,5],
|   [64....|
|   61|65.5|false|        64|    foo|      AUS| (10, [0,65,0,4],[65....|(10,[0,65,0,4],
|   [65....|
|   62|66.6|false|        65|    bar|      CHN| (10, [0,66,0,3],[66....|(10,[0,66,0,3],
|   [66....|
|   63|67.7|false|        66|    baz|      AUS| (10, [0,67,0,2],[67....|(10,[0,67,0,2],
|   [67....|
|   64|68.8|false|        67|    foo|      CHN| (10, [0,68,0,1],[68....|(10,[0,68,0,1],
|   [68....|
|   65|69.9|false|        68|    bar|      AUS| (10, [0,69,0,0],[69....|(10,[0,69,0,0],
|   [69....|
|   66|70.0|false|        69|    baz|      CHN| (10, [0,70,0,9],[70....|(10,[0,70,0,9],
|   [70....|
|   67|71.1|false|        70|    foo|      AUS| (10, [0,71,0,8],[71....|(10,[0,71,0,8],
|   [71....|
|   68|72.2|false|        71|    bar|      CHN| (10, [0,72,0,7],[72....|(10,[0,72,0,7],
|   [72....|
|   69|73.3|false|        72|    baz|      AUS| (10, [0,73,0,6],[73....|(10,[0,73,0,6],
|   [73....|
|   70|74.4|false|        73|    foo|      CHN| (10, [0,74,0,5],[74....|(10,[0,74,0,5],
|   [74....|
|   71|75.5|false|        74|    bar|      AUS| (10, [0,75,0,4],[75....|(10,[0,75,0,4],
|   [75....|
|   72|76.6|false|        75|    baz|      CHN| (10, [0,76,0,3],[76....|(10,[0,76,0,3],
|   [76....|
|   73|77.7|false|        76|    foo|      AUS| (10, [0,77,0,2],[77....|(10,[0,77,0,2],
|   [77....|
|   74|78.8|false|        77|    bar|      CHN| (10, [0,78,0,1],[78....|(10,[0,78,0,1],
|   [78....|
|   75|79.9|false|        78|    baz|      AUS| (10, [0,79,0,0],[79....|(10,[0,79,0,0],
|   [79....|
|   76|80.0|false|        79|    foo|      CHN| (10, [0,80,0,9],[80....|(10,[0,80,0,9],
|   [80....|
|   77|81.1|false|        80|    bar|      AUS| (10, [0,81,0,8],[81....|(10,[0,81,0,8],
|   [81....|
|   78|82.2|false|        81|    baz|      CHN| (10, [0,82,0,7],[82....|(10,[0,82,0,7],
|   [82....|
|   79|83.3|false|        82|    foo|      AUS| (10, [0,83,0,6],[83....|(10,[0,83,0,6],
|   [83....|
|   80|84.4|false|        83|    bar|      CHN| (10, [0,84,0,5],[84....|(10,[0,84,0,5],
|   [84....|
|   81|85.5|false|        84|    baz|      AUS| (10, [0,85,0,4],[85....|(10,[0,85,0,4],
|   [85....|
|   82|86.6|false|        85|    foo|      CHN| (10, [0,86,0,3],[86....|(10,[0,86,0,3],
|   [86....|
|   83|87.7|false|        86|    bar|      AUS| (10, [0,87,0,2],[87....|(10,[0,87,0,2],
|   [87....|
|   84|88.8|false|        87|    baz|      CHN| (10, [0,88,0,1],[88....|(10,[0,88,0,1],
|   [88....|
|   85|89.9|false|        88|    foo|      AUS| (10, [0,89,0,0],[89....|(10,[0,89,0,0],
|   [89....|
|   86|90.0|false|        89|    bar|      CHN| (10, [0,90,0,9],[90....|(10,[0,90,0,9],
|   [90....|
|   87|91.1|false|        90|    baz|      AUS| (10, [0,91,0,8],[91....|(10,[0,91,0,8],
|   [91....|
|   88|92.2|false|        91|    foo|      CHN| (10, [0,92,0,7],[92....|(10,[0,92,0,7],
|   [92....|
|   89|93.3|false|        92|    bar|      AUS| (10, [0,93,0,6],[93....|(10,[0,93,0,6],
|   [93....|
|   90|94.4|false|        93|    baz|      CHN| (10, [0,94,0,5],[94....|(10,[0,94,0,5],
|   [94....|
|   91|95.5|false|        94|    foo|      AUS| (10, [0,95,0,4],[95....|(10,[0,95,0,4],
|   [95....|
|   92|96.6|false|        95|    bar|      CHN| (10, [0,96,0,3],[96....|(10,[0,96,0,3],
|   [96....|
|   93|97.7|false|        96|    baz|      AUS| (10, [0,97,0,2],[97....|(10,[0,97,0,2],
|   [97....|
|   94|98.8|false|        97|    foo|      CHN| (10, [0,98,0,1],[98....|(10,[0,98,0,1],
|   [98....|
|   95|99.9|false|        98|    bar|      AUS| (10, [0,99,0,0],[99....|(10,[0,99,0,0],
|   [99....|
|   96|100.0|false|        99|    baz|      CHN| (10, [0,100,0,9],[100....|(10,[0,100,0,9],
|   [100....|
|   97|101.1|false|        100|    foo|      AUS| (10, [0,101,0,8],[101....|(10,[0,101,0,8],
|   [101....|
|   98|102.2|false|        101|    bar|      CHN| (10, [0,102,0,7],[102....|(10,[0,102,0,7],
|   [102....|
|   99|103.3|false|        102|    baz|      AUS| (10, [0,103,0,6],[103....|(10,[0,103,0,6],
|   [103....|
|   100|104.4|false|        103|    foo|      CHN| (10, [0,104,0,5],[104....|(10,[0,104,0,5],
|   [104....|
|   101|105.5|false|        104|    bar|      AUS| (10, [0,105,0,4],[105....|(10,[0,105,0,4],
|   [105....|
|   102|106.6|false|        105|    baz|      CHN| (10, [0,106,0,3],[106....|(10,[0,106,0,3],
|   [106....|
|   103|107.7|false|        106|    foo|      AUS| (10, [0,107,0,2],[107....|(10,[0,107,0,2],
|   [107....|
|   104|108.8|false|        107|    bar|      CHN| (10, [0,108,0,1],[108....|(10,[0,108,0,1],
|   [108....|
|   105|109.9|false|        108|    baz|      AUS| (10, [0,109,0,0],[109....|(10,[0,109,0,0],
|   [109....|
|   106|110.0|false|        109|    foo|      CHN| (10, [0,110,0,9],[110....|(10,[0,110,0,9],
|   [110....|
|   107|111.1|false|        110|    bar|      AUS| (10, [0,111,0,8],[111....|(10,[0,111,0,8],
|   [111....|
|   108|112.2|false|        111|    baz|      CHN| (10, [0,112,0,7],[112....|(10,[0,112,0,7],
|   [112....|
|   109|113.3|false|        112|    foo|      AUS| (10, [0,113,0,6],[113....|(10,[0,113,0,6],
|   [113....|
|   110|114.4|false|        113|    bar|      CHN| (10, [0,114,0,5],[114....|(10,[0,114,0,5],
|   [114....|
|   111|115.5|false|        114|    baz|      AUS| (10, [0,115,0,4],[115....|(10,[0,115,0,4],
|   [115....|
|   112|116.6|false|        115|    foo|      CHN| (10, [0,116,0,3],[116....|(10,[0,116,0,3],
|   [116....|
|   113|117.7|false|        116|    bar|      AUS| (10, [0,117,0,2],[117....|(10,[0,117,0,2],
|   [117....|
|   114|118.8|false|        117|    baz|      CHN| (10, [0,118,0,1],[118....|(10,[0,118,0,1],
|   [118....|
|   115|119.9|false|        118|    foo|      AUS| (10, [0,119,0,0],[119....|(10,[0,119,0,0],
|   [119....|
|   116|120.0|false|        119|    bar|      CHN| (10, [0,120,0,9],[120....|(10,[0,120,0,9],
|   [120....|
|   117|121.1|false|        120|    baz|      AUS| (10, [0,121,0,8],[121....|(10,[0,121,0,8],
|   [121....|
|   118|122.2|false|        121|    foo|      CHN| (10, [0,122,0,7],[122....|(10,[0,122,0,7],
|   [122....|
|   119|123.3|false|        122|    bar|      AUS| (10, [0,123,0,6],[123....|(10,[0,123,0,6],
|   [123....|
|   120|124.4|false|        123|    baz|      CHN| (10, [0,124,0,5],[124....|(10,[0,124,0,5],
|   [124....|
|   121|125.5|false|        124|    foo|      AUS| (10, [0,125,0,4],[125....|(10,[0,125,0,4],
|   [125....|
|   122|126.6|false|        125|    bar|      CHN| (10, [0,126,0,3],[126....|(10,[0,126,0,3],
|   [126....|
|   123|127.7|false|        126|    baz|      AUS| (10, [0,127,0,2],[127....|(10,[0,127,0,2],
|   [127....|
|   124|128.8|false|        127|    foo|      CHN| (10, [0,128,0,1],[128....|(10,[0,128,0,1],
|   [128....|
|   125|129.9|false|        128|    bar|      AUS| (10, [0,129,0,0],[129....|(10,[0,129,0,0],
|   [129....|
|   126|130.0|false|        129|    baz|      CHN| (10, [0,130,0,9],[130....|(10,[0,130,0,9],
|   [130....|
|   127|131.1|false|        130|    foo|      AUS| (10, [0,131,0,8],[131....|(10,[0,131,0,8],
|   [131....|
|   128|132.2|false|        131|    bar|      CHN| (10, [0,132,0,7],[132....|(10,[0,132,0,7],
|   [132....|
|   129|133.3|false|        132|    baz|      AUS| (10, [0,133,0,6],[133....|(10,[0,133,0,6],
|   [133....|
|   130|134.4|false|        133|    foo|      CHN| (10, [0,134,0,5],[134....|(10,[0,134,0,5],
|   [134....|
|   131|135.5|false|        134|    bar|      AUS| (10, [0,135,0,4],[135....|(10,[0,135,0,4],
|   [135....|
|   132|136.6|false|        135|    baz|      CHN| (10, [0,136,0,3],[136....|(10,[0,136,0,3],
|   [136....|
|   133|137.7|false|        136|    foo|      AUS| (10, [0,137,0,2],[137....|(10,[0,137,0,2],
|   [137....|
|   134|138.8|false|        137|    bar|      CHN| (10, [0,138,0,1],[138....|(10,[0,138,0,1],
|   [138....|
|   135|139.9|false|        138|    baz|      AUS| (10, [0,139,0,0],[139....|(10,[0,139,0,0],
|   [139....|
|   136|140.0|false|        139|    foo|      CHN| (10, [0,140,0,9],[140....|(10,[0,140,0,9],
|   [140....|
|   137|141.1|false|        140|    bar|      AUS| (10, [0,141,0,8],[141....|(10,[0,141,0,8],
|   [141....|
|   138|142.2|false|        141|    baz|      CHN| (10, [0,142,0,7],[142....|(10,[0,142,0,7],
|   [142....|
|   139|143.3|false|        142|    foo|      AUS| (10, [0,143,0,6],[143....|(10,[0,143,0,6],
|   [143....|
|   140|144.4|false|        143|    bar|      CHN| (10, [0,144,0,5],[144....|(10,[0,144,0,5],
|   [144....|
|   141|145.5|false|        144|    baz|      AUS| (10, [0,145,0,4],[145....|(10,[0,145,0,4],
|   [145....|
|   142|146.6|false|        145|    foo|      CHN| (10, [0,146,0,3],[146....|(10,[0,146,0,3],
|   [146....|
|   143|147.7|false|        146|    bar|      AUS| (10, [0,147,0,2],[147....|(10,[0,147,0,2],
|   [147....|
|   144|148.8|false|        147|    baz|      CHN| (10, [0,148,0,1],[148....|(10,[0,148,0,1],
|   [148....|
|   145|149.9|false|        148|    foo|      AUS| (10, [0,149,0,0],[149....|(10,[0,149,0,0],
|   [149....|
|   146|150.0|false|        149|    bar|      CHN| (10, [0,150,0,9],[150....|(10,[0,150,0,9],
|   [150....|
|   147|151.1|false|        150|    baz|      AUS| (10, [0,151,0,8],[151....|(10,[0,151,0,8],
|   [151....|
|   148|152.2|false|        151|    foo|      CHN| (10, [0,152,0,7],[152....|(10,[0,152,0,7],
|   [152....|
|   149|153.3|false|        152|    bar|      AUS| (10, [0,153,0,6],[153....|(10,[0,153,0,6],
|   [153....|
|   150|154.4|false|        153|    baz|      CHN| (10, [0,154,0,5],[154....|(10,[0,154,0,5],
|   [154....|
|   151|155.5|false|        154|    foo|      AUS| (10, [0,155,0,4],[155....|(10,[0,155,0,4],
|   [155....|
|   152|156.6|false|        155|    bar|      CHN| (10, [0,156,0,3],[156....|(10,[0,156,0,3],
|   [156....|
|   153|157.7|false|        156|    baz|      AUS| (10, [0,157,0,2],[157....|(10,[0,157,0,2],
|   [157....|
|   154|158.8|false|        157|    foo|      CHN| (10, [0,158,0,1],[158....|(10,[0,158,0,1],
|   [158....|
|   155|159.9|false|        158|    bar|      AUS| (10, [0,159,0,0],[159....|(10,[0,159,0,0],
|   [159....|
|   156|160.0|false|        159|    baz|      CHN| (10, [0,160,0,9],[160....|(10,[0,160,0,9],
|   [160....|
|   157|161.1|false|        160|    foo|      AUS| (10, [0,161,0,8],[161....|(10,[0,161,0,8],
|   [161....|
|   158|162.2|false|        161|    bar|      CHN| (10, [0,162,0,7],[162....|(10,[0,162,0,7],
|   [162....|
|   159|163.3|false|        162|    baz|      AUS| (10, [0,163,0,6],[163....|(10,[0,163,0,6],
|   [163....|
|   160|164.4|false|        163|    foo|      CHN| (10, [0,164,0,5],[164....|(10,[0,164,0,5],
|   [164....|
|   161|165.5|false|        164|    bar|      AUS| (10, [0,165,0,4],[165....|(10,[0,165,0,4],
|   [165....|
|   162|166.6|false|        165|    baz|      CHN| (10, [0,166,0,3],[166....|(10,[0,166,0,3],
|   [166....|
|   163|167.7|false|        166|    foo|      AUS| (10, [0,167,0,2],[167....|(10,[0,167,0,2],
|   [167....|
|   164|168.8|false|        167|    bar|      CHN| (10, [0,168,0,1],[168....|(10,[0,168,0,1],
|   [168....|
|   165|169.9|false|        168|    baz|      AUS| (10, [0,169,0,0],[169....|(10,[0,169,0,0],
|   [169....|
|   166|170.0|false|        169|    foo|      CHN| (10, [0,170,0,9],[170....
```

8.2.10 VectorAssembler

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler

dataset = spark.createDataFrame(
    [(0, 18, 1.0, Vectors.dense([0.0, 10.0, 0.5]), 1.0)],
    ["id", "hour", "mobile", "userFeatures", "clicked"])

assembler = VectorAssembler(
    inputCols=["hour", "mobile", "userFeatures"],
    outputCol="features")

output = assembler.transform(dataset)
print("Assembled columns 'hour', 'mobile', 'userFeatures' to vector column\n"
      "'features'")
output.select("features", "clicked").show(truncate=False)
```

```
Assembled columns 'hour', 'mobile', 'userFeatures' to vector column 'features'
+-----+-----+
|features          |clicked|
+-----+-----+
|[18.0,1.0,0.0,10.0,0.5]|1.0   |
+-----+-----+
```

8.2.11 OneHotEncoder

o) 노트는 OneHotEncoder를 설명하기 위해 독자분들을 위해 작성된 것입니다:

SparkSession 만들기 및 불러오기

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark create RDD example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

```
df = spark.createDataFrame([
    (0, "a"),
    (1, "b"),
    (2, "c"),
    (3, "a"),
    (4, "a"),
    (5, "c")
], ["id", "category"])
df.show()
```

```
+---+-----+
| id|category|
+---+-----+
| 0 |     a |
| 1 |     b |
| 2 |     c |
| 3 |     a |
| 4 |     a |
| 5 |     c |
+---+-----+
```

OneHotEncoder

Encoder

```
from pyspark.ml.feature import OneHotEncoder, StringIndexer

stringIndexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
model = stringIndexer.fit(df)
indexed = model.transform(df)

# 기본 설정: dropLast=True 마지막 카테고리를 버릴 것인지 설정하는 옵션
encoder = OneHotEncoder(inputCol="categoryIndex", outputCol="categoryVec",
                         dropLast=False)
encoded = encoder.fit(indexed)
encoded = encoder.transform(indexed)
encoded.show()
```

```
+---+-----+-----+-----+
| id|category|categoryIndex|  categoryVec|
+---+-----+-----+-----+
| 0 |     a |          0.0 | (3, [0], [1.0]) |
| 1 |     b |          2.0 | (3, [2], [1.0]) |
| 2 |     c |          1.0 | (3, [1], [1.0]) |
| 3 |     a |          0.0 | (3, [0], [1.0]) |
| 4 |     a |          0.0 | (3, [0], [1.0]) |
| 5 |     c |          1.0 | (3, [1], [1.0]) |
+---+-----+-----+-----+
```

노트: OneHotEncoder의 기본 설정은: dropLast=True입니다.

```
# 기본 설정: dropLast=True
encoder = OneHotEncoder(inputCol="categoryIndex", outputCol=
                         "categoryVec")
encoded = encoder.transform(indexed)
encoded.show()
```

<code>id</code>	<code>category</code>	<code>categoryIndex</code>	<code>categoryVec</code>
0	a	0.0	(2, [0], [1.0])
1	b	2.0	(2, [], [])
2	c	1.0	(2, [1], [1.0])
3	a	0.0	(2, [0], [1.0])
4	a	0.0	(2, [0], [1.0])
5	c	1.0	(2, [1], [1.0])

벡터 어셈블러 (Vector Assembler)

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler
categoricalCols = ['category']

indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
             for c in categoricalCols ]
# 기본 설정: dropLast=True
encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                            outputCol="{0}_encoded".format(indexer.getOutputCol())),
             dropLast=False]
             for indexer in indexers ]
assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in
                                       encoders],
                             outputCol="features")
pipeline = Pipeline(stages=indexers + encoders + [assembler])

model=pipeline.fit(df)
data = model.transform(df)
```

<code>id</code>	<code>category</code>	<code>category_indexed</code>	<code>category_indexed_encoded</code>	<code>features</code>
0	a	0.0	(3, [0], [1.0]) [1.0, 0.0, 0.0]	
1	b	2.0	(3, [2], [1.0]) [0.0, 0.0, 1.0]	
2	c	1.0	(3, [1], [1.0]) [0.0, 1.0, 0.0]	
3	a	0.0	(3, [0], [1.0]) [1.0, 0.0, 0.0]	
4	a	0.0	(3, [0], [1.0]) [1.0, 0.0, 0.0]	
5	c	1.0	(3, [1], [1.0]) [0.0, 1.0, 0.0]	

응용: 더미 변수 만들기

```
def get_dummy(df, indexCol, categoricalCols, continuousCols, labelCol,
             dropLast=False):
    """
    머신러닝 모델링을 위해 더미 변수를 생성하고 연속형 변수와 결합합니다
    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터의 이름 목록
    :param labelCol: 라벨 열의 이름
    :param dropLast: 마지막 열의 drop 여부
    :return: 특징 행렬
    :작성자: Wenqiang Feng
    :이메일: von198@gmail.com

>>> df = spark.createDataFrame([
        (0, "a"),
        (1, "b"),
        (2, "c"),
        (3, "a"),
        (4, "a"),
        (5, "c")
    ], ["id", "category"])

>>> indexCol = 'id'
>>> categoricalCols = ['category']
>>> continuousCols = []
>>> labelCol = []

>>> mat = get_dummy(df, indexCol, categoricalCols, continuousCols, labelCol)
>>> mat.show()

>>>
+---+-----+
| id| features|
+---+-----+
| 0|[1.0,0.0,0.0]|
| 1|[0.0,0.0,1.0]|
| 2|[0.0,1.0,0.0]|
| 3|[1.0,0.0,0.0]|
| 4|[1.0,0.0,0.0]|
| 5|[0.0,1.0,0.0]|
+---+-----+
    ...

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
VectorAssembler
from pyspark.sql.functions import col
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
             for c in categoricalCols ]

# 기본 설정: dropLast=True
encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                            outputCol="{0}_encoded".format(indexer.getOutputCol())),
             ↪dropLast=False)
             for indexer in indexers ]

assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder
                                         ↪in encoders]
                             + continuousCols, outputCol="features")

pipeline = Pipeline(stages=indexers + encoders + [assembler])

model=pipeline.fit(df)
data = model.transform(df)

if indexCol and labelCol:
    # 지도 학습에서 인덱스와 라벨 칼럼이 있는 경우
    data = data.withColumn('label', col(labelCol))
    return data.select(indexCol, 'features', 'label')
elif not indexCol and labelCol:
    # 지도 학습에서 라벨 칼럼만 있는 경우
    data = data.withColumn('label', col(labelCol))
    return data.select('features', 'label')
elif indexCol and not labelCol:
    # 비지도 학습에서 인덱스 칼럼만 있는 경우
    return data.select(indexCol, 'features')
elif not indexCol and not labelCol:
    # 비지도 학습에서 라벨과 인덱스 칼럼이 모두 없는 경우
    return data.select('features')

```

비지도 학습 시나리오

```

df = spark.createDataFrame([
    (0, "a"),
    (1, "b"),
    (2, "c"),
    (3, "a"),
    (4, "a"),
    (5, "c")
], ["id", "category"])
df.show()

indexCol = 'id'
categoricalCols = ['category']
continuousCols = []
labelCol = []

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
mat = get_dummy(df, indexCol, categoricalCols, continuousCols, labelCol)
```

```
mat.show()
```

```
+---+-----+
| id |      features |
+---+-----+
| 0 | [1.0,0.0,0.0] |
| 1 | [0.0,0.0,1.0] |
| 2 | [0.0,1.0,0.0] |
| 3 | [1.0,0.0,0.0] |
| 4 | [1.0,0.0,0.0] |
| 5 | [0.0,1.0,0.0] |
+---+-----+
```

지도 학습 시나리오

```
df = spark.read.csv(path='bank.csv',
                     sep=',', encoding='UTF-8', comment=None,
                     header=True, inferSchema=True)

indexCol = []
catCols = ['job', 'marital', 'education', 'default',
           'housing', 'loan', 'contact', 'poutcome']

contCols = ['balance', 'duration', 'campaign', 'pdays', 'previous']
labelCol = 'y'

data = get_dummy(df, indexCol, catCols, contCols, labelCol, dropLast=False)
data.show(5)
```

```
+-----+-----+
|      features | label |
+-----+-----+
| (37, [8,12,17,19,2... |  no |
| (37, [4,12,15,19,2... |  no |
| (37, [0,13,16,19,2... |  no |
| (37, [0,12,16,19,2... |  no |
| (37, [1,12,15,19,2... |  no |
+-----+-----+
only showing top 5 rows
```

주피터 노트북은 Colab에서 찾을 수 있습니다: OneHotEncoder .

8.2.12 스케일러(Scaler)

```
from pyspark.ml.feature import Normalizer, StandardScaler, MinMaxScaler,_
→MaxAbsScaler

scaler_type = 'Normal'
if scaler_type=='Normal':
    scaler = Normalizer(inputCol="features", outputCol="scaledFeatures", p=1.
→0)
elif scaler_type=='Standard':
    scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures",
                           withStd=True, withMean=False)
elif scaler_type=='MinMaxScaler':
    scaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")
elif scaler_type=='MaxAbsScaler':
    scaler = MaxAbsScaler(inputCol="features", outputCol="scaledFeatures")
```

```
from pyspark.ml import Pipeline
from pyspark.ml.linalg import Vectors

df = spark.createDataFrame([
    (0, Vectors.dense([1.0, 0.5, -1.0])),
    (1, Vectors.dense([2.0, 1.0, 1.0])),
    (2, Vectors.dense([4.0, 10.0, 2.0]))
], ["id", "features"])
df.show()

pipeline = Pipeline(stages=[scaler])

model = pipeline.fit(df)
data = model.transform(df)
data.show()
```

id	features
0	[1.0, 0.5, -1.0]
1	[2.0, 1.0, 1.0]
2	[4.0, 10.0, 2.0]

id	features	scaledFeatures
0	[1.0, 0.5, -1.0]	[0.4, 0.2, -0.4]
1	[2.0, 1.0, 1.0]	[0.5, 0.25, 0.25]
2	[4.0, 10.0, 2.0]	[0.25, 0.625, 0.125]

Normalizer

```

from pyspark.ml.feature import Normalizer
from pyspark.ml.linalg import Vectors

dataFrame = spark.createDataFrame([
    (0, Vectors.dense([1.0, 0.5, -1.0])),,
    (1, Vectors.dense([2.0, 1.0, 1.0])),,
    (2, Vectors.dense([4.0, 10.0, 2.0])),)
], ["id", "features"])

# $L^1$ 놈(norm)을 이용한 각 특징 벡터의 정규화
normalizer = Normalizer(inputCol="features", outputCol="normFeatures", p=1.0)
l1NormData = normalizer.transform(dataFrame)
print("L^1 놈을 이용한 특징 벡터 정규화")
l1NormData.show()

# $L^\infty$ 놈(norm) 이용한 각 특징 벡터의 정규화
lInfNormData = normalizer.transform(dataFrame, {normalizer.p: float("inf")})
print("L^inf 놈을 이용한 특징 벡터 정규화")
lInfNormData.show()

```

L¹ 놈을 이용한 특징 벡터 정규화

id	features	normFeatures
0	[1.0, 0.5, -1.0]	[0.4, 0.2, -0.4]
1	[2.0, 1.0, 1.0]	[0.5, 0.25, 0.25]
2	[4.0, 10.0, 2.0]	[0.25, 0.625, 0.125]

L^{inf} 놈을 이용한 특징 벡터 정규화

id	features	normFeatures
0	[1.0, 0.5, -1.0]	[1.0, 0.5, -1.0]
1	[2.0, 1.0, 1.0]	[1.0, 0.5, 0.5]
2	[4.0, 10.0, 2.0]	[0.4, 1.0, 0.2]

StandardScaler

```

from pyspark.ml.feature import Normalizer, StandardScaler, MinMaxScaler, ↴
    MaxAbsScaler

from pyspark.ml.linalg import Vectors

dataFrame = spark.createDataFrame([
    (0, Vectors.dense([1.0, 0.5, -1.0])),,
    (1, Vectors.dense([2.0, 1.0, 1.0])),,

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
(2, Vectors.dense([4.0, 10.0, 2.0])),)
], ["id", "features"])

scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures",
                        withStd=True, withMean=False)
scaler.fit((dataFrame)).transform(dataFrame)
scaler.show(truncate=False)
```

<code> id features</code>	<code> scaledFeatures</code>
<code> 0 [1.0, 0.5, -1.0]</code>	<code> [0.6546536707079772, 0.09352195295828244, -0.</code>
<code> 1 [2.0, 1.0, 1.0]</code>	<code> [1.3093073414159544, 0.1870439059165649, 0.</code>
<code> 2 [4.0, 10.0, 2.0]</code>	<code> [2.618614682831909, 1.870439059165649, 1.3093073414159542]</code>

MinMaxScaler

```
from pyspark.ml.feature import Normalizer, StandardScaler, MinMaxScaler,
                                MaxAbsScaler

from pyspark.ml.linalg import Vectors

dataFrame = spark.createDataFrame([
    (0, Vectors.dense([1.0, 0.5, -1.0])),
    (1, Vectors.dense([2.0, 1.0, 1.0])),
    (2, Vectors.dense([4.0, 10.0, 2.0]))
], ["id", "features"])

scaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")
scaledData = scaler.fit((dataFrame)).transform(dataFrame)
scaledData.show(truncate=False)
```

<code> id features</code>	<code> scaledFeatures</code>
<code> 0 [1.0, 0.5, -1.0]</code>	<code> [0.0, 0.0, 0.0]</code>
<code> 1 [2.0, 1.0, 1.0]</code>	<code> [0.3333333333333333, 0.05263157894736842, 0.</code>
<code> 2 [4.0, 10.0, 2.0]</code>	<code> [1.0, 1.0, 1.0]</code>

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
| 2 | [4.0,10.0,2.0] | [1.0,1.0,1.0]
+--+
| 2 | [4.0,10.0,2.0] | [1.0,1.0,1.0]
```

MaxAbsScaler

```
from pyspark.ml.feature import Normalizer, StandardScaler, MinMaxScaler, MaxAbsScaler

from pyspark.ml.linalg import Vectors

dataFrame = spark.createDataFrame([
    (0, Vectors.dense([1.0, 0.5, -1.0])),,
    (1, Vectors.dense([2.0, 1.0, 1.0])),,
    (2, Vectors.dense([4.0, 10.0, 2.0])),)
], ["id", "features"])

scaler = MaxAbsScaler(inputCol="features", outputCol="scaledFeatures")
scaledData = scaler.fit((dataFrame)).transform(dataFrame)
scaledData.show(truncate=False)
```

<code>id</code>	<code>features</code>	<code>scaledFeatures</code>
0	[1.0, 0.5, -1.0]	[0.25, 0.05, -0.5]
1	[2.0, 1.0, 1.0]	[0.5, 0.1, 0.5]
2	[4.0, 10.0, 2.0]	[1.0, 1.0, 1.0]

8.2.13 주성분 분석 (PCA)

```
from pyspark.ml.feature import PCA
from pyspark.ml.linalg import Vectors

data = [(Vectors.sparse(5, [(1, 1.0), (3, 7.0)]),),
        (Vectors.dense([2.0, 0.0, 3.0, 4.0, 5.0]),),
        (Vectors.dense([4.0, 0.0, 0.0, 6.0, 7.0]),)]
df = spark.createDataFrame(data, ["features"])

pca = PCA(k=3, inputCol="features", outputCol="pcaFeatures")
model = pca.fit(df)

result = model.transform(df).select("pcaFeatures")
result.show(truncate=False)
```

```
+-----+
| pcaFeatures
+-----+
| [1.6485728230883807, -4.013282700516296, -5.524543751369388] |
| [-4.645104331781534, -1.1167972663619026, -5.524543751369387] |
| [-6.428880535676489, -5.337951427775355, -5.524543751369389] |
+-----+
```

8.2.14 DCT

```
from pyspark.ml.feature import DCT
from pyspark.ml.linalg import Vectors

df = spark.createDataFrame([
    (Vectors.dense([0.0, 1.0, -2.0, 3.0]),),
    (Vectors.dense([-1.0, 2.0, 4.0, -7.0]),),
    (Vectors.dense([14.0, -2.0, -5.0, 1.0])),], ["features"])

dct = DCT(inverse=False, inputCol="features", outputCol="featuresDCT")

dctDf = dct.transform(df)

dctDf.select("featuresDCT").show(truncate=False)
```

```
+-----+
| featuresDCT
+-----+
| [1.0, -1.1480502970952693, 2.0000000000000004, -2.7716385975338604] |
| [-1.0, 3.378492794482933, -7.000000000000001, 2.9301512653149677] |
| [4.0, 9.304453421915744, 11.000000000000002, 1.5579302036357163] |
+-----+
```

8.3 특징 선택

8.3.1 LASSO

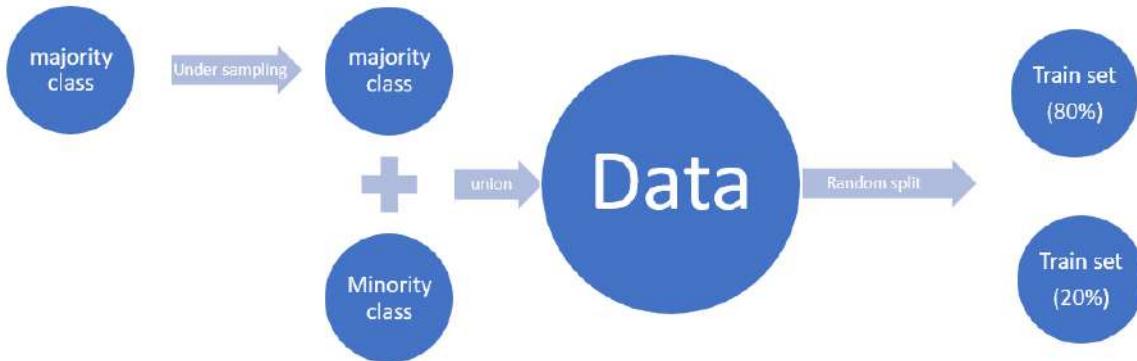
변수 선택 및 상관 변수 제거. Ridge 방법은 상관성이 있는 특징 변수의 계수를 축소하는 반면 LASSO 방법은 한 특징 변수를 선택하면 다른 특징 변수를 무시합니다. elastic net 패널티는 이 두 가지를 혼합한 것입니다. 그룹들 안에 속한 특징 변수들이 서로 상관되어 있으면 $\alpha = 0.5$ 은 특징 변수들을 그룹 내부 혹은 외부로 선택하는 경향이 있습니다. α 가 1에 가까우면 elastic net은 LASSO 방법과 매우 유사한 성능을 발휘하며, 특징들 간 극단적인 상관관계로 인한 퇴보 및 거친 움직임 모두 제거 합니다.

8.3.2 랜덤 포레스트(RandomForest)

AutoFeatures는 PySpark 자동 특징 선택기(Auto Feature Selector) 라이브러리 저장소입니다. AutoFeatures의 자세한 내용은 해당 저장소에서 확인하실 수 있습니다.

8.4 불균형 데이터: 언더샘플링

우리가 PySpark을 사용하여 빅데이터를 처리하므로 불균형 분류를 위한 언더샘플링은 불균형 데이터를 처리하는 데 유용한 방법입니다. 언더샘플링은 클래스 분포의 편향을 줄이기 위해 불균형 데이터 셋에 널리 사용되는 기술이다. 그러나 한 클래스를 언더샘플링하는 것은 훈련 셋의 사전 확률을 수정하고 결과적으로 분류기의 사후 확률에 편향을 갖게 한다는 것이 잘 알려져 있습니다. 언더샘플링을 적용한 후에는 확률을 다시 보정해야 합니다. 불균형 분류를 위해 언더샘플링을 사용해 확률을 보정하기.



```

df = spark.createDataFrame([
    (0, "Yes"),
    (1, "Yes"),
    (2, "Yes"),
    (3, "Yes"),
    (4, "No"),
    (5, "No")
], ["id", "label"])
df.show()
  
```

```

+---+-----+
| id | label |
+---+-----+
| 0 | Yes |
| 1 | Yes |
| 2 | Yes |
| 3 | Yes |
| 4 | No |
| 5 | No |
+---+-----+
  
```

8.4.1 언더샘플링 비율 계산하기

```
import math
def round_up(n, decimals=0):
    multiplier = 10 ** decimals
    return math.ceil(n * multiplier) / multiplier

# 결측값을 갖는 행 버리기
df = df.dropna()
# 언더-샘플링의 다수 집합
label_Y = df.filter(df.label=='Yes')
label_N = df.filter(df.label=='No')
sampleRatio = round_up(label_N.count() / df.count(), 2)
```

8.4.2 언더샘플링(Undersampling)

```
label_Y_sample = label_Y.sample(False, sampleRatio)
# 소수 집합과 언더-샘플링한 다수 집합을 합치기
data = label_N.unionAll(label_Y_sample)
data.show()
```

```
+---+-----+
| id | label |
+---+-----+
| 4 | No |
| 5 | No |
| 1 | Yes |
| 2 | Yes |
+---+-----+
```

8.4.3 확률 보정하기

언더샘플링은 클래스 분포들의 왜곡을 줄이기 위해 불균형 데이터 셋에 널리 사용되는 기법입니다. 하지만 한 클래스를 언더 샘플링하는 것은 훈련 셋의 사전 확률을 수정하고 결론적으로 분류기의 사후 확률에 편향을 갖게 하는 것은 잘 알려진 사실입니다. 불균형 분류를 위한 언더 샘플링의 확률 보정하기.

```
predication.withColumn('adj_probability', sampleRatio*F.col('probability')/
    ((sampleRatio-1)*F.col('probability')+1))
```

회귀분석

중국 속담

천 리의 여정은 한 걸음으로 시작된다 – 중국의 옛 속담

통계 모델링에서 회귀 분석은 종속 변수와 하나 이상의 독립 변수 사이의 관계를 조사하는 데 중점을 둡니다. 위키피디아 회귀 분석.

데이터 마이닝에서 회귀(regression)는 하나 이상의 특징(또는 범주 또는 수치형일 수 있는 예측변수)과 라벨 값 사이의 관계를 나타내는 모델입니다.

9.1 선형 회귀

9.1.1 도입

n 개의 특징(변수들)과 m 개의 샘플(데이터 점들)을 가지는 데이터 집합 $\{x_{i1}, \dots, x_{in}, y_i\}_{i=1}^m$ 이 주어졌을 때, j 개의 독립변수를 가지는 m 개의 데이터 점들을 모델링하기 위한 단순회귀모델에서는 다음과 같은 수식을 가집니다:

$$y_i = \beta_0 + \beta_j x_{ij}, \text{ 여기서, } i = 1, \dots, m, j = 1, \dots, n.$$

행렬 표기법에서, 데이터 집합은 $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$ with $\mathbf{x}_j = \{x_{ij}\}_{i=1}^m$, $\mathbf{y} = \{y_i\}_{i=1}^m$ 로 쓰여집니다 (그림. 특징 행렬과 라벨). $\boldsymbol{\beta}^\top = \{\beta_j\}_{j=1}^n$. 행렬 형식 방정식은 다음과 같이 씁니다.

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta}. \quad (9.1)$$

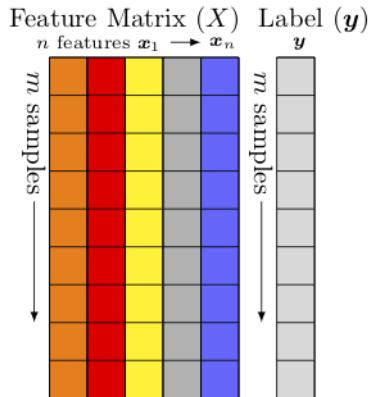


그림. 1: 특징 행렬과 라벨

9.1.2 회귀방정식을 어떻게 풀까요?

1. 직접적인 방법들(자세한 내용은 수치 분석을 위한 사전 참고 노트를 참조하시길 바랍니다)

- 제곱 행렬 또는 직사각형 행렬의 경우
 - 특이값 분해
 - 그램-슈미트(Gram-Schmidt) 직교화
 - QR 분해
- 제곱 행렬의 경우
 - LU 분해
 - 콜레스키(Cholesky) 분해
 - 규칙 분할

2. 반복적인 방법들

- 정적 사례 반복법
 - 야코비(Jacobi) 방법
 - 가우스-자이델(Gauss-Seidel) 방법
 - 리차드슨(Richardson) 방법
 - 축차 가속 완화(SOR) 방법
- 동적 사례 반복법
 - 체비셰프(Chebyshev) 반복법
 - 최소 잔차 방법
 - 최소 교정 반복법
 - 최급강하법

- 결례기울기법

9.1.3 최소제곱법

수학에서 (9.1)은 굳게 자리잡은 하나의 체계입니다. 최소제곱법은 이 체계에 대한 대략적인 솔루션을 찾는 데 사용될 수 있습니다. (9.1)의 경우, 최소 제곱 공식은 다음으로부터 구할 수 있습니다:

$$\min_{\beta} \|\mathbf{X}\beta - \mathbf{y}\|, \quad (9.2)$$

해는 다음과 같은 정규 방정식으로 쓸 수 있습니다:

$$\beta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad (9.3)$$

여기서 T는 $(\mathbf{X}^T \mathbf{X})^{-1}$ 이 존재하는 경우(즉, \mathbf{X} 가 전체 열 순위를 갖는 경우) 행렬의 전치를 나타냅니다.

노트: 사실, (9.3)은 \mathbf{X}^T 를 (9.1)에 곱하고 $(\mathbf{X}^T \mathbf{X})^{-1}$ 을 이전 결과의 양쪽에 곱하는 방법으로 유도할 수 있습니다. 최대 최소 정리를 (9.2)에 적용해 해 (9.3)를 찾을 수도 있습니다.

9.1.4 경사 하강법

다음 가설을 사용해봅시다:

$$h_{\beta} = \beta_0 + \beta_j \mathbf{x}_j, \text{ 여기서, } j = 1, \dots, n.$$

그렇다면 (9.2)의 해는 다음과 같은 비용 함수를 최소화하는 것과 같습니다:

9.1.5 비용 함수

$$J(\beta) = \frac{1}{2m} \sum_{i=1}^m \left(h_{\beta}(x^{(i)}) - y^{(i)} \right)^2 \quad (9.4)$$

노트: 우리가 (9.2)보다 (9.4)로 해를 찾는 것을 선호하는 이유는 (9.4)가 볼록(convex)하고 충분히 작은 학습 속도에 대해 에너지가 안정적이고 유일하게 해를 찾을 수 있기 때문입니다. 오목하지 않은 비용함수에 사례에 대해 관심 있는 독자분들은 [Feng2016PSD]에 자세한 내용 나와 있으니 이를 참조하시길 바랍니다.

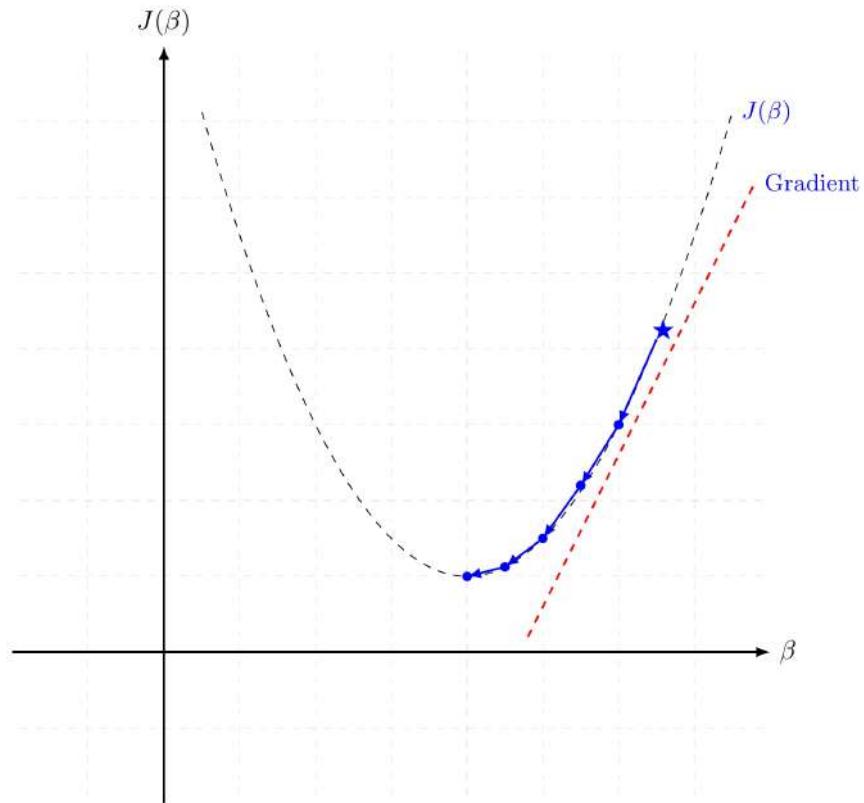


그림. 2: 일차원에서의 경사 하강법

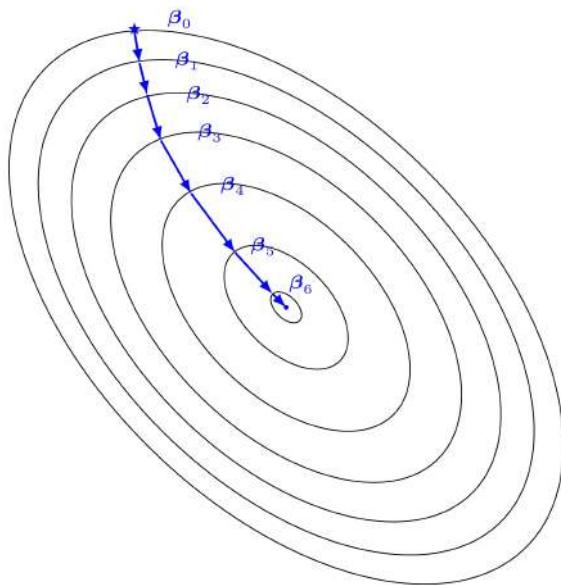


그림. 3: 이차원에서의 경사 하강법

9.1.6 일괄 경사 하강법(Batch Gradient Descent)

경사 하강법은 함수의 최소값을 찾기 위한 1차 반복 최적화 알고리즘입니다. 이는 학습률(검색단계) α 와 함께 기울기의 음에 의해 정의되는 가장 가파른 하강 방향을 탐색합니다.(참조 그림. 일차원에서 경사 하강법과 2차원에서 경사 하강법, 1차원과 2차원 각각에 대해)

9.1.7 확률적 경사 하강법

9.1.8 미니-배치 경사 하강법

9.1.9 데모

- 주피터 노트북은 파이프라인을 사용하지 않고 구현된 선형 회귀에서 다운로드할 수 있습니다.
- 주피터 노트북은 파이프라인을 사용하여 구현된 파이프라인 선형 회귀에서 다운로드할 수 있습니다.
- 코드는 아래와 같이 파이프라인 스타일로만 제시하겠습니다.
- 파라미터에 대한 자세한 내용은 선형 회귀 API를 참조하시길 바랍니다.

1. 스파크 컨텍스트 및 스파크세션 설정

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark regression example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. 데이터셋 로드하기

```
df = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', \
    inferschema='true') \
    .load("../data/Advertising.csv", header=True);
```

데이터셋 확인하기

```
df.show(5, True)
df.printSchema()
```

여러분은 아래와 같은 결과를 얻을 수 있습니다.

	TV		Radio		Newspaper		Sales	
+-----+-----+-----+-----+								
	230.1		37.8		69.2		22.1	

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
| 44.5| 39.3|      45.1| 10.4|
| 17.2| 45.9|      69.3|   9.3|
|151.5| 41.3|      58.5| 18.5|
|180.8| 10.8|      58.4| 12.9|
+----+----+----+----+
only showing top 5 rows
```

```
root
| -- TV: double (nullable = true)
| -- Radio: double (nullable = true)
| -- Newspaper: double (nullable = true)
| -- Sales: double (nullable = true)
```

여러분은 데이터 프레임으로부터 통계를 얻을 수 있습니다(이는 수치형 변수에만 해당됩니다).

```
df.describe().show()
```

여러분은 아래와 같은 결과를 얻을 수 있습니다.

```
+-----+-----+-----+-----+
| summary |           TV |           Radio |       Newspaper |
+-----+-----+-----+-----+
| count |      200 |      200 |      200 |
| mean | 147.0425 | 23.26400000000024 | 30.55399999999995 | 14.
| stddev | 85.85423631490805 | 14.846809176168728 | 21.77862083852283 | 5.
| min |      0.7 |      0.0 |      0.3 |
| max |    296.4 |     49.6 |    114.0 |
+-----+-----+-----+-----+
```

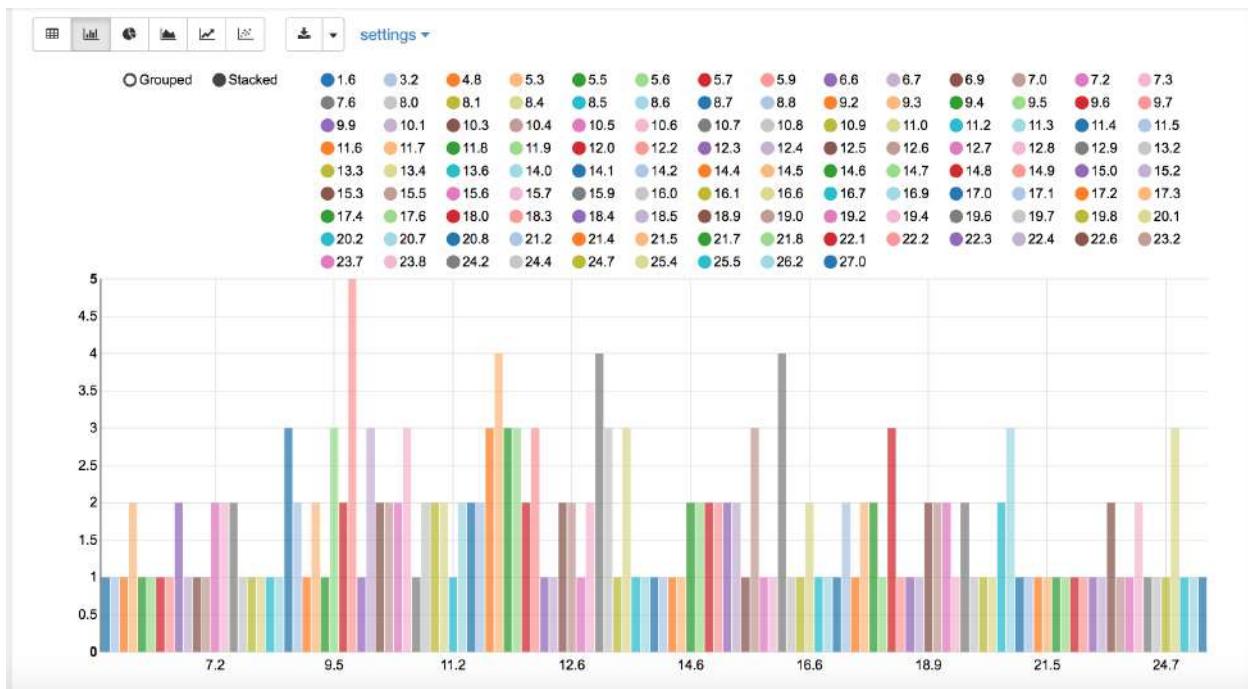


그림. 4: 판매 분포

3. 데이터를 밀집 벡터로 변환하기(특징과 라벨)

```
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

# 특징 및 라벨을 만드는 두 가지 방법을 작성해 보겠습니다

# 방법 1 (적은 특징에 적합):
def transData(row):
    #     return Row(label=row["Sales"],
    #                 features=Vectors.dense([row["TV"],
    #                                         row["Radio"],
    #                                         row["Newspaper"]]))
# 

# 방법 2 (많은 특징에 적합):
def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1]), r[-1]]).toDF(['features',
    ↵'label'])
```

노트:

데이터 셋의 범주형 데이터를 처리하기 위해 get_dummy 함수를 사용해 보시길 적극 권장 합니다.

지도 학습 버전:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols,
    ↵labelCol):
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer,
→OneHotEncoder, VectorAssembler
from pyspark.sql.functions import col

indexers = [ StringIndexer(inputCol=c, outputCol="{0}_"
→indexed".format(c))
            for c in categoricalCols ]

# 기본 설정: dropLast=True
encoders = [ OneHotEncoder(inputCol=indexer.
→getOutputCol(),
                           outputCol="{0}_encoded".format(indexer.
→getOutputCol()))
            for indexer in indexers ]

assembler = VectorAssembler(inputCols=[encoder.
→getOutputCol() for encoder in encoders]
                             + continuousCols, outputCol=
→"features")

pipeline = Pipeline(stages=indexers + encoders +_
→[assembler])

model=pipeline.fit(df)
data = model.transform(df)

data = data.withColumn('label', col(labelCol))

if indexCol:
    return data.select(indexCol, 'features', 'label')
else:
    return data.select('features', 'label')

```

비지도학습 버전:

```

def get_dummy(df, indexCol, categoricalCols, continuousCols):
    """
    더미 변수를 가져와 비지도 학습을 위해 연속 변수와 결합합니다.

    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터의 이름 목록
    :반환 k: 특징 행렬

    :작자: Wenqiang Feng
    :이메일: von198@gmail.com
    """

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

indexers = [ StringIndexer(inputCol=c, outputCol="{0}_  
→indexed".format(c))  
            for c in categoricalCols ]  
  
# 기본 설정: dropLast=True  
encoders = [ OneHotEncoder(inputCol=indexer.  
→getOutputCol(),  
                           outputCol="{0}_encoded".format(indexer.  
→getOutputCol()))  
            for indexer in indexers ]  
  
assembler = VectorAssembler(inputCols=[encoder.  
→getOutputCol() for encoder in encoders]  
                            + continuousCols, outputCol=  
→"features")  
  
pipeline = Pipeline(stages=indexers + encoders +  
→[assembler])  
  
model=pipeline.fit(df)  
data = model.transform(df)  
  
if indexCol:  
    return data.select(indexCol,'features')  
else:  
    return data.select('features')

```

두 가지 버전을 한 번에 적용하기:

```

def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol,  
→dropLast=False):  
  
    ...  
    더미 변수를 가져와 머신러닝 모델링을 위한 연속형 변수와 결합합니다.  
  
    :param df: 데이터 프레임  
    :param categoricalCols: 범주형 데이터의 이름 목록  
    :param continuousCols: 수치형 데이터의 이름 목록  
    :param labelCol: 라벨 열의 이름  
    :param dropLast: 마지막 열을 버릴 것인지를 설정하는 옵션  
    :반환: 특징 행렬  
  
    :저자: Wenqiang Feng  
    :이메일: von198@gmail.com  
  
>>> df = spark.createDataFrame([  
        (0, "a"),  
        (1, "b"),  
        (2, "c"),  
        (3, "a"),  
        (4, "a"),  
        (5, "c")]

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

        ], ["id", "category"])

>>> indexCol = 'id'
>>> categoricalCols = ['category']
>>> continuousCols = []
>>> labelCol = []

>>> mat = get_dummy(df, indexCol, categoricalCols, continuousCols,
↪labelCol)
>>> mat.show()

>>>
+---+-----+
| id|    features|
+---+-----+
|  0|[1.0,0.0,0.0]|
|  1|[0.0,0.0,1.0]|
|  2|[0.0,1.0,0.0]|
|  3|[1.0,0.0,0.0]|
|  4|[1.0,0.0,0.0]|
|  5|[0.0,1.0,0.0]|
+---+-----+
   ..

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder,_
↪VectorAssembler
from pyspark.sql.functions import col

indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
             for c in categoricalCols ]

# 기본 설정: dropLast=True
encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                           outputCol="{0}_encoded".format(indexer.
↪getOutputCol()), dropLast=False)
             for indexer in indexers ]

assembler = VectorAssembler(inputCols=[encoder.getOutputCol()].
↪for encoder in encoders]
                           + continuousCols, outputCol="features
↪")

pipeline = Pipeline(stages=indexers + encoders + [assembler])

model=pipeline.fit(df)
data = model.transform(df)

if indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

    return data.select(indexCol, 'features', 'label')
elif not indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select('features', 'label')
elif indexCol and not labelCol:
    # 비지도학습의 경우
    return data.select(indexCol, 'features')
elif not indexCol and not labelCol:
    # 비지도학습의 경우
    return data.select('features')

```

4. 데이터 셋을 데이터 프레임으로 변환하기

```
transformed= transData(df)
transformed.show(5)
```

```
+-----+----+
|      features|label |
+-----+----+
| [230.1,37.8,69.2]| 22.1|
| [44.5,39.3,45.1]| 10.4|
| [17.2,45.9,69.3]|  9.3|
| [151.5,41.3,58.5]| 18.5|
| [180.8,10.8,58.4]| 12.9|
+-----+----+
only showing top 5 rows
```

노트: Spark의 모든 지도 기계 학습 알고리즘이 **특징과 라벨**을 기반으로 한다는 것을 알게 될 것입니다(Spark에서 비지도 기계 학습 알고리즘은 특징을 기반으로 합니다). 즉, 파이프라인 아키텍처에 **특징과 라벨**을 준비되면 Spark의 모든 머신러닝 알고리즘을 사용할 수 있습니다.

5. 범주형 변수 처리하기

```

from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

# 범주형 특징을 자동으로 식별하고 인덱싱합니다.
# maxCategories를 지정하여 4개 이상의 고유 값을 가진 특징이 연속형으로 처리되도록 합니다.

featureIndexer = VectorIndexer(inputCol="features", \
                                outputCol="indexedFeatures", \
                                maxCategories=4).fit(transformed)

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
data = featureIndexer.transform(transformed)
```

이 시점에서 데이터를 확인하면

```
data.show(5, True)
```

아래와 같습니다.

```
+-----+-----+-----+
|       features|label| indexedFeatures|
+-----+-----+-----+
| [230.1,37.8,69.2]| 22.1|[230.1,37.8,69.2]|
| [44.5,39.3,45.1]| 10.4|[44.5,39.3,45.1]|
| [17.2,45.9,69.3]|  9.3|[17.2,45.9,69.3]|
| [151.5,41.3,58.5]| 18.5|[151.5,41.3,58.5]|
| [180.8,10.8,58.4]| 12.9|[180.8,10.8,58.4]|
+-----+-----+-----+
only showing top 5 rows
```

6. 데이터를 훈련 및 테스트 셋으로 분할하기(40%는 테스트 셋).

```
# 데이터를 훈련 및 테스트 셋으로 분할합니다(40%는 테스트 셋)
(trainingData, testData) = transformed.randomSplit([0.6, 0.4])
```

다음과 같이 훈련 및 테스트 데이터를 확인할 수 있습니다 (프로토타입 구문 동안 데이터를 계속 확인해보는 것이 좋습니다):

```
trainingData.show(5)
testData.show(5)
```

아래와 같은 결과를 얻을 수 있습니다.

```
+-----+-----+-----+
|       features|label| indexedFeatures|
+-----+-----+-----+
| [4.1,11.6,5.7]| 3.2|[4.1,11.6,5.7]|
| [5.4,29.9,9.4]| 5.3|[5.4,29.9,9.4]|
| [7.3,28.1,41.4]| 5.5|[7.3,28.1,41.4]|
| [7.8,38.9,50.6]| 6.6|[7.8,38.9,50.6]|
| [8.6,2.1,1.0]| 4.8|[8.6,2.1,1.0]|
+-----+-----+-----+
only showing top 5 rows

+-----+-----+-----+
|       features|label| indexedFeatures|
+-----+-----+-----+
| [0.7,39.6,8.7]| 1.6|[0.7,39.6,8.7]|
| [8.4,27.2,2.1]| 5.7|[8.4,27.2,2.1]|
| [11.7,36.9,45.2]| 7.3|[11.7,36.9,45.2]|
| [13.2,15.9,49.6]| 5.6|[13.2,15.9,49.6]|
+-----+-----+-----+
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
| [16.9, 43.7, 89.4] |  8.7 | [16.9, 43.7, 89.4] |
+-----+-----+-----+
only showing top 5 rows
```

7. 일반 최소 제곱 회귀 모형 적합하기

파라미터에 대한 자세한 설명은 [회귀 분석 API](#)를 참조하세요.

```
# 선형 회귀 클래스 불러오기
from pyspark.ml.regression import LinearRegression

# 선형 회귀 알고리즘 정의하기
lr = LinearRegression()
```

8. 파이프라인 아키텍처

```
# 파이프라인에서 인덱서 및 트리 뮤기
pipeline = Pipeline(stages=[featureIndexer, lr])

model = pipeline.fit(trainingData)
```

9. 모델 요약하기

Spark는 데이터와 모델에 대해 좋지 않은 요약을 가지고 있습니다. 아래 PySpark에서 회귀분석을 위해 **R** 출력 형식과 비슷한 요약 함수를 작성했습니다.

```
def modelsummary(model):
    import numpy as np
    print ("Note: the last rows are the information for Intercept")
    print ("##", "-----")
    print ("##", " Estimate | Std.Error | t Values | P-value")
    coef = np.append(list(model.coefficients),model.intercept)
    Summary=model.summary

    for i in range(len(Summary.pValues)):
        print ("##", '{:10.6f}'.format(coef[i]), \
              '{:10.6f}'.format(Summary.coefficientStandardErrors[i]), \
              '{:8.3f}'.format(Summary.tValues[i]), \
              '{:10.6f}'.format(Summary.pValues[i]))

    print ("##", '---')
    print ("##", "Mean squared error: % .6f" \
          % Summary.meanSquaredError, ", RMSE: % .6f" \
          % Summary.rootMeanSquaredError )
    print ("##", "Multiple R-squared: %f" % Summary.r2, ", \
          Total iterations: %i" % Summary.totalIterations)
```

```
modelsummary(model.stages[-1])
```

여러분은 요약 출력 결과를 다음과 같이 확인할 수 있습니다.

```
Note: the last rows are the information for Intercept
('##', '-----')
('##', ' Estimate | Std.Error | t Values | P-value')
('##', ' 0.044186', ' 0.001663', ' 26.573', ' 0.000000')
('##', ' 0.206311', ' 0.010846', ' 19.022', ' 0.000000')
('##', ' 0.001963', ' 0.007467', ' 0.263', ' 0.793113')
('##', ' 2.596154', ' 0.379550', ' 6.840', ' 0.000000')
('##', '---')
('##', 'Mean squared error: 2.588230', 'RMSE: 1.608798')
('##', 'Multiple R-squared: 0.911869', 'Total iterations: 1')
```

10. 예측값 만들기

```
# 예측값 만들기
predictions = model.transform(testData)
```

```
# 표시할 예제 행 선택하기
predictions.select("features", "label", "prediction").show(5)
```

features	label	prediction
[0.7, 39.6, 8.7]	1.6	10.81405928637388
[8.4, 27.2, 2.1]	5.7	8.583086404079918
[11.7, 36.9, 45.2]	7.3	10.814712818232422
[13.2, 15.9, 49.6]	5.6	6.557106943899219
[16.9, 43.7, 89.4]	8.7	12.534151375058645

only showing top 5 rows

9. 평가하기

```
from pyspark.ml.evaluation import RegressionEvaluator
# (예측, 실제 라벨)을 선택하고 검정 오차를 계산하기
evaluator = RegressionEvaluator(labelCol="label",
                                 predictionCol="prediction",
                                 metricName="rmse")

rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

최종 RMSE(Root Mean Squared Error)는 다음과 같습니다:

```
Root Mean Squared Error (RMSE) on test data = 1.63114
```

검증 데이터의 R^2 값은 다음과 같습니다:

```
y_true = predictions.select("label").toPandas()
y_pred = predictions.select("prediction").toPandas()

import sklearn.metrics
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {}'.format(r2_score))
```

여러분은 다음과 같은 결과를 얻을 수 있습니다.

```
r2_score: 0.854486655585
```

경고: 여러분은 대부분의 소프트웨어가 모델에 상수항이 없을 때 결정 계수를 계산하기 위해 서로 다른 공식을 사용한다는 것을 알고 계셔야 합니다. 여러분은 StackExchange에서 더 많은 정보를 확인하실 수 있으실 것입니다.

9.2 일반화 선형 회귀

9.2.1 도입

9.2.2 회귀방정식을 어떻게 풀까요?

9.2.3 데모

- 주피터 노트북은 일반화 선형 회귀에서 다운로드할 수 있습니다.
 - 파라미터에 대한 자세한 내용은 일반화 선형 회귀 API를 참조하시길 바랍니다.
- 스파크 컨텍스트 및 스파크세션 설정

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark regression example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. 데이터셋 로드하기

```
df = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', \
    inferschema='true') \
    .load("../data/Advertising.csv", header=True);
```

데이터셋 확인하기

```
df.show(5, True)
df.printSchema()
```

여러분은 아래와 같은 결과를 확인할 수 있을 것입니다.

```
+----+----+----+----+
|   TV|Radio|Newspaper|Sales|
+----+----+----+----+
| 230.1| 37.8|    69.2| 22.1|
| 44.5| 39.3|    45.1| 10.4|
| 17.2| 45.9|    69.3|  9.3|
|151.5| 41.3|    58.5| 18.5|
|180.8| 10.8|    58.4| 12.9|
+----+----+----+----+
only showing top 5 rows

root
|-- TV: double (nullable = true)
|-- Radio: double (nullable = true)
|-- Newspaper: double (nullable = true)
|-- Sales: double (nullable = true)
```

여러분은 데이터 프레임으로부터 통계를 얻을 수 있습니다(이는 수치형 변수에만 해당됩니다).

```
df.describe().show()
```

여러분은 아래와 같은 결과를 얻을 수 있습니다

```
+----+-----+-----+-----+-----+
|summary|          TV|        Radio|      Newspaper|       Sales|
+----+-----+-----+-----+-----+
|  count|        200|        200|        200|        200|
|  mean | 147.0425|23.26400000000024|30.55399999999995|14.02250000000003|
| stddev|85.85423631490805|14.846809176168728| 21.77862083852283| 5.217456565710477|
|  min |        0.7|        0.0|        0.3|        1.6|
|  max |      296.4|       49.6|      114.0|       27.0|
+----+-----+-----+-----+-----+
```

3. 데이터를 밀집 벡터로 변환하기(특징과 라벨)

노트:

범주형 데이터를 처리하기 위해 `get_dummies` 함수를 사용해 보시길 적극 권장합니다.

지도 학습 버전:

```

def get_dummy(df, indexCol, categoricalCols, continuousCols,
    ↪labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer,
    ↪OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_"
    ↪indexed".format(c))
        for c in categoricalCols ]

    # 기본 설정: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.
    ↪getOutputCol(),
        outputCol="{0}_encoded".format(indexer.
    ↪getOutputCol()))
        for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.
    ↪getOutputCol() for encoder in encoders]
        + continuousCols, outputCol=
    ↪"features")

    pipeline = Pipeline(stages=indexers + encoders +_
    ↪[assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label', col(labelCol))

    if indexCol:
        return data.select(indexCol, 'features', 'label')
    else:
        return data.select('features', 'label')

```

비지도학습 버전:

```

def get_dummy(df, indexCol, categoricalCols, continuousCols):
    """
    더미 변수를 가져와 비지도 학습을 위해 연속 변수와 결합합니다.

    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터의 이름 목록

    :반환 k: 특징 행렬

    :저자: Wenqiang Feng
    :이메일: von198@gmail.com

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

    ...
    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_"
→indexed".format(c))
        for c in categoricalCols ]

    # 기본 설정: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.
→getOutputCol(),
        outputCol="{0}_encoded".format(indexer.
→getOutputCol()))
        for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.
→getOutputCol() for encoder in encoders]
        + continuousCols, outputCol=
→"features")

    pipeline = Pipeline(stages=indexers + encoders +_
→[assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    if indexCol:
        return data.select(indexCol, 'features')
    else:
        return data.select('features')

```

두 가지 버전을 한 번에 적용하기:

```

def get_dummy(df, indexCol, categoricalCols, continuousCols, labelCol,
→dropLast=False):

    ...
    더미 변수를 가져와 머신러닝 모델링을 위한 연속형 변수와 결합합니다.

    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터의 이름 목록
    :param labelCol: 라벨 열의 이름
    :param dropLast: 마지막 열을 버릴 것인지를 설정하는 옵션
    :반환: 특징 행렬

    :작자: Wenqiang Feng
    :이메일: von198@gmail.com

>>> df = spark.createDataFrame([
        (0, "a"),
        (1, "b"),
        (2, "c"),
        (3, "a"),

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

        (4, "a"),
        (5, "c")
    ], ["id", "category"])

>>> indexCol = 'id'
>>> categoricalCols = ['category']
>>> continuousCols = []
>>> labelCol = []

>>> mat = get_dummy(df, indexCol, categoricalCols, continuousCols,
labelCol)
>>> mat.show()

>>>
+---+-----+
| id| features|
+---+-----+
| 0|[1.0,0.0,0.0]|
| 1|[0.0,0.0,1.0]|
| 2|[0.0,1.0,0.0]|
| 3|[1.0,0.0,0.0]|
| 4|[1.0,0.0,0.0]|
| 5|[0.0,1.0,0.0]|
+---+-----+
'''

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
VectorAssembler
from pyspark.sql.functions import col

indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
format(c))
             for c in categoricalCols ]

# 기본 설정: dropLast=True
encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
outputCol="{0}_encoded".format(indexer.
getOutputCol()), dropLast=dropLast)
             for indexer in indexers ]

assembler = VectorAssembler(inputCols=[encoder.getOutputCol()].
for encoder in encoders]
                     + continuousCols, outputCol="features"
)

pipeline = Pipeline(stages=indexers + encoders + [assembler])

model=pipeline.fit(df)
data = model.transform(df)

if indexCol and labelCol:

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
# 지도 학습의 경우
data = data.withColumn('label', col(labelCol))
return data.select(indexCol, 'features', 'label')
elif not indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select('features', 'label')
elif indexCol and not labelCol:
    # 비지도 학습의 경우
    return data.select(indexCol, 'features')
elif not indexCol and not labelCol:
    # 비지도 학습의 경우
    return data.select('features')
```

```
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

# 특징 및 라벨을 만드는 두 가지 방법을 작성해 보겠습니다

# 방법 1 (적은 특징에 적합):
def transData(row):
    #     return Row(label=row["Sales"],
    #                 features=Vectors.dense([row["TV"],
    #                                         row["Radio"],
    #                                         row["Newspaper"]]))

# 방법 2 (많은 특징에 적합):
def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1]), r[-1]]).toDF(['features',
                                                               'label'])
```

```
transformed= transData(df)
transformed.show(5)
```

```
+-----+----+
|      features|label|
+-----+----+
| [230.1,37.8,69.2]| 22.1|
| [44.5,39.3,45.1]| 10.4|
| [17.2,45.9,69.3]|  9.3|
| [151.5,41.3,58.5]| 18.5|
| [180.8,10.8,58.4]| 12.9|
+-----+----+
only showing top 5 rows
```

노트: Spark의 모든 지도 기계 학습 알고리즘이 **특징과 라벨**을 기반으로 한다는 것을 알게 될 것입니다. 즉, **특징과 라벨**을 준비되면 Spark의 모든 머신러닝 알고리즘을 사용할 수 있습니다.

4. 데이터셋을 고밀도 벡터로 변환하기

```
# 데이터셋을 고밀도 벡터로 변환하기
def transData(data):
    return data.rdd.map(lambda r: [r[-1], Vectors.dense(r[:-1])]).\
        toDF(['label', 'features'])

from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

data= transData(df)
data.show()
```

5. 범주형 변수 처리하기

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

# 범주형 특징을 자동으로 식별하고 인덱싱합니다
# maxCategories를 지정하여 4개 이상의 고유 값을 가진 특징이 연속형으로 처리되도록 합니다

featureIndexer = VectorIndexer(inputCol="features", \
                                outputCol="indexedFeatures", \
                                maxCategories=4).fit(transformed)

data = featureIndexer.transform(transformed)
```

이 시점에서 데이터를 확인하면 아래와 같습니다

```
+-----+-----+-----+
|      features|label| indexedFeatures|
+-----+-----+-----+
| [230.1, 37.8, 69.2] | 22.1 | [230.1, 37.8, 69.2] |
| [44.5, 39.3, 45.1] | 10.4 | [44.5, 39.3, 45.1] |
| [17.2, 45.9, 69.3] |  9.3 | [17.2, 45.9, 69.3] |
| [151.5, 41.3, 58.5] | 18.5 | [151.5, 41.3, 58.5] |
| [180.8, 10.8, 58.4] | 12.9 | [180.8, 10.8, 58.4] |
+-----+-----+-----+
only showing top 5 rows
```

6. 데이터를 훈련 및 테스트 셋으로 분할하기(40%는 테스트 셋).

```
# 데이터를 훈련 및 테스트 셋으로 분할합니다(40%는 테스트 셋)
(trainingData, testData) = transformed.randomSplit([0.6, 0.4])
```

다음과 같이 훈련 및 테스트 데이터를 확인할 수 있습니다(프로토타입 구문 동안 데이터를 계속 확인해보는 것이 좋습니다):

```
trainingData.show(5)  
testData.show(5)
```

아래와 같은 결과를 얻을 수 있습니다

```
+-----+-----+-----+  
|       features|label| indexedFeatures|  
+-----+-----+-----+  
| [5.4,29.9,9.4]| 5.3| [5.4,29.9,9.4] |  
| [7.8,38.9,50.6]| 6.6| [7.8,38.9,50.6] |  
| [8.4,27.2,2.1]| 5.7| [8.4,27.2,2.1] |  
| [8.7,48.9,75.0]| 7.2| [8.7,48.9,75.0] |  
| [11.7,36.9,45.2]| 7.3| [11.7,36.9,45.2] |  
+-----+-----+-----+  
only showing top 5 rows  
  
+-----+-----+-----+  
|       features|label| indexedFeatures|  
+-----+-----+-----+  
| [0.7,39.6,8.7]| 1.6| [0.7,39.6,8.7] |  
| [4.1,11.6,5.7]| 3.2| [4.1,11.6,5.7] |  
| [7.3,28.1,41.4]| 5.5|[7.3,28.1,41.4] |  
| [8.6,2.1,1.0]| 4.8| [8.6,2.1,1.0] |  
| [17.2,4.1,31.6]| 5.9|[17.2,4.1,31.6] |  
+-----+-----+-----+  
only showing top 5 rows
```

7. 일반화 선형 회귀 모형 적합하기

```
# 선형 회귀 class 불러오기  
from pyspark.ml.regression import GeneralizedLinearRegression  
  
# 선형 회귀 알고리즘 정의하기  
glr = GeneralizedLinearRegression(family="gaussian", link="identity",  
                                    maxIter=10, regParam=0.3)
```

8. 파일프라인 아키텍쳐

```
# 파일프라인에서 인덱서와 트리 뮤기  
pipeline = Pipeline(stages=[featureIndexer, glr])  
  
model = pipeline.fit(trainingData)
```

9. 모델 요약하기

Spark는 데이터와 모델에 대해 좋지 않은 요약을 가지고 있습니다. 아래 PySpark에서 회귀분석을 위해 R 출력 형식과 비슷한 요약 함수를 작성했습니다.

```
def modelsummary(model):  
    import numpy as np  
    print ("Note: the last rows are the information for Intercept")  
    print ("##", "-----")
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

print ("##", " Estimate | Std.Error | t Values | P-value")
coef = np.append(list(model.coefficients),model.intercept)
Summary=model.summary

for i in range(len(Summary.pValues)):
    print ("##", '{:10.6f}'.format(coef[i]), \
           '{:10.6f}'.format(Summary.coefficientStandardErrors[i]), \
           '{:8.3f}'.format(Summary.tValues[i]), \
           '{:10.6f}'.format(Summary.pValues[i]))

print ("##", '---')
#     print ("##", "Mean squared error: % .6f" \
#             % Summary.meanSquaredError, ", RMSE: % .6f" \
#             % Summary.rootMeanSquaredError )
#     print ("##", "Multiple R-squared: %f" % Summary.r2, ", \
#             Total iterations: %i"% Summary.totalIterations)

```

```
modelsummary(model.stages[-1])
```

여러분은 다음과 같이 요약 결과를 확인할 수 있습니다:

```

Note: the last rows are the information for Intercept
('##', '-----')
('##', ' Estimate | Std.Error | t Values | P-value')
('##', ' 0.042857', ' 0.001668', ' 25.692', ' 0.000000')
('##', ' 0.199922', ' 0.009881', ' 20.232', ' 0.000000')
('##', ' -0.001957', ' 0.006917', ' -0.283', ' 0.777757')
('##', ' 3.007515', ' 0.406389', ' 7.401', ' 0.000000')
('##', '---')

```

10. 예측값 만들기

```
# 예측값 만들기
predictions = model.transform(testData)
```

```
# 표시할 예제 행 선택하기
predictions.select("features", "label", "predictedLabel").show(5)
```

features	label	prediction
[0.7, 39.6, 8.7]	1.6	10.937383732327625
[4.1, 11.6, 5.7]	3.2	5.491166258750164
[7.3, 28.1, 41.4]	5.5	8.8571603947873
[8.6, 2.1, 1.0]	4.8	3.793966281660073
[17.2, 4.1, 31.6]	5.9	4.502507124763654

only showing top 5 rows

11. 평가하기

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.evaluation import RegressionEvaluator
# (예측, 실제 라벨)을 선택하고 검정 오차를 계산하기
evaluator = RegressionEvaluator(labelCol="label",
                                 predictionCol="prediction",
                                 metricName="rmse")

rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

최종 RMSE(Root Mean Squared Error)는 다음과 같습니다:

```
Root Mean Squared Error (RMSE) on test data = 1.89857
```

```
y_true = predictions.select("label").toPandas()
y_pred = predictions.select("prediction").toPandas()

import sklearn.metrics
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {}'.format(r2_score))
```

R^2 값은 다음과 같습니다:

```
r2_score: 0.87707391843
```

9.3 의사결정 나무 - 회귀

9.3.1 도입

9.3.2 어떻게 문제를 해결할까요?

9.3.3 데모

- 주피터 노트북은 의사결정 나무 회귀에서 다운로드할 수 있습니다.
 - 파라미터에 대한 자세한 내용은 의사결정 나무 회귀자 API를 참조하시길 바랍니다
- 스파크 컨텍스트 및 스파크세션 설정

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark regression example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

- 데이터셋 로드하기

```
df = spark.read.format('com.databricks.spark.csv') .\
    options(header='true', \
    inferSchema='true') .\
    load("../data/Advertising.csv", header=True);
```

데이터셋 확인하기

```
df.show(5, True)
df.printSchema()
```

여러분은 아래와 같은 결과를 확인할 수 있을 것 입니다

```
+----+----+----+----+
|   TV|Radio|Newspaper|Sales|
+----+----+----+----+
| 230.1| 37.8|     69.2| 22.1|
|  44.5| 39.3|     45.1| 10.4|
| 17.2| 45.9|     69.3|  9.3|
|151.5| 41.3|     58.5| 18.5|
|180.8| 10.8|     58.4| 12.9|
+----+----+----+----+
only showing top 5 rows

root
 |-- TV: double (nullable = true)
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)
```

여러분은 데이터프레임으로부터 통계를 얻을 수 있습니다(이는 수치형 변수에만 해당됩니다).

```
df.describe().show()
```

여러분은 아래와 같은 결과를 얻을 수 있습니다

```
+----+-----+-----+-----+
|summary|          TV|        Radio|      Newspaper|
|Sales|-----+-----+-----+
| count |      200 |      200 |      200 |
| mean | 147.0425|23.26400000000024|30.55399999999995|14.
| stddev | 85.85423631490805|14.846809176168728| 21.77862083852283| 5.
| 217456565710477 |
| min |      0.7 |      0.0 |      0.3 |
| 1.6 |-----+-----+-----+
| max |     296.4 |     49.6 |    114.0 |
| 27.0 |-----+-----+-----+
+----+-----+-----+-----+
```

(다음 페이지에 계속)

3. 데이터를 밀집 벡터로 변환하기(특징과 라벨)

노트:

범주형 데이터를 처리하기 위해 get_dummy 함수를 사용해 보시길 적극 권장합니다.

지도 학습 버전:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols,
              labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer,
        OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_"
        .format(c))
        for c in categoricalCols ]

    # 기본 설정: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.
        getOutputCol(),
        outputCol="{0}_encoded".format(indexer.
        getOutputCol()))
        for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.
        getOutputCol() for encoder in encoders]
        + continuousCols, outputCol=
        "features")

    pipeline = Pipeline(stages=indexers + encoders +_
        [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label', col(labelCol))

    return data.select(indexCol, 'features', 'label')
```

비지도학습 버전:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols):
    '''

    더미 변수를 가져와 비지도 학습을 위해 연속 변수와 결합합니다.
    '''


```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

:param df: 데이터 프레임
:param categoricalCols: 범주형 데이터의 이름 목록
:param continuousCols: 수치형 데이터의 이름 목록
'''

:반환 k: 특징 행렬

:저자: Wenqiang Feng
:이메일: von198@gmail.com
'''

indexers = [ StringIndexer(inputCol=c, outputCol="{0}_"
                           .format(c))
             for c in categoricalCols ]

# 기본 설정: dropLast=True
encoders = [ OneHotEncoder(inputCol=indexer.
                            .getOutputCol(),
                            outputCol="{0}_encoded".format(indexer.
                            .getOutputCol()))
              for indexer in indexers ]

assembler = VectorAssembler(inputCols=[encoder.
                                         .getOutputCol() for encoder in encoders]
                             + continuousCols, outputCol=
                                         "features")

pipeline = Pipeline(stages=indexers + encoders +_
                     [assembler])

model=pipeline.fit(df)
data = model.transform(df)

return data.select(indexCol, 'features')

```

두 가지 버전을 한 번에 적용하기:

```

def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol,
             dropLast=False):

'''

더미 변수를 가져와 머신러닝 모델링을 위한 연속형 변수와 결합합니다.

:param df: 데이터 프레임
:param categoricalCols: 범주형 데이터의 이름 목록
:param continuousCols: 수치형 데이터의 이름 목록
:param labelCol: 라벨 열의 이름
:param dropLast: 마지막 열을 버릴 것인지를 설정하는 옵션
:반환: 특징 행렬

:저자: Wenqiang Feng
:이메일: von198@gmail.com

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

>>> df = spark.createDataFrame([
    (0, "a"),
    (1, "b"),
    (2, "c"),
    (3, "a"),
    (4, "a"),
    (5, "c")
], ["id", "category"])

>>> indexCol = 'id'
>>> categoricalCols = ['category']
>>> continuousCols = []
>>> labelCol = []

>>> mat = get_dummy(df, indexCol, categoricalCols, continuousCols,
↪labelCol)
>>> mat.show()

>>>
+---+-----+
| id|    features|
+---+-----+
|  0|[1.0,0.0,0.0]|
|  1|[0.0,0.0,1.0]|
|  2|[0.0,1.0,0.0]|
|  3|[1.0,0.0,0.0]|
|  4|[1.0,0.0,0.0]|
|  5|[0.0,1.0,0.0]|
+---+-----+
   ..

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
↪VectorAssembler
from pyspark.sql.functions import col

indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
             for c in categoricalCols ]

# 기본 설정: dropLast=True
encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                           outputCol="{0}_encoded".format(indexer.
↪getOutputCol()), dropLast=False)
             for indexer in indexers ]

assembler = VectorAssembler(inputCols=[encoder.getOutputCol() ↪
↪for encoder in encoders]
                             + continuousCols, outputCol="features
↪")

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

pipeline = Pipeline(stages=indexers + encoders + [assembler])

model=pipeline.fit(df)
data = model.transform(df)

if indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select(indexCol,'features','label')
elif not indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select('features','label')
elif indexCol and not labelCol:
    # 비지도학습의 경우
    return data.select(indexCol,'features')
elif not indexCol and not labelCol:
    # 비지도 학습의 경우
    return data.select('features')

```

```

from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

# 특징 및 라벨을 만드는 두 가지 방법을 작성해 보겠습니다

# 방법 1 (적은 특징에 적합) :
def transData(row):
    return Row(label=row["Sales"],
               features=Vectors.dense([row["TV"],
                                      row["Radio"],
                                      row["Newspaper"]]))

# 방법 2 (많은 특징에 적합) :
def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1]), r[-1]]).toDF(['features',
                                                               'label'])

```

```

transformed= transData(df)
transformed.show(5)

```

```

+-----+-----+
|      features|label|
+-----+-----+
| [230.1,37.8,69.2]| 22.1 |
| [44.5,39.3,45.1]| 10.4 |
| [17.2,45.9,69.3]|  9.3 |
| [151.5,41.3,58.5]| 18.5 |
| [180.8,10.8,58.4]| 12.9 |
+-----+-----+
only showing top 5 rows

```

노트: Spark의 모든 지도 기계 학습 알고리즘이 **특징과 라벨**을 기반으로 한다는 것을 알게 될 것입니다. 즉, **특징과 라벨**을 준비되면 Spark의 모든 머신러닝 알고리즘을 사용할 수 있습니다.

4. 데이터 셋을 고밀도 벡터로 변환하기

```
# 데이터 셋을 고밀도 벡터로 변환하기
def transData(data):
    return data.rdd.map(lambda r: [r[-1], Vectors.dense(r[:-1])]).\
        toDF(['label', 'features'])

transformed = transData(df)
transformed.show(5)
```

5. 범주형 변수 처리하기

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

# 범주형 특징을 자동으로 식별하고 인덱싱합니다.
# maxCategories를 지정하여 4개 이상의 고유 값을 가진 특징이 연속형으로 처리되도록 합니다.

featureIndexer = VectorIndexer(inputCol="features", \
                                 outputCol="indexedFeatures", \
                                 maxCategories=4).fit(transformed)

data = featureIndexer.transform(transformed)
```

이 시점에서 데이터를 확인하면 아래와 같습니다.

```
+-----+-----+-----+
|      features|label| indexedFeatures|
+-----+-----+-----+
| [230.1, 37.8, 69.2] | 22.1 | [230.1, 37.8, 69.2] |
| [44.5, 39.3, 45.1] | 10.4 | [44.5, 39.3, 45.1] |
| [17.2, 45.9, 69.3] |  9.3 | [17.2, 45.9, 69.3] |
| [151.5, 41.3, 58.5] | 18.5 | [151.5, 41.3, 58.5] |
| [180.8, 10.8, 58.4] | 12.9 | [180.8, 10.8, 58.4] |
+-----+-----+-----+
only showing top 5 rows
```

6. 데이터를 훈련 및 테스트 셋으로 분할하기(40%는 테스트 셋).

```
# 데이터를 훈련 및 테스트 셋으로 분할합니다(40%는 테스트 셋).
(trainingData, testData) = transformed.randomSplit([0.6, 0.4])
```

다음과 같이 훈련 및 테스트 데이터를 확인할 수 있습니다(프로토타입 구문 동안 데이터를 계속 확인해보는 것이 좋습니다).

```
trainingData.show(5)
testData.show(5)
```

아래와 같은 결과를 얻을 수 있습니다.

```
+-----+-----+-----+
|      features|label|indexedFeatures|
+-----+-----+-----+
| [4.1,11.6,5.7]|  3.2| [4.1,11.6,5.7] |
| [7.3,28.1,41.4]|  5.5|[7.3,28.1,41.4] |
| [8.4,27.2,2.1]|  5.7|[8.4,27.2,2.1] |
| [8.6,2.1,1.0] |  4.8|[8.6,2.1,1.0] |
| [8.7,48.9,75.0]|  7.2|[8.7,48.9,75.0] |
+-----+-----+-----+
only showing top 5 rows

+-----+-----+-----+
|      features|label| indexedFeatures|
+-----+-----+-----+
| [0.7,39.6,8.7]|  1.6| [0.7,39.6,8.7] |
| [5.4,29.9,9.4] |  5.3| [5.4,29.9,9.4] |
| [7.8,38.9,50.6]|  6.6| [7.8,38.9,50.6] |
| [17.2,45.9,69.3]|  9.3|[17.2,45.9,69.3] |
| [18.7,12.1,23.4]|  6.7|[18.7,12.1,23.4] |
+-----+-----+-----+
only showing top 5 rows
```

7. 의사결정 나무 회귀 모형 적합하기

```
from pyspark.ml.regression import DecisionTreeRegressor

# 의사결정 나무 모형 학습하기
dt = DecisionTreeRegressor(featuresCol="indexedFeatures")
```

8. 파이프라인 아키텍처

```
# 파일에 인덱서와 트리 뮤기
pipeline = Pipeline(stages=[featureIndexer, dt])

model = pipeline.fit(trainingData)
```

9. 예측값 만들기

```
# 예측값 만들기
predictions = model.transform(testData)
```

```
# 표시할 예시 행 선택하기
predictions.select("features", "label", "predictedLabel").show(5)
```

```
+-----+-----+
|prediction|label|      features|
+-----+-----+
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

	7.2	1.6	[0.7, 39.6, 8.7]
	7.3	5.3	[5.4, 29.9, 9.4]
	7.2	6.6	[7.8, 38.9, 50.6]
	8.64	9.3	[17.2, 45.9, 69.3]
	6.45	6.7	[18.7, 12.1, 23.4]
+-----+-----+-----+			
only showing top 5 rows			

10. 평가하기

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.evaluation import RegressionEvaluator
# (예측, 실제 라벨)을 선택하고 검정 오차를 계산하기
evaluator = RegressionEvaluator(labelCol="label",
                                 predictionCol="prediction",
                                 metricName="rmse")

rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

최종 RMSE(Root Mean Squared Error)는 다음과 같습니다:

```
Root Mean Squared Error (RMSE) on test data = 1.50999
```

```
y_true = predictions.select("label").toPandas()
y_pred = predictions.select("prediction").toPandas()

import sklearn.metrics
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {}'.format(r2_score))
```

R^2 값은 다음과 같습니다:

```
r2_score: 0.911024318967
```

여러분은 또한 특징의 중요도도 확인할 수 있습니다:

```
model.stages[1].featureImportances
```

각 특징의 가중치는 다음과 같습니다.

```
SparseVector(3, {0: 0.6811, 1: 0.3187, 2: 0.0002})
```

9.4 랜덤 포레스트 - 회귀

9.4.1 도입

9.4.2 어떻게 문제를 해결할까?

9.4.3 데모

- 주피터 노트북은 랜덤 포레스트 회귀에서 다운로드할 수 있습니다.
 - 파라미터에 대한 자세한 내용은 랜덤 포레스트 회귀자 API를 참조하시길 바랍니다.
- 스파크 컨텍스트 및 스파크세션 설정

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark RandomForest Regression example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. 데이터셋 로드하기

```
df = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', \
    inferSchema='true') \
    .load("../data/Advertising.csv", header=True);

df.show(5, True)
df.printSchema()
```

```
+----+----+----+----+
| TV | Radio | Newspaper | Sales |
+----+----+----+----+
| 230.1 | 37.8 | 69.2 | 22.1 |
| 44.5 | 39.3 | 45.1 | 10.4 |
| 17.2 | 45.9 | 69.3 | 9.3 |
| 151.5 | 41.3 | 58.5 | 18.5 |
| 180.8 | 10.8 | 58.4 | 12.9 |
+----+----+----+----+
only showing top 5 rows

root
 |-- TV: double (nullable = true)
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)
```

```
df.describe().show()
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

	TV	Radio	Newspaper	
summary				
Sales				
count	200	200	200	
mean	147.0425 23.26400000000024 30.553999999999995 14.			
stddev	85.85423631490805 14.846809176168728 21.77862083852283 5.			
min	0.7 1.6 27.0	0.0	0.3	
max	296.4	49.6	114.0	

3. 데이터를 밀집 벡터로 변환하기(특징과 라벨)

노트:

범주형 데이터를 처리하기 위해 get_dummy 함수를 사용해 보시길 적극 권장합니다.

지도 학습 버전:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols,
             labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer,
    OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_"
    .format(c))
        for c in categoricalCols ]

    # 기본 설정: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.
    .getOutputCol(),
        outputCol="{0}_encoded".format(indexer.
    .getOutputCol()))
        for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.
    .getOutputCol() for encoder in encoders]
        + continuousCols, outputCol=
    "features")
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

    pipeline = Pipeline(stages=indexers + encoders +_
↪[assembler])

model=pipeline.fit(df)
data = model.transform(df)

data = data.withColumn('label', col(labelCol))

return data.select(indexCol,'features','label')

```

비지도학습 버전:

```

def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    더미 변수를 가져와 비지도 학습을 위해 연속 변수와 결합합니다.

    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터의 이름 목록

    :반환 k: 특징 행렬

    :저자: Wenqiang Feng
    :이메일: von198@gmail.com
    '''

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}__"
↪indexed".format(c))
        for c in categoricalCols ]

    # 기본 설정: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.
↪getOutputCol(),
        outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
        for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.
↪getOutputCol() for encoder in encoders]
                                + continuousCols, outputCol=
↪"features")

    pipeline = Pipeline(stages=indexers + encoders +_
↪[assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

return data.select(indexCol,'features')

```

두 가지 버전을 한 번에 적용하기:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols, labelCol,
    ↪dropLast=False):

    """
    더미 변수를 가져와 머신러닝 모델링을 위한 연속형 변수와 결합합니다.

    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터의 이름 목록
    :param labelCol: 라벨 열의 이름
    :param dropLast: 마지막 열을 버릴 것인지를 설정하는 옵션
    :반환 : 특징 행렬

    :작자: Wenqiang Feng
    :이메일: von198@gmail.com

>>> df = spark.createDataFrame([
        (0, "a"),
        (1, "b"),
        (2, "c"),
        (3, "a"),
        (4, "a"),
        (5, "c")
    ], ["id", "category"])

>>> indexCol = 'id'
>>> categoricalCols = ['category']
>>> continuousCols = []
>>> labelCol = []

>>> mat = get_dummy(df, indexCol, categoricalCols, continuousCols,
    ↪labelCol)
>>> mat.show()

>>>
+---+-----+
| id|    features|
+---+-----+
|  0|[1.0,0.0,0.0]|
|  1|[0.0,0.0,1.0]|
|  2|[0.0,1.0,0.0]|
|  3|[1.0,0.0,0.0]|
|  4|[1.0,0.0,0.0]|
|  5|[0.0,1.0,0.0]|
+---+-----+
    """

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
    ↪VectorAssembler
from pyspark.sql.functions import col
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
    ↪format(c))
            for c in categoricalCols ]

# 기본 설정: dropLast=True
encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
    outputCol="{0}_encoded".format(indexer.
    ↪getOutputCol()), dropLast=dropLast)
            for indexer in indexers ]

assembler = VectorAssembler(inputCols=[encoder.getOutputCol()].
    ↪for encoder in encoders]
                    + continuousCols, outputCol="features"
    ↪" )

pipeline = Pipeline(stages=indexers + encoders + [assembler])

model=pipeline.fit(df)
data = model.transform(df)

if indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select(indexCol, 'features', 'label')
elif not indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select('features', 'label')
elif indexCol and not labelCol:
    # 비지도학습의 경우
    return data.select(indexCol, 'features')
elif not indexCol and not labelCol:
    # 비지도학습의 경우
    return data.select('features')

```

```

from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

# 데이터를 고밀도 벡터로 변환하기
def transData(row):
#     return Row(label=row["Sales"],
#                 features=Vectors.dense([row["TV"],
#                                         row["Radio"],
#                                         row["Newspaper"]]))
def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1]), r[-1]]).toDF([
        ↪'features', 'label'])

```

4. 데이터셋을 고밀도 벡터로 변환하기

```
transformed= transData(df)
transformed.show(5)
```

```
+-----+-----+
|       features|label|
+-----+-----+
| [230.1,37.8,69.2]| 22.1 |
| [44.5,39.3,45.1]| 10.4 |
| [17.2,45.9,69.3]|  9.3 |
| [151.5,41.3,58.5]| 18.5 |
| [180.8,10.8,58.4]| 12.9 |
+-----+-----+
only showing top 5 rows
```

5. 범주형 변수 처리하기

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

featureIndexer = VectorIndexer(inputCol="features", \
                                outputCol="indexedFeatures", \
                                maxCategories=4).fit(transformed)

data = featureIndexer.transform(transformed)
data.show(5, True)
```

```
+-----+-----+-----+
|       features|label| indexedFeatures|
+-----+-----+-----+
| [230.1,37.8,69.2]| 22.1|[230.1,37.8,69.2] |
| [44.5,39.3,45.1]| 10.4|[44.5,39.3,45.1] |
| [17.2,45.9,69.3]|  9.3|[17.2,45.9,69.3] |
| [151.5,41.3,58.5]| 18.5|[151.5,41.3,58.5] |
| [180.8,10.8,58.4]| 12.9|[180.8,10.8,58.4] |
+-----+-----+-----+
only showing top 5 rows
```

6. 데이터를 훈련 및 테스트 셋으로 분할하기(40%는 테스트 셋)

```
# 데이터를 훈련 및 테스트 셋으로 분할합니다(40%는 테스트 셋)
(trainingData, testData) = data.randomSplit([0.6, 0.4])

trainingData.show(5)
testData.show(5)
```

```
+-----+-----+-----+
|       features|label| indexedFeatures|
+-----+-----+-----+
| [0.7,39.6,8.7]| 1.6|[0.7,39.6,8.7] |
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
| [8.6,2.1,1.0] | 4.8| [8.6,2.1,1.0] |
| [8.7,48.9,75.0] | 7.2| [8.7,48.9,75.0] |
| [11.7,36.9,45.2] | 7.3| [11.7,36.9,45.2] |
| [13.2,15.9,49.6] | 5.6| [13.2,15.9,49.6] |
+-----+-----+
only showing top 5 rows

+-----+-----+
| features|label|indexedFeatures|
+-----+-----+
| [4.1,11.6,5.7] | 3.2| [4.1,11.6,5.7] |
| [5.4,29.9,9.4] | 5.3| [5.4,29.9,9.4] |
| [7.3,28.1,41.4] | 5.5| [7.3,28.1,41.4] |
| [7.8,38.9,50.6] | 6.6| [7.8,38.9,50.6] |
| [8.4,27.2,2.1] | 5.7| [8.4,27.2,2.1] |
+-----+-----+
only showing top 5 rows
```

7. 랜덤 포레스트 회귀 모형 적합하기

```
# 랜덤 포레스트 회귀 class 불러오기
from pyspark.ml.regression import RandomForestRegressor

# 랜덤 포레스트 회귀 알고리즘 정의하기
rf = RandomForestRegressor() # featuresCol="indexedFeatures", numTrees=2,
                           ↴maxDepth=2, seed=42
```

노트: indexedFeatures 특징을 사용하기로 결정한 경우, 파라미터 featuresCol="indexedFeatures"를 추가해야 합니다.

8. 파이프라인 아키텍쳐

```
# 파이프라인에서 인덱서와 체인 묶기
pipeline = Pipeline(stages=[featureIndexer, rf])
model = pipeline.fit(trainingData)
```

9. 예측값 만들기

```
predictions = model.transform(testData)

# 표시할 예제 행 선택하기
predictions.select("features", "label", "prediction").show(5)
```

```
+-----+-----+
| features|label| prediction|
+-----+-----+
| [4.1,11.6,5.7] | 3.2| 8.155439814814816|
| [5.4,29.9,9.4] | 5.3| 10.412769901394899|
| [7.3,28.1,41.4] | 5.5| 12.13735648148148|
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
| [7.8,38.9,50.6] | 6.6|11.321796703296704|
| [8.4,27.2,2.1] | 5.7|12.071421957671957|
+-----+-----+
only showing top 5 rows
```

10. 평가하기

```
# (예측, 실제 라벨) 을 선택하고 검정 오차를 계산하기
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
Root Mean Squared Error (RMSE) on test data = 2.35912
```

```
import sklearn.metrics
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {:.4f}'.format(r2_score))
```

```
r2_score: 0.831
```

11. 특징 중요도

```
model.stages[-1].featureImportances
```

```
SparseVector(3, {0: 0.4994, 1: 0.3196, 2: 0.181})
```

```
model.stages[-1].trees
```

```
[DecisionTreeRegressionModel (uid=dtr_c75f1c75442c) of depth 5 with 43 nodes,
DecisionTreeRegressionModel (uid=dtr_70fc2d441581) of depth 5 with 45 nodes,
DecisionTreeRegressionModel (uid=dtr_bc8464f545a7) of depth 5 with 31 nodes,
DecisionTreeRegressionModel (uid=dtr_a8a7e5367154) of depth 5 with 59 nodes,
DecisionTreeRegressionModel (uid=dtr_3ea01314fcbe) of depth 5 with 47 nodes,
DecisionTreeRegressionModel (uid=dtr_be9a04ac22a6) of depth 5 with 45 nodes,
DecisionTreeRegressionModel (uid=dtr_38610d47328a) of depth 5 with 51 nodes,
DecisionTreeRegressionModel (uid=dtr_bf14aea0ad3b) of depth 5 with 49 nodes,
DecisionTreeRegressionModel (uid=dtr_cde24ebd6bb6) of depth 5 with 39 nodes,
DecisionTreeRegressionModel (uid=dtr_a1fc9bd4fbef) of depth 5 with 57 nodes,
DecisionTreeRegressionModel (uid=dtr_37798d6db1ba) of depth 5 with 41 nodes,
DecisionTreeRegressionModel (uid=dtr_c078b73ada63) of depth 5 with 41 nodes,
DecisionTreeRegressionModel (uid=dtr_fd00e3a070ad) of depth 5 with 55 nodes,
DecisionTreeRegressionModel (uid=dtr_9d01d5fb8604) of depth 5 with 45 nodes,
DecisionTreeRegressionModel (uid=dtr_8bd8bdddf642) of depth 5 with 41 nodes,
DecisionTreeRegressionModel (uid=dtr_e53b7bae30f8) of depth 5 with 49 nodes,
DecisionTreeRegressionModel (uid=dtr_808a869db21c) of depth 5 with 47 nodes,
DecisionTreeRegressionModel (uid=dtr_64d0916bceb0) of depth 5 with 33 nodes,
DecisionTreeRegressionModel (uid=dtr_0891055fff94) of depth 5 with 55 nodes,
DecisionTreeRegressionModel (uid=dtr_19c8bbad26c2) of depth 5 with 51 nodes]
```

9.5 그레디언트 부스팅 트리 - 회귀

9.5.1 도입

9.5.2 어떻게 문제를 해결할까요?

9.5.3 예제

- 주피터 노트북은 [그레디언트 부스팅 회귀](#)에서 다운로드 할 수 있습니다.
 - 파라미터에 대한 자세한 내용은 [그레디언트 부스팅 트리 API](#)를 참조하시길 바랍니다.
- 스파크 컨텍스트 및 스파크세션 설정

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark GBTRRegressor example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. 데이터셋 로드하기

```
df = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', \
    inferSchema='true') \
    .load("../data/Advertising.csv", header=True);

df.show(5, True)
df.printSchema()
```

```
+----+----+----+----+
| TV | Radio | Newspaper | Sales |
+----+----+----+----+
| 230.1 | 37.8 | 69.2 | 22.1 |
| 44.5 | 39.3 | 45.1 | 10.4 |
| 17.2 | 45.9 | 69.3 | 9.3 |
| 151.5 | 41.3 | 58.5 | 18.5 |
| 180.8 | 10.8 | 58.4 | 12.9 |
+----+----+----+----+
only showing top 5 rows

root
 |-- TV: double (nullable = true)
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)
```

```
df.describe().show()
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

	TV	Radio	Newspaper	
summary				
Sales				
count	200	200	200	
mean	147.0425 23.26400000000024 30.553999999999995 14.			
stddev	85.85423631490805 14.846809176168728 21.77862083852283 5.			
min	0.7	0.0	0.3	
max	296.4	49.6	114.0	

3. 데이터를 밀집 벡터로 변환하기(특징과 라벨)

노트:

범주형 데이터를 처리하기 위해 get_dummy 함수를 사용해 보시길 적극 권장합니다.

지도 학습 버전:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols,
             labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer,
    OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_"
    .format(c))
        for c in categoricalCols ]

    # 기본 설정: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.
    getIndexer().getOutputCol(),
        outputCol="{0}_encoded".format(indexer.
    getIndexer().getOutputCol()))
        for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.
    getIndexer().getOutputCol() for encoder in encoders]
        + continuousCols, outputCol=
    "features")
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

    pipeline = Pipeline(stages=indexers + encoders +_
↪[assembler])

model=pipeline.fit(df)
data = model.transform(df)

data = data.withColumn('label', col(labelCol))

return data.select(indexCol,'features','label')

```

비지도학습 버전:

```

def get_dummy(df,indexCol,categoricalCols,continuousCols):
    '''
    더미 변수를 가져와 비지도 학습을 위해 연속 변수와 결합합니다.

    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터 이름 목록

    :반환 k: 특징 행렬

    :저자: Wenqiang Feng
    :이메일: von198@gmail.com
    '''

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}__"
↪indexed".format(c))
        for c in categoricalCols ]

    # 기본 설정: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.
↪getOutputCol(),
        outputCol="{0}_encoded".format(indexer.
↪getOutputCol()))
        for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.
↪getOutputCol() for encoder in encoders]
        + continuousCols, outputCol=
↪"features")

    pipeline = Pipeline(stages=indexers + encoders +_
↪[assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

return data.select(indexCol,'features')

```

두 가지 버전을 한 번에 적용하기:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols, labelCol,
    ↪dropLast=False):

    """
    더미 변수를 가져와 머신러닝 모델링을 위한 연속형 변수와 결합합니다.

    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터의 이름 목록
    :param labelCol: 라벨 열의 이름
    :param dropLast: 마지막 열을 버릴 것인지를 설정하는 옵션
    :반환: 특징 행렬

    :작자: Wenqiang Feng
    :이메일: von198@gmail.com

>>> df = spark.createDataFrame([
        (0, "a"),
        (1, "b"),
        (2, "c"),
        (3, "a"),
        (4, "a"),
        (5, "c")
    ], ["id", "category"])

>>> indexCol = 'id'
>>> categoricalCols = ['category']
>>> continuousCols = []
>>> labelCol = []

>>> mat = get_dummy(df, indexCol, categoricalCols, continuousCols,
    ↪labelCol)
>>> mat.show()

>>>
+---+-----+
| id|    features|
+---+-----+
|  0|[1.0,0.0,0.0]|
|  1|[0.0,0.0,1.0]|
|  2|[0.0,1.0,0.0]|
|  3|[1.0,0.0,0.0]|
|  4|[1.0,0.0,0.0]|
|  5|[0.0,1.0,0.0]|
+---+-----+
    """

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
    ↪VectorAssembler
from pyspark.sql.functions import col
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
    ↪format(c))
            for c in categoricalCols ]

# 기본 설정: dropLast=True
encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
    outputCol="{0}_encoded".format(indexer.
    ↪getOutputCol()), dropLast=dropLast)
            for indexer in indexers ]

assembler = VectorAssembler(inputCols=[encoder.getOutputCol()].
    ↪for encoder in encoders]
                    + continuousCols, outputCol="features"
    ↪" )

pipeline = Pipeline(stages=indexers + encoders + [assembler])

model=pipeline.fit(df)
data = model.transform(df)

if indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select(indexCol, 'features', 'label')
elif not indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select('features', 'label')
elif indexCol and not labelCol:
    # 비지도학습의 경우
    return data.select(indexCol, 'features')
elif not indexCol and not labelCol:
    # 비지도학습의 경우
    return data.select('features')

```

```

from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

# 데이터를 고밀도 벡터로 변환하기
def transData(row):
#     return Row(label=row["Sales"],
#                 features=Vectors.dense([row["TV"],
#                                         row["Radio"],
#                                         row["Newspaper"]]))
def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1]), r[-1]]).toDF([
        ↪'features', 'label'])

```

4. 데이터셋을 고밀도 벡터로 변환하기

```
transformed= transData(df)
transformed.show(5)
```

```
+-----+-----+
|       features|label|
+-----+-----+
| [230.1,37.8,69.2]| 22.1 |
| [44.5,39.3,45.1]| 10.4 |
| [17.2,45.9,69.3]|  9.3 |
| [151.5,41.3,58.5]| 18.5 |
| [180.8,10.8,58.4]| 12.9 |
+-----+-----+
only showing top 5 rows
```

5. 범주형 변수 처리하기

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import GBTRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

featureIndexer = VectorIndexer(inputCol="features", \
                                outputCol="indexedFeatures", \
                                maxCategories=4).fit(transformed)

data = featureIndexer.transform(transformed)
data.show(5, True)
```

```
+-----+-----+-----+
|       features|label| indexedFeatures|
+-----+-----+-----+
| [230.1,37.8,69.2]| 22.1|[230.1,37.8,69.2] |
| [44.5,39.3,45.1]| 10.4|[44.5,39.3,45.1] |
| [17.2,45.9,69.3]|  9.3|[17.2,45.9,69.3] |
| [151.5,41.3,58.5]| 18.5|[151.5,41.3,58.5] |
| [180.8,10.8,58.4]| 12.9|[180.8,10.8,58.4] |
+-----+-----+-----+
only showing top 5 rows
```

6. 데이터를 훈련 및 테스트 셋으로 분할하기(40%는 테스트 셋).

```
# 데이터를 훈련 및 테스트 셋으로 분할합니다(40%는 테스트 셋)
(trainingData, testData) = data.randomSplit([0.6, 0.4])

trainingData.show(5)
testData.show(5)
```

```
+-----+-----+-----+
|       features|label| indexedFeatures|
+-----+-----+-----+
| [0.7,39.6,8.7]| 1.6|[0.7,39.6,8.7] |
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
| [8.6,2.1,1.0] | 4.8| [8.6,2.1,1.0] |
| [8.7,48.9,75.0] | 7.2| [8.7,48.9,75.0] |
| [11.7,36.9,45.2] | 7.3| [11.7,36.9,45.2] |
| [13.2,15.9,49.6] | 5.6| [13.2,15.9,49.6] |
+-----+-----+
only showing top 5 rows

+-----+-----+
| features|label|indexedFeatures|
+-----+-----+
| [4.1,11.6,5.7] | 3.2| [4.1,11.6,5.7] |
| [5.4,29.9,9.4] | 5.3| [5.4,29.9,9.4] |
| [7.3,28.1,41.4] | 5.5|[7.3,28.1,41.4] |
| [7.8,38.9,50.6] | 6.6|[7.8,38.9,50.6] |
| [8.4,27.2,2.1] | 5.7| [8.4,27.2,2.1] |
+-----+-----+
only showing top 5 rows
```

7. 그레이디언트 부스팅 회귀 모형 적합하기

```
# 그레이디언트 부스팅 회귀 클래스 불러오기
from pyspark.ml.regression import GBTRegressor

# 그레이디언트 부스팅 회귀 알고리즘 정의하기
rf = GBTRegressor() #numTrees=2, maxDepth=2, seed=42
```

노트: 만약 여러분들께서 indexedFeatures 특징을 사용할 경우, 파라미터 featuresCol="indexedFeatures"를 추가해야 합니다.

8. 파일프라인 아키텍처

```
# 파일프라인에서 인덱서와 트리 룰기
pipeline = Pipeline(stages=[featureIndexer, rf])
model = pipeline.fit(trainingData)
```

9. 예측값 만들기

```
predictions = model.transform(testData)

# 표시할 예제 행 선택하기
predictions.select("features", "label", "prediction").show(5)
```

```
+-----+-----+
| features|label| prediction|
+-----+-----+
| [7.8,38.9,50.6] | 6.6| 6.836040343319862 |
| [8.6,2.1,1.0] | 4.8| 5.652202764688849 |
| [8.7,48.9,75.0] | 7.2| 6.908750296855572 |
| [13.1,0.4,25.6] | 5.3| 5.784020210692574 |
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
| [19.6,20.1,17.0] | 7.6|6.8678921062629295 |
+-----+-----+
only showing top 5 rows
```

10. 평가하기

```
# (예측, 실제 라벨) 을 선택하고 검정 오차를 계산하기
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
Root Mean Squared Error (RMSE) on test data = 1.36939
```

```
import sklearn.metrics
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {:.4f}'.format(r2_score))
```

```
r2_score: 0.932
```

11. 특징 중요도

```
model.stages[-1].featureImportances
```

```
SparseVector(3, {0: 0.3716, 1: 0.3525, 2: 0.2759})
```

```
model.stages[-1].trees
```

```
[DecisionTreeRegressionModel (uid=dtr_7f5cd2ef7cb6) of depth 5 with 61 nodes,
DecisionTreeRegressionModel (uid=dtr_ef3ab6baeac9) of depth 5 with 39 nodes,
DecisionTreeRegressionModel (uid=dtr_07c6e3cf3819) of depth 5 with 45 nodes,
DecisionTreeRegressionModel (uid=dtr_ce724af79a2b) of depth 5 with 47 nodes,
DecisionTreeRegressionModel (uid=dtr_d149ecc71658) of depth 5 with 55 nodes,
DecisionTreeRegressionModel (uid=dtr_d3a79bdea516) of depth 5 with 43 nodes,
DecisionTreeRegressionModel (uid=dtr_7abc1a337844) of depth 5 with 51 nodes,
DecisionTreeRegressionModel (uid=dtr_480834b46d8f) of depth 5 with 33 nodes,
DecisionTreeRegressionModel (uid=dtr_0cbd1eaaa3874) of depth 5 with 39 nodes,
DecisionTreeRegressionModel (uid=dtr_8088ac71a204) of depth 5 with 57 nodes,
DecisionTreeRegressionModel (uid=dtr_2ceb9e8deb45) of depth 5 with 47 nodes,
DecisionTreeRegressionModel (uid=dtr_cc334e84e9a2) of depth 5 with 57 nodes,
DecisionTreeRegressionModel (uid=dtr_a665c562929e) of depth 5 with 41 nodes,
DecisionTreeRegressionModel (uid=dtr_2999b1ffd2dc) of depth 5 with 45 nodes,
DecisionTreeRegressionModel (uid=dtr_29965cbe8cfc) of depth 5 with 55 nodes,
DecisionTreeRegressionModel (uid=dtr_731df51bf0ad) of depth 5 with 41 nodes,
DecisionTreeRegressionModel (uid=dtr_354cf33424da) of depth 5 with 51 nodes,
DecisionTreeRegressionModel (uid=dtr_4230f200b1c0) of depth 5 with 41 nodes,
DecisionTreeRegressionModel (uid=dtr_3279cdc1ce1d) of depth 5 with 45 nodes,
DecisionTreeRegressionModel (uid=dtr_f474a99ff06e) of depth 5 with 55 nodes]
```

정칙화

수학, 통계학, 컴퓨터 과학, 특히 기계학습과 역문제 분야에서 정칙화는 잘못된 문제를 해결하거나 과적합(Wikipedia Regularization)을 방지하기 위해 추가 정보를 도입하는 과정입니다.

데이터내 희소성으로 인해, 훈련 세트가 종종 타당치 않게 됩니다(특이). 회귀에 정칙화를 적용하면 다음과 같은 많은 이점이 있습니다:

1. 패널티 파라미터, λ 를 통한 추가정보를 덧붙임으로써 불량조건을 우량조건 문제로 전환
2. 과적합 방지
3. 변수 선택 및 상관 변수 제거(Glmnet Vignette). 리지 방법은 상관 변수의 계수를 축소하는 반면, LASSO 방법은 한 변수를 선택하고 선택되지 않은 다른 변수를 버립니다. 엘라스틱 넷 패널티는 이 두 가지가 혼합된 것입니다. 만약 변수들이 그룹들 안에 상관되어 있다면 $\alpha = 0.5$ 가 그룹들을 내부 또는 외부로 선택하는 경향이 있습니다. α 가 1에 가까울 경우 탄성망은 LASSO 방식과 매우 유사한 기능을 하며 극단적인 상관관계에 의한 거친 행동(wild behavior)과 의존적 성질을 제거합니다.

10.1 최소제곱법 회귀분석

$$\min_{\beta \in \mathbb{R}^n} \frac{1}{n} \|\mathbf{X}\beta - \mathbf{y}\|^2$$

$\lambda = 0$ (i.e. `regParam = 0`) 일때는, 패널티가 없습니다.

```
LinearRegression(featuresCol="features", labelCol="label", predictionCol=
  "prediction", maxIter=100,
  regParam=0.0, elasticNetParam=0.0, tol=1e-6, fitIntercept=True,
  standardization=True, solver="auto",
  weightCol=None, aggregationDepth=2)
```

10.2 리지 회귀분석

$$\min_{\beta \in \mathbb{R}^n} \frac{1}{n} \|\mathbf{X}\beta - \mathbf{y}\|^2 + \lambda \|\beta\|_2^2$$

$\lambda > 0$ (\Rightarrow , regParam > 0) 이며 $\alpha = 0$ (\Rightarrow , elasticNetParam = 0) 이면 L2 폐널티입니다.

```
LinearRegression(featuresCol="features", labelCol="label", predictionCol=
    "prediction", maxIter=100,
    regParam=0.1, elasticNetParam=0.0, tol=1e-6, fitIntercept=True,
    standardization=True, solver="auto",
    weightCol=None, aggregationDepth=2)
```

10.3 Least Absolute Shrinkage and Selection Operator (LASSO)

$$\min_{\beta \in \mathbb{R}^n} \frac{1}{n} \|\mathbf{X}\beta - \mathbf{y}\|^2 + \lambda \|\beta\|_1$$

$\lambda > 0$ (\Rightarrow , regParam > 0) 이며 $\alpha = 1$ (\Rightarrow , elasticNetParam = 1) 이면 L1 폐널티입니다

```
LinearRegression(featuresCol="features", labelCol="label", predictionCol=
    "prediction", maxIter=100,
    regParam=0.1, elasticNetParam=1.0, tol=1e-6, fitIntercept=True,
    standardization=True, solver="auto",
    weightCol=None, aggregationDepth=2)
```

10.4 엘라스틱 넷(Elastic net)

$$\min_{\beta \in \mathbb{R}^n} \frac{1}{n} \|\mathbf{X}\beta - \mathbf{y}\|^2 + \lambda(\alpha \|\beta\|_1 + (1 - \alpha) \|\beta\|_2^2), \alpha \in (0, 1)$$

$\lambda > 0$ (\Rightarrow , regParam > 0) 이며 elasticNetParam $\in (0, 1)$ (\Rightarrow , $\alpha \in (0, 1)$) 이면 L1 + L2 폐널티입니다.

```
LinearRegression(featuresCol="features", labelCol="label", predictionCol=
    "prediction", maxIter=100,
    regParam=0.1, elasticNetParam=0.5, tol=1e-6, fitIntercept=True,
    standardization=True, solver="auto",
    weightCol=None, aggregationDepth=2)
```

분류

중국 속담

사람은 끼리끼리 모이기 마련이다 – 중국의 옛 속담

11.1 이항 로지스틱 회귀분석

11.1.1 도입

11.1.2 데모

- 주피터 노트북은 [로지스틱 회귀분석](#)에서 다운로드 할 수 있습니다.
- 파라미터에 대한 자세한 내용은 [로지스틱 회귀분석 API](#)를 참조하시길 바랍니다.

노트: 저는 이번 데모에서 범주형 데이터를 다루기 위해 `get_dummy`라는 새로운 함수를 소개 할 것입니다. 여러분들께서 다른 분석 사례에서 저의 `get_dummy`함수를 사용해 보시길 적극 권장 합니다. 이 함수가 아마 여러분의 시간을 절약시켜 줄 것입니다.

1. 스파크 컨텍스트 및 스파크세션 설정

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark Logistic Regression example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. 데이터셋 로드하기

```
df = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', inferSchema='true') \
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
.load("./data/bank.csv", header=True);
df.drop('day', 'month', 'poutcome').show(5)
```

```
+---+-----+-----+-----+-----+-----+-----+-----+
| age |          | | | | | | | | | | |
| job | marital | education | default | balance | housing | loan | contact | duration | campaign | pdays | previous |
|   y |          |
+---+-----+-----+-----+-----+-----+-----+-----+
| 58 | management | married | tertiary | no | 2143 | yes | no | unknown |    |
| 261 |           1 | -1 | 0 | no |    |
| 44 | technician | single | secondary | no | 29 | yes | no | unknown |    |
| 151 |           1 | -1 | 0 | no |    |
| 33 | entrepreneur | married | secondary | no | 2 | yes | yes | unknown |    |
| 76 |           1 | -1 | 0 | no |    |
| 47 | blue-collar | married | unknown | no | 1506 | yes | no | unknown |    |
| 92 |           1 | -1 | 0 | no |    |
| 33 | unknown | single | unknown | no | 1 | no | no | unknown |    |
| 198 |           1 | -1 | 0 | no |    |
+---+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
df.printSchema()
```

```
root
| -- age: integer (nullable = true)
| -- job: string (nullable = true)
| -- marital: string (nullable = true)
| -- education: string (nullable = true)
| -- default: string (nullable = true)
| -- balance: integer (nullable = true)
| -- housing: string (nullable = true)
| -- loan: string (nullable = true)
| -- contact: string (nullable = true)
| -- day: integer (nullable = true)
| -- month: string (nullable = true)
| -- duration: integer (nullable = true)
| -- campaign: integer (nullable = true)
| -- pdays: integer (nullable = true)
| -- previous: integer (nullable = true)
| -- poutcome: string (nullable = true)
| -- y: string (nullable = true)
```

노트:

여러분들께서 복잡한 데이터 셋에서 범주형 데이터를 처리하기 위해 저의 get_dummy 함수를 사용해 보시길 적극 권장합니다.

지도 학습 버전:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols,
              labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer,
        OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_"
        .format(c))
        for c in categoricalCols ]

    # 기본 설정: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.
        getOutputCol(),
        outputCol="{0}_encoded".format(indexer.
        getOutputCol()))
        for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.
        getOutputCol() for encoder in encoders]
        + continuousCols, outputCol=
        "features")

    pipeline = Pipeline(stages=indexers + encoders +_
        [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label', col(labelCol))

    return data.select(indexCol,'features','label')
```

비지도학습 버전:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols):
    """
    디미 변수를 가져와 비지도 학습을 위해 연속 변수와 결합합니다.

    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터의 이름 목록

    :반환 k: 특징 행렬

    :저자: Wenqiang Feng
    :이메일: von198@gmail.com
    """

    ...
```

(다음 페이지에 계속)

(o] 전 페이지에서 계속)

```

indexers = [ StringIndexer(inputCol=c, outputCol="{0}_  
↳indexed".format(c))  
            for c in categoricalCols ]  
  
# 기본 설정: dropLast=True  
encoders = [ OneHotEncoder(inputCol=indexer.  
↳getOutputCol(),  
                           outputCol="{0}_encoded".format(indexer.  
↳getOutputCol()))  
            for indexer in indexers ]  
  
assembler = VectorAssembler(inputCols=[encoder.  
↳getOutputCol() for encoder in encoders]  
                            + continuousCols, outputCol=  
↳"features")  
  
pipeline = Pipeline(stages=indexers + encoders +  
↳[assembler])  
  
model=pipeline.fit(df)  
data = model.transform(df)  
  
return data.select(indexCol,'features')

```

두 가지 버전을 한 번에 적용하기:

```

def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol,  
↳dropLast=False):  
  
    ...  
    더미 변수를 가져와 머신러닝 모델링을 위한 연속형 변수와 결합합니다.  
  
    :param df: 데이터 프레임  
    :param categoricalCols: 범주형 데이터의 이름 목록  
    :param continuousCols: 수치형 데이터의 이름 목록  
    :param labelCol: 라벨 열의 이름  
    :param dropLast: 마지막 열을 버릴 것인지를 설정하는 옵션  
    :반환: 특징 행렬  
  
    :저자: Wenqiang Feng  
    :이메일: von198@gmail.com  
  
>>> df = spark.createDataFrame([  
        (0, "a"),  
        (1, "b"),  
        (2, "c"),  
        (3, "a"),  
        (4, "a"),  
        (5, "c")  
    ], ["id", "category"])

```

>>> indexCol = 'id'

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

>>> categoricalCols = ['category']
>>> continuousCols = []
>>> labelCol = []

>>> mat = get_dummy(df, indexCol, categoricalCols, continuousCols,
↪labelCol)
>>> mat.show()

>>>
+---+-----+
| id|    features|
+---+-----+
|  0|[1.0,0.0,0.0]|
|  1|[0.0,0.0,1.0]|
|  2|[0.0,1.0,0.0]|
|  3|[1.0,0.0,0.0]|
|  4|[1.0,0.0,0.0]|
|  5|[0.0,1.0,0.0]|
+---+-----+
```

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
↪VectorAssembler
from pyspark.sql.functions import col

indexers = [StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
 for c in categoricalCols]

기본 설정: dropLast=True
encoders = [OneHotEncoder(inputCol=indexer.getOutputCol(),
 outputCol="{0}_encoded".format(indexer.
↪getOutputCol()), dropLast=dropLast)
 for indexer in indexers]

assembler = VectorAssembler(inputCols=[encoder.getOutputCol()].
↪for encoder in encoders]
 + continuousCols, outputCol="features"
↪")

pipeline = Pipeline(stages=indexers + encoders + [assembler])

model=pipeline.fit(df)
data = model.transform(df)

if indexCol and labelCol:
 # 지도 학습의 경우
 data = data.withColumn('label', col(labelCol))
 return data.select(indexCol, 'features', 'label')
elif not indexCol and labelCol:
 # 지도 학습의 경우

```

(다음 페이지에서 계속)

(이전 페이지에서 계속)

```

data = data.withColumn('label', col(labelCol))
return data.select('features', 'label')
elif indexCol and not labelCol:
 # 빠지도 학습의 경우
 return data.select(indexCol, 'features')
elif not indexCol and not labelCol:
 # 빠지도 학습의 경우
 return data.select('features')

```

```

def get_dummy(df, categoricalCols, continuousCols, labelCol):

 from pyspark.ml import Pipeline
 from pyspark.ml.feature import StringIndexer, OneHotEncoder,
 ↪ VectorAssembler
 from pyspark.sql.functions import col

 indexers = [StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
 for c in categoricalCols]

 # 기본 설정: dropLast=True
 encoders = [OneHotEncoder(inputCol=indexer.getOutputCol(),
 outputCol="{0}_encoded".format(indexer.getOutputCol()))
 for indexer in indexers]

 assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder
 ↪ in encoders]
 + continuousCols, outputCol="features")

 pipeline = Pipeline(stages=indexers + encoders + [assembler])

 model=pipeline.fit(df)
 data = model.transform(df)

 data = data.withColumn('label', col(labelCol))

 return data.select('features', 'label')

```

### 3. 범주형 데이터를 처리하고 데이터를 고밀도 벡터로 변환하기

```

catcols = ['job', 'marital', 'education', 'default',
 'housing', 'loan', 'contact', 'poutcome']

num_cols = ['balance', 'duration', 'campaign', 'pdays', 'previous',]
labelCol = 'y'

data = get_dummy(df, catcols, num_cols, labelCol)
data.show(5)

```

|  | features | label |
|--|----------|-------|
|--|----------|-------|

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
+-----+-----+
(29, [1,11,14,16,1...	no
(29, [2,12,13,16,1...	no
(29, [7,11,13,16,1...	no
(29, [0,11,16,17,1...	no
(29, [12,16,18,20,...	no
+-----+-----+
only showing top 5 rows
```

#### 4. 범주형 라벨과 변수들 처리하기

```
from pyspark.ml.feature import StringIndexer
라벨 색인을 만들고 메타데이터에 라벨 컬럼을 추가합니다
labelIndexer = StringIndexer(inputCol='label',
 outputCol='indexedLabel').fit(data)
labelIndexer.transform(data).show(5, True)
```

```
+-----+-----+-----+
| features|label|indexedLabel|
+-----+-----+-----+
(29, [1,11,14,16,1...	no	0.0
(29, [2,12,13,16,1...	no	0.0
(29, [7,11,13,16,1...	no	0.0
(29, [0,11,16,17,1...	no	0.0
(29, [12,16,18,20,...	no	0.0
+-----+-----+-----+
only showing top 5 rows
```

```
from pyspark.ml.feature import VectorIndexer
범주형 특징을 자동으로 식별하고 인덱싱합니다.
maxCategories를 지정하여 4개 이상의 고유 값을 가진 특징이 연속형으로 처리되도록 합니다.

featureIndexer = VectorIndexer(inputCol="features", \
 outputCol="indexedFeatures", \
 maxCategories=4).fit(data)
featureIndexer.transform(data).show(5, True)
```

```
+-----+-----+-----+
| features|label| indexedFeatures|
+-----+-----+-----+
(29, [1,11,14,16,1...	no	(29, [1,11,14,16,1...
(29, [2,12,13,16,1...	no	(29, [2,12,13,16,1...
(29, [7,11,13,16,1...	no	(29, [7,11,13,16,1...
(29, [0,11,16,17,1...	no	(29, [0,11,16,17,1...
(29, [12,16,18,20,...	no	(29, [12,16,18,20,...
+-----+-----+-----+
only showing top 5 rows
```

#### 5. 데이터를 훈련 및 테스트 셋으로 분할하기

```
데이터를 훈련 및 테스트 셋으로 분할합니다(40%는 테스트 셋)
(trainingData, testData) = data.randomSplit([0.6, 0.4])

trainingData.show(5, False)
testData.show(5, False)
```

```
+-----+-----+
| features | label |
+-----+-----+
| (29, [0,11,13,16,17,18,19,21,24,25,26,27], [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
| 731.0,401.0,4.0,-1.0]) | no |
| (29, [0,11,13,16,17,18,19,21,24,25,26,27], [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
| 723.0,112.0,2.0,-1.0]) | no |
| (29, [0,11,13,16,17,18,19,21,24,25,26,27], [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
| 626.0,205.0,1.0,-1.0]) | no |
| (29, [0,11,13,16,17,18,19,21,24,25,26,27], [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
| 498.0,357.0,1.0,-1.0]) | no |
| (29, [0,11,13,16,17,18,19,21,24,25,26,27], [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
| 477.0,473.0,2.0,-1.0]) | no |
+-----+-----+
only showing top 5 rows

+-----+-----+
| features | label |
+-----+-----+
| (29, [0,11,13,16,17,18,19,21,24,25,26,27], [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
| 648.0,280.0,2.0,-1.0]) | no |
| (29, [0,11,13,16,17,18,19,21,24,25,26,27], [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
| 596.0,147.0,1.0,-1.0]) | no |
| (29, [0,11,13,16,17,18,19,21,24,25,26,27], [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
| 529.0,416.0,4.0,-1.0]) | no |
| (29, [0,11,13,16,17,18,19,21,24,25,26,27], [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
| 518.0,46.0,5.0,-1.0]) | no |
| (29, [0,11,13,16,17,18,19,21,24,25,26,27], [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,-
| 470.0,275.0,2.0,-1.0]) | no |
+-----+-----+
only showing top 5 rows
```

### 6. 로지스틱 회귀분석 모형 적합하기

```
from pyspark.ml.classification import LogisticRegression
logr = LogisticRegression(featuresCol='indexedFeatures', labelCol=
 'indexedLabel')
```

### 7. 파이프라인 아키텍처

```
인덱싱된 라벨들을 본래 라벨들로 변환하기
from pyspark.ml.feature import IndexToString
labelConverter = IndexToString(inputCol="prediction", outputCol=
→"predictedLabel", labels=labelIndexer.labels)
```

```
파이프라인에 인덱서와 체인 묶기
from pyspark.ml import Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, logr,
→labelConverter])
```

```
모형 훈련하기. 인덱서들도 같이 실행됩니다.
model = pipeline.fit(trainingData)
```

## 8. 예측값 만들기

```
예측값 만들기
predictions = model.transform(testData)
표시할 예제 행 선택하기
predictions.select("features", "label", "predictedLabel").show(5)
```

```
+-----+-----+-----+
| features|label|predictedLabel|
+-----+-----+-----+
| (29, [0,11,13,16,1...| no | no |
+-----+-----+-----+
only showing top 5 rows
```

## 9. 평가하기

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

(예측, 실제 라벨) 을 선택하고 검정 오차 계산하기
evaluator = MulticlassClassificationEvaluator(
 labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy"
→")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))
```

```
Test Error = 0.0987688
```

```
lrModel = model.stages[2]
trainingSummary = lrModel.summary

매 반복에서 목적함수 구하기
objectiveHistory = trainingSummary.objectiveHistory
print("objectiveHistory:")
for objective in objectiveHistory:
```

(다음 페이지에서 계속)

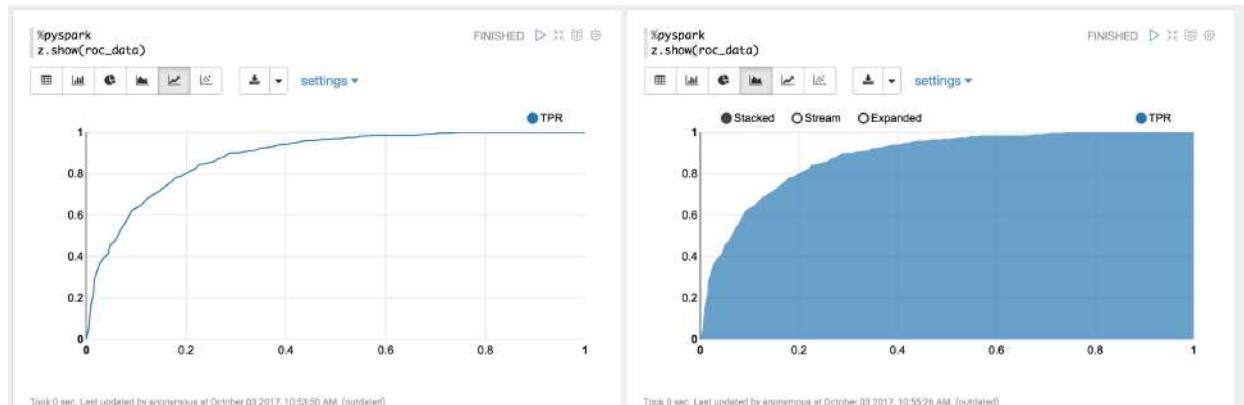
(이전 페이지에서 계속)

```
print(objective)

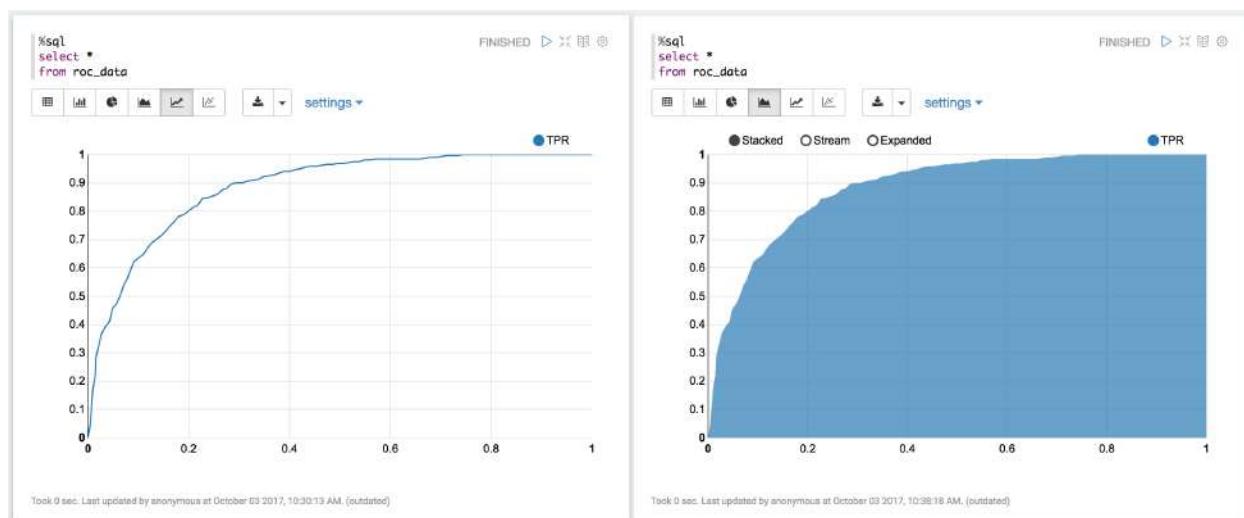
데이터 프레임과 areaUnderROC로 ROC 구하기
trainingSummary.roc.show(5)
print("areaUnderROC: " + str(trainingSummary.areaUnderROC))

F-척도를 최대화하는 임곗값을 모형으로 설정하기
fMeasure = trainingSummary.fMeasureByThreshold
maxFMeasure = fMeasure.groupBy().max('F-Measure').select('max(F-Measure)').head(5)
bestThreshold = fMeasure.where(fMeasure['F-Measure'] == maxFMeasure['max(F-
Measure)']) \
.select('threshold').head()['threshold']
lr.setThreshold(bestThreshold)
```

여러분은 z.show() 를 사용해 ROC 곡선들과 데이터를 얻을 수 있습니다:



또한 임시 테이블 data.registerTempTable('roc\_data') 을 생성하고 sql 로 ROC 곡선을 그릴 수 있습니다:



## 10. 시각화하기

```

import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm, classes,
 normalize=False,
 title='Confusion matrix',
 cmap=plt.cm.Blues):

 """
 이 함수는 혼동 행렬을 출력하고 그립니다. 정규화는 'normalize=True'를
 설정하여 적용할 수 있습니다.
 """

 if normalize:
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
 print("Normalized confusion matrix")
 else:
 print('Confusion matrix, without normalization')

 print(cm)

 plt.imshow(cm, interpolation='nearest', cmap=cmap)
 plt.title(title)
 plt.colorbar()
 tick_marks = np.arange(len(classes))
 plt.xticks(tick_marks, classes, rotation=45)
 plt.yticks(tick_marks, classes)

 fmt = '.2f' if normalize else 'd'
 thresh = cm.max() / 2.
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
 plt.text(j, i, format(cm[i, j], fmt),
 horizontalalignment="center",
 color="white" if cm[i, j] > thresh else "black")

 plt.tight_layout()
 plt.ylabel('True label')
 plt.xlabel('Predicted label')

```

```

class_temp = predictions.select("label").groupBy("label") \
 .count().sort('count', ascending=False).toPandas()
class_temp = class_temp["label"].values.tolist()
class_names = map(str, class_temp)
print(class_name
class_names

```

```
['no', 'yes']
```

```

from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

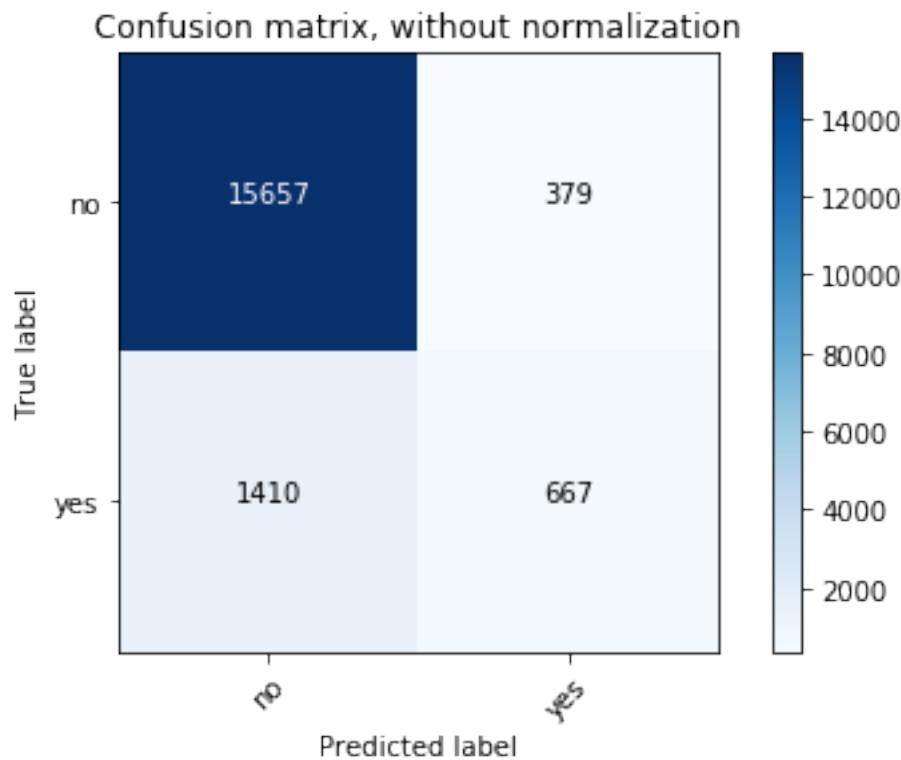
```
y_pred = predictions.select("predictedLabel")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred, labels=class_names)
cnf_matrix
```

```
array([[15657, 379],
 [1410, 667]])
```

```
비정규화된 혼동 행렬 그리기
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
 title='Confusion matrix, without normalization')
plt.show()
```

```
Confusion matrix, without normalization
[[15657 379]
 [1410 667]]
```



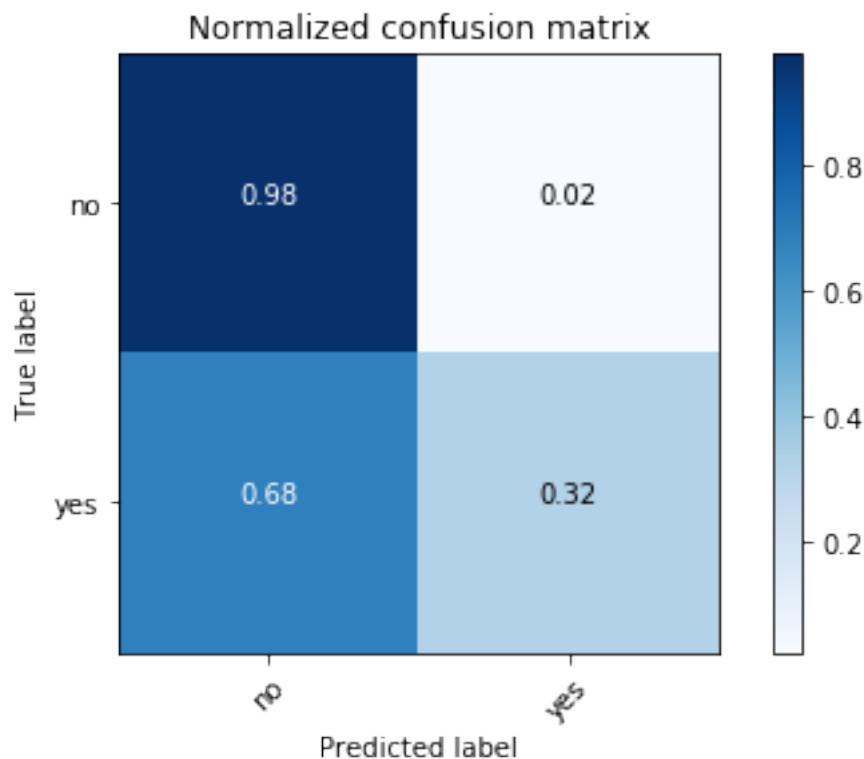
```
정규화된 혼동 행렬 그리기
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
 title='Normalized confusion matrix')
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
plt.show()
```

```
Normalized confusion matrix
[[0.97636568 0.02363432]
 [0.67886375 0.32113625]]
```



## 11.2 다항 로지스틱 회귀분석

### 11.2.1 도입

### 11.2.2 테모

- 주피터 노트북은 [로지스틱 회귀분석](#)에서 다운로드할 수 있습니다
- 파라미터에 대한 자세한 내용은 [로지스틱 회귀분석 API](#)를 참조하시길 바랍니다.

**노트:** 저는 이번 테모에서 범주형 데이터를 다루기 위해 저의 `get_dummy` 함수를 사용할 것입니다. 여러분께서 다른 분석 사례에서 저의 `get_dummy` 함수를 사용해 보시길 적극 권장합니다. 이 함수가 아마 여러분의 시간을 절약시켜 줄 것입니다.

#### 1. 스파크 컨텍스트 및 스파크세션 설정

```
from pyspark.sql import SparkSession

spark = SparkSession \
 .builder \
 .appName("Python Spark MultinomialLogisticRegression classification") \
 .config("spark.some.config.option", "some-value") \
 .getOrCreate()
```

### 2. 데이터셋 로드하기

```
df = spark.read.format('com.databricks.spark.csv') \
 .options(header='true', inferSchema='true') \
 .load("./data/WineData2.csv", header=True);
df.show(5)
```

```
+---+-----+-----+-----+-----+-----+-----+-----+-----+
|fixed|volatile|citric|sugar|chlorides|free|total|density|
|pH|sulphates|alcohol|quality|
+---+-----+-----+-----+-----+-----+-----+-----+
7.4	0.7	0.0	1.9	0.076	11.0	34.0	0.9978	3.51	0.56
9.4	5								
7.8	0.88	0.0	2.6	0.098	25.0	67.0	0.9968	3.2	0.68
9.8	5								
7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.997	3.26	0.65
9.8	5								
11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.998	3.16	0.58
9.8	6								
7.4	0.7	0.0	1.9	0.076	11.0	34.0	0.9978	3.51	0.56
9.4	5								
+---+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
df.printSchema()
```

```
root
|-- fixed: double (nullable = true)
|-- volatile: double (nullable = true)
|-- citric: double (nullable = true)
|-- sugar: double (nullable = true)
|-- chlorides: double (nullable = true)
|-- free: double (nullable = true)
|-- total: double (nullable = true)
|-- density: double (nullable = true)
|-- pH: double (nullable = true)
|-- sulphates: double (nullable = true)
|-- alcohol: double (nullable = true)
|-- quality: string (nullable = true)
```

```
float 형식으로 변환하기
def string_to_float(x):
 return float(x)

#
def condition(r):
 if (0 <= r <= 4):
 label = "low"
 elif(4 < r <= 6):
 label = "medium"
 else:
 label = "high"
 return label

from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType
string_to_float_udf = udf(string_to_float, DoubleType())
quality_udf = udf(lambda x: condition(x), StringType())

df = df.withColumn("quality", quality_udf("quality"))

df.show(5, True)
```

| fixed | volatile | citric | sugar | chlorides | free | total | density | pH   | sulphates | alcohol | quality |
|-------|----------|--------|-------|-----------|------|-------|---------|------|-----------|---------|---------|
| 7.4   | 0.7      | 0.0    | 1.9   | 0.076     | 11.0 | 34.0  | 0.9978  | 3.51 |           | 0.56    | medium  |
| 9.4   | 0.88     | 0.0    | 2.6   | 0.098     | 25.0 | 67.0  | 0.9968  | 3.2  |           | 0.68    | medium  |
| 7.8   | 0.76     | 0.04   | 2.3   | 0.092     | 15.0 | 54.0  | 0.997   | 3.26 |           | 0.65    | medium  |
| 11.2  | 0.28     | 0.56   | 1.9   | 0.075     | 17.0 | 60.0  | 0.998   | 3.16 |           | 0.58    | medium  |
| 7.4   | 0.7      | 0.0    | 1.9   | 0.076     | 11.0 | 34.0  | 0.9978  | 3.51 |           | 0.56    | medium  |

only showing top 5 rows

```
df.printSchema()
```

```
root
|-- fixed: double (nullable = true)
|-- volatile: double (nullable = true)
|-- citric: double (nullable = true)
|-- sugar: double (nullable = true)
|-- chlorides: double (nullable = true)
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
|-- free: double (nullable = true)
|-- total: double (nullable = true)
|-- density: double (nullable = true)
|-- pH: double (nullable = true)
|-- sulphates: double (nullable = true)
|-- alcohol: double (nullable = true)
|-- quality: string (nullable = true)
```

### 3. 범주형 데이터를 처리하고 데이터를 고밀도 벡터로 변환하기

#### 노트:

여러분들께서 복잡한 데이터 셋에서 범주형 데이터를 처리하기 위해 저의 get\_dummy 함수를 사용해 보시길 적극 권장합니다.

지도 학습 버전:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols,
 labelCol):

 from pyspark.ml import Pipeline
 from pyspark.ml.feature import StringIndexer,
 OneHotEncoder, VectorAssembler
 from pyspark.sql.functions import col

 indexers = [StringIndexer(inputCol=c, outputCol="{0}_"
 .format(c))
 for c in categoricalCols]

 # 기본 설정: dropLast=True
 encoders = [OneHotEncoder(inputCol=indexer.
 .getOutputCol(),
 outputCol="{0}_encoded".format(indexer.
 .getOutputCol()))
 for indexer in indexers]

 assembler = VectorAssembler(inputCols=[encoder.
 .getOutputCol() for encoder in encoders]
 + continuousCols, outputCol=
 "features")

 pipeline = Pipeline(stages=indexers + encoders +
 [assembler])

 model=pipeline.fit(df)
 data = model.transform(df)

 data = data.withColumn('label', col(labelCol))

 return data.select(indexCol, 'features', 'label')
```

비지도학습 버전:

```

def get_dummy(df, indexCol, categoricalCols, continuousCols):
 """
 더미 변수를 가져와 비지도 학습을 위해 연속 변수와 결합합니다.

 :param df: 데이터 프레임
 :param categoricalCols: 범주형 데이터의 이름 목록
 :param continuousCols: 수치형 데이터의 이름 목록
 """

 :반환 k: 특정 행렬

 :저자: Wenqiang Feng
 :이메일: von198@gmail.c
 """

 indexers = [StringIndexer(inputCol=c, outputCol="{0}__indexed".format(c))
 for c in categoricalCols]

 # 기본 설정: dropLast=True
 encoders = [OneHotEncoder(inputCol=indexer.
 getOutputCol(),
 outputCol="{0}_encoded".format(indexer.
 getOutputCol()))
 for indexer in indexers]

 assembler = VectorAssembler(inputCols=[encoder.
 getOutputCol() for encoder in encoders]
 + continuousCols, outputCol=
 "features")

 pipeline = Pipeline(stages=indexers + encoders +_
 [assembler])

 model=pipeline.fit(df)
 data = model.transform(df)

 return data.select(indexCol, 'features')

```

두 가지 버전을 한 번에 적용하기:

```

def get_dummy(df, indexCol, categoricalCols, continuousCols, labelCol,
 dropLast=False):

 """
 더미 변수를 가져와 머신러닝 모델링을 위한 연속형 변수와 결합합니다.

 :param df: 데이터 프레임
 :param categoricalCols: 범주형 데이터의 이름 목록
 :param continuousCols: 수치형 데이터의 이름 목록
 :param labelCol: 라벨 열의 이름
 :param dropLast: 마지막 열을 버릴 것인지를 설정하는 옵션
 """

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

:반환: 특징 행렬

:저자: Wenqiang Feng
:이메일: von198@gmail.com

>>> df = spark.createDataFrame([
 (0, "a"),
 (1, "b"),
 (2, "c"),
 (3, "a"),
 (4, "a"),
 (5, "c")
], ["id", "category"])

>>> indexCol = 'id'
>>> categoricalCols = ['category']
>>> continuousCols = []
>>> labelCol = []

>>> mat = get_dummy(df, indexCol, categoricalCols, continuousCols,
↪labelCol)
>>> mat.show()

>>>
+---+-----+
| id| features|
+---+-----+
0	[1.0,0.0,0.0]
1	[0.0,0.0,1.0]
2	[0.0,1.0,0.0]
3	[1.0,0.0,0.0]
4	[1.0,0.0,0.0]
5	[0.0,1.0,0.0]
+---+-----+
```
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
↪VectorAssembler
from pyspark.sql.functions import col

indexers = [StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c)) for c in categoricalCols]

# 기본 설정: dropLast=True
encoders = [OneHotEncoder(inputCol=indexer.getOutputCol(),
                           outputCol="{0}_encoded".format(indexer.
↪getOutputCol()), dropLast=dropLast)
            for indexer in indexers]
assembler = VectorAssembler(inputCols=[encoder.getOutputCol()
                                       for encoder in encoders]
                           )

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

    + continuousCols, outputCol="features"
    ↵" )

pipeline = Pipeline(stages=indexers + encoders + [assembler])

model=pipeline.fit(df)
data = model.transform(df)

if indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select(indexCol,'features','label')
elif not indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select('features','label')
elif indexCol and not labelCol:
    # 비지도학습의 경우
    return data.select(indexCol,'features')
elif not indexCol and not labelCol:
    # 비지도학습의 경우
    return data.select('features')

```

```

def get_dummy(df,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder,
    ↵VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    #기본 설정: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                                outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder
    ↵in encoders]
                                 + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label', col(labelCol))

    return data.select('features','label')

```

4. 데이터 셋을 데이터프레임으로 변환하기

```
from pyspark.ml.linalg import Vectors
# !!!!주의: from pyspark.mllib.linalg import Vectors가 아닙니다.
from pyspark.ml import Pipeline
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1]), r[-1]]).toDF(['features',
    'label'])
```

```
transformed = transData(df)
transformed.show(5)
```

```
+-----+-----+
|      features| label|
+-----+-----+
| [7.4, 0.7, 0.0, 1.9, ... |medium|
| [7.8, 0.88, 0.0, 2.6... |medium|
| [7.8, 0.76, 0.04, 2.... |medium|
| [11.2, 0.28, 0.56, 1... |medium|
| [7.4, 0.7, 0.0, 1.9, ... |medium|
+-----+-----+
only showing top 5 rows
```

4. 범주형 라벨과 변수들 처리하기

```
# 라벨을 색인화하고 메타데이터에 라벨 열 추가하기
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(transformed)
labelIndexer.transform(transformed).show(5, True)
```

```
+-----+-----+-----+
|      features| label|indexedLabel|
+-----+-----+-----+
| [7.4, 0.7, 0.0, 1.9, ... |medium|          0.0|
| [7.8, 0.88, 0.0, 2.6... |medium|          0.0|
| [7.8, 0.76, 0.04, 2.... |medium|          0.0|
| [11.2, 0.28, 0.56, 1... |medium|          0.0|
| [7.4, 0.7, 0.0, 1.9, ... |medium|          0.0|
+-----+-----+-----+
only showing top 5 rows
```

```
# 범주형 특징을 자동으로 식별하고 인덱싱합니다.
# maxCategories를 지정하여 4개 이상의 고유 값을 가진 특징이 연속형으로 처리되도록 합니다.
```

```
featureIndexer = VectorIndexer(inputCol="features", \
                                outputCol="indexedFeatures", \
                                maxCategories=4).fit(transformed)
featureIndexer.transform(transformed).show(5, True)
```

```
+-----+-----+
|       features | label |      indexedFeatures |
+-----+-----+
| [7.4, 0.7, 0.0, 1.9, ... | medium | [7.4, 0.7, 0.0, 1.9, ... |
| [7.8, 0.88, 0.0, 2.6, ... | medium | [7.8, 0.88, 0.0, 2.6, ... |
| [7.8, 0.76, 0.04, 2, ... | medium | [7.8, 0.76, 0.04, 2, ... |
| [11.2, 0.28, 0.56, 1, ... | medium | [11.2, 0.28, 0.56, 1, ... |
| [7.4, 0.7, 0.0, 1.9, ... | medium | [7.4, 0.7, 0.0, 1.9, ...
+-----+-----+
only showing top 5 rows
```

5. 데이터를 훈련 및 테스트 셋으로 분할하기

```
# 데이터를 훈련 및 테스트 셋으로 분할합니다(40%는 테스트 셋)
(trainingData, testData) = data.randomSplit([0.6, 0.4])

trainingData.show(5, False)
testData.show(5, False)
```

```
+-----+-----+
| features | label |
+-----+-----+
| [4.7, 0.6, 0.17, 2.3, 0.058, 17.0, 106.0, 0.9932, 3.85, 0.6, 12.9] | medium |
| [5.0, 0.38, 0.01, 1.6, 0.048, 26.0, 60.0, 0.99084, 3.7, 0.75, 14.0] | medium |
| [5.0, 0.4, 0.5, 4.3, 0.046, 29.0, 80.0, 0.9902, 3.49, 0.66, 13.6] | medium |
| [5.0, 0.74, 0.0, 1.2, 0.041, 16.0, 46.0, 0.99258, 4.01, 0.59, 12.5] | medium |
| [5.1, 0.42, 0.0, 1.8, 0.044, 18.0, 88.0, 0.99157, 3.68, 0.73, 13.6] | high |
+-----+-----+
only showing top 5 rows

+-----+-----+
| features | label |
+-----+-----+
| [4.6, 0.52, 0.15, 2.1, 0.054, 8.0, 65.0, 0.9934, 3.9, 0.56, 13.1] | low |
| [4.9, 0.42, 0.0, 2.1, 0.048, 16.0, 42.0, 0.99154, 3.71, 0.74, 14.0] | high |
| [5.0, 0.42, 0.24, 2.0, 0.06, 19.0, 50.0, 0.9917, 3.72, 0.74, 14.0] | high |
| [5.0, 1.02, 0.04, 1.4, 0.045, 41.0, 85.0, 0.9938, 3.75, 0.48, 10.5] | low |
| [5.0, 1.04, 0.24, 1.6, 0.05, 32.0, 96.0, 0.9934, 3.74, 0.62, 11.5] | medium |
+-----+-----+
only showing top 5 rows
```

6. 다항 로지스틱 회귀분석 분류 모형 적합하기

```
from pyspark.ml.classification import LogisticRegression
logr = LogisticRegression(featuresCol='indexedFeatures', labelCol=
    'indexedLabel')
```

7. 파이프라인 아키텍처

```
# 인덱싱된 라벨들을 본래 라벨들로 변환하기
labelConverter = IndexToString(inputCol="prediction", outputCol=
    "predictedLabel",
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
labels=labelIndexer.labels)
```

```
# 파일에 인덱서와 트리 뮤기
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, logr,
                           labelConverter])
```

```
# 모형을 훈련하기. 인덱서들도 같이 실행됩니다.
model = pipeline.fit(trainingData)
```

8. 예측값 만들기

```
# 예측값 만들기
predictions = model.transform(testData)
# 표시할 예제 행 선택하기
predictions.select("features", "label", "predictedLabel").show(5)
```

```
+-----+-----+-----+
|       features| label|predictedLabel|
+-----+-----+-----+
| [4.6, 0.52, 0.15, 2....|   low|      medium|
| [4.9, 0.42, 0.0, 2.1....|   high|      high|
| [5.0, 0.42, 0.24, 2....|   high|      high|
| [5.0, 1.02, 0.04, 1....|   low|      medium|
| [5.0, 1.04, 0.24, 1....|medium|      medium|
+-----+-----+-----+
only showing top 5 rows
```

9. 평가하기

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# (예측, 실제 라벨) 을 선택하고 검정 오차 계산하기
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))
```

```
Test Error = 0.181287
```

```
lrModel = model.stages[2]
trainingSummary = lrModel.summary

# 매 반복에서 목표 함수 구하기
# objectiveHistory = trainingSummary.objectiveHistory
# print("objectiveHistory:")
# for objective in objectiveHistory:
#     print(objective)
```

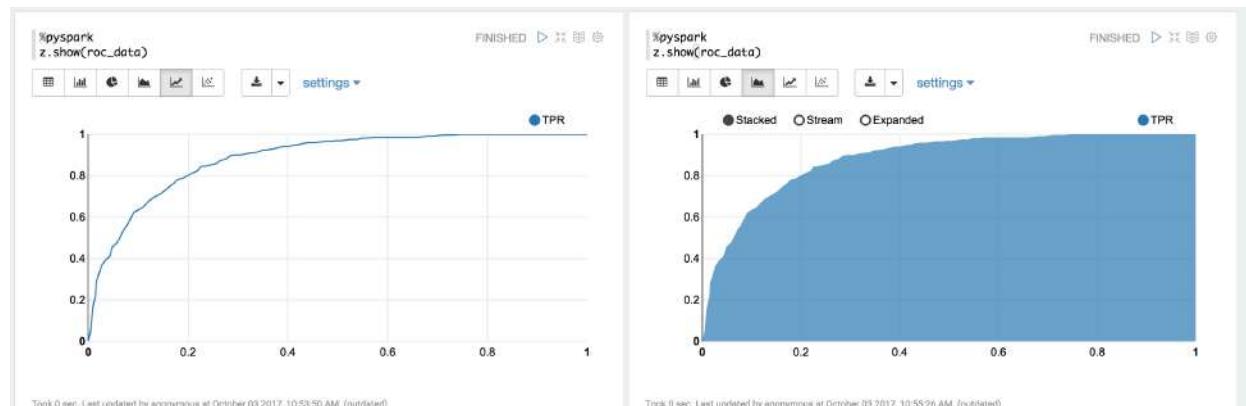
(다음 페이지에 계속)

(이전 페이지에서 계속)

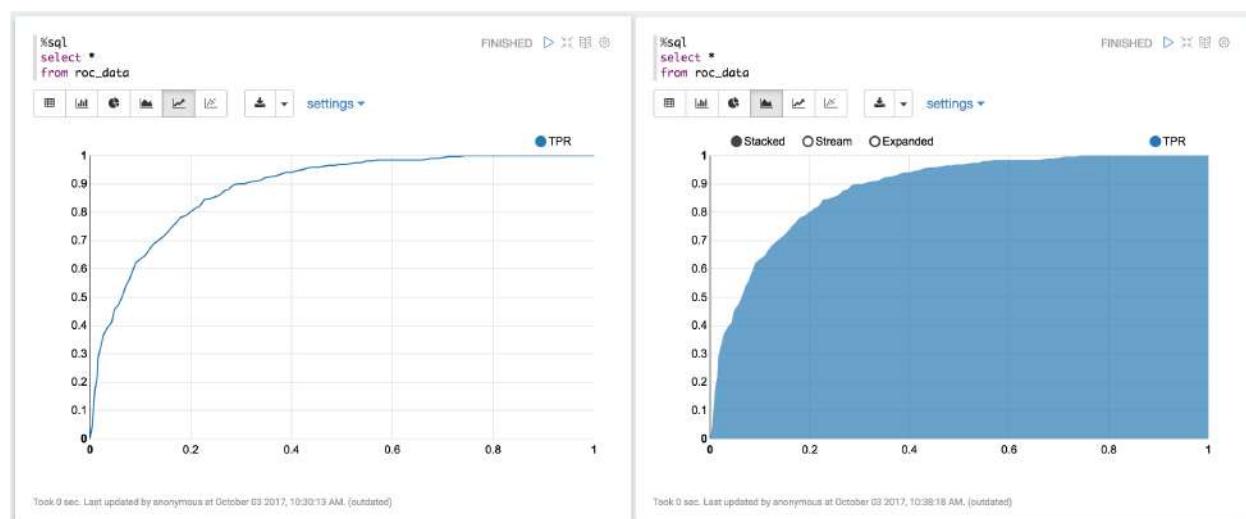
```
# 데이터 프레임과 areaUnderROC로 ROC 구하기
trainingSummary.roc.show(5)
print("areaUnderROC: " + str(trainingSummary.areaUnderROC))

# F-척도를 최대화하는 임곗값을 모형으로 설정하기
fMeasure = trainingSummary.fMeasureByThreshold
maxFMeasure = fMeasure.groupBy().max('F-Measure').select('max(F-Measure)').head(5)
# bestThreshold = fMeasure.where(fMeasure['F-Measure'] == maxFMeasure['max(F-Measure)']) \
#     .select('threshold').head()['threshold']
# lr.setThreshold(bestThreshold)
```

여러분은 `z.show()` 를 사용해 ROC 곡선들과 데이터를 얻을 수 있습니다:



또한 임시 테이블 `data.registerTempTable('roc_data')` 을 생성하고 sql로 ROC 곡선을 그릴 수 있습니다:



10. 시각화하기

11.2. 다항 로지스틱 회귀분석

```

import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm, classes,
                         normalize=False,
                         title='Confusion matrix',
                         cmap=plt.cm.Blues):

    """
    이 함수는 혼동 행렬을 출력하고 그립니다. 정규화는 'normalize=True'를
    설정하여 적용할 수 있습니다.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

```

```

class_temp = predictions.select("label").groupBy("label") \
    .count().sort('count', ascending=False).toPandas()
class_temp = class_temp["label"].values.tolist()
class_names = map(str, class_temp)
# # # print(class_name)
class_names

```

```
[ 'medium', 'high', 'low' ]
```

```

from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()

```

(다음 페이지에 계속)

(이전 페이지에 계속)

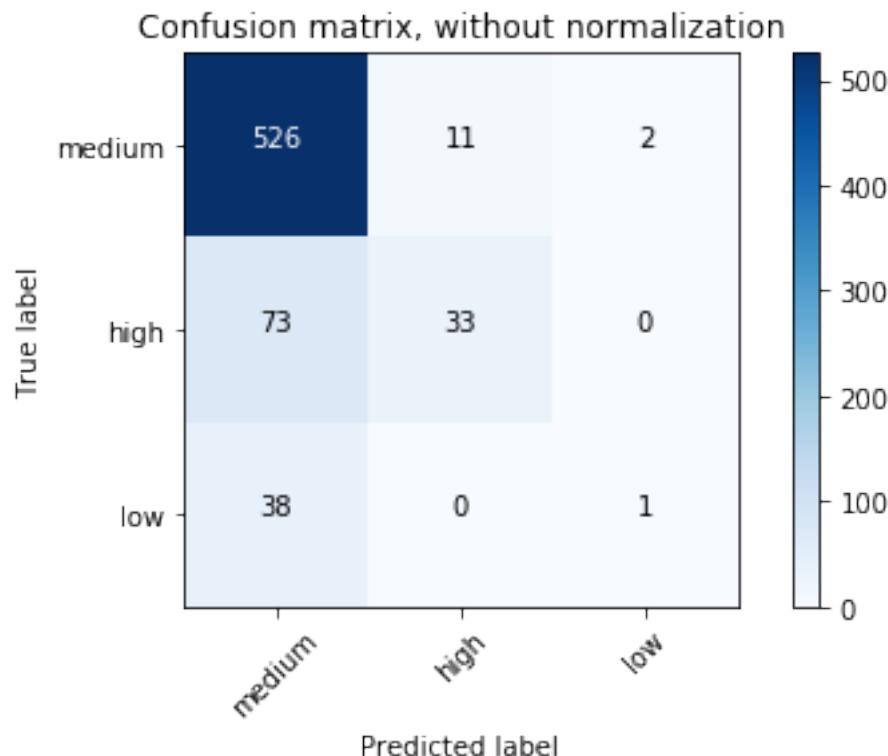
```
y_pred = predictions.select("predictedLabel")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred, labels=class_names)
cnf_matrix
```

```
array([[526,   11,    2],
       [ 73,   33,    0],
       [ 38,    0,    1]])
```

```
# 비정규화된 혼동 행렬 그리기
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')
plt.show()
```

```
Confusion matrix, without normalization
[[526 11  2]
 [ 73 33  0]
 [ 38  0  1]]
```



```
# 정규화된 혼동 행렬 그리기
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
```

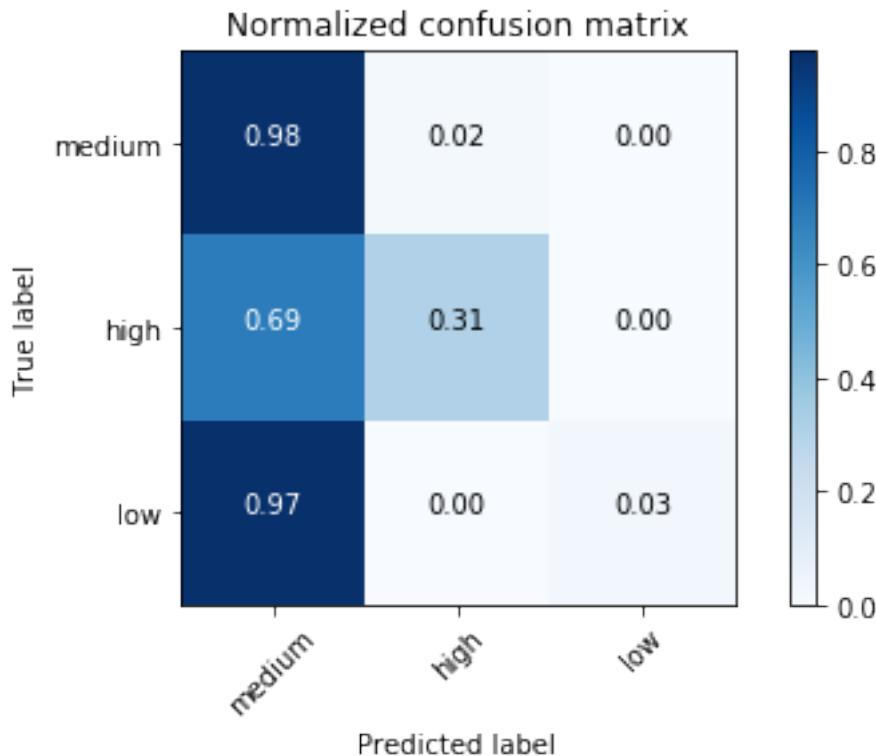
(다음 페이지에 계속)

(이전 페이지에 계속)

```
title='Normalized confusion matrix')

plt.show()
```

```
Normalized confusion matrix
[[0.97588126 0.02040816 0.00371058]
 [0.68867925 0.31132075 0.        ]
 [0.97435897 0.        0.02564103]]
```



11.3 의사결정 나무 - 분류

11.3.1 도입

11.3.2 테모

- 주피터 노트북은 의사결정 나무 분류에서 다운로드할 수 있습니다.
 - 자세한 내용은 의사결정 나무 분류자 API를 참조하시길 바랍니다.
1. 스파크 컨텍스트 및 스파크세션 설정

```
from pyspark.sql import SparkSession
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
spark = SparkSession \
    .builder \
    .appName("Python Spark Decision Tree classification") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. 데이터셋 로드하기

```
df = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', \
             inferSchema='true') \
    .load("../data/WineData2.csv", header=True);
df.show(5, True)
```

	fixed volatile citric sugar chlorides free total density	pH sulphates alcohol quality
7.4	0.7 9.4	0.0 5
7.8	0.88 9.8	0.0 5
7.8	0.76 9.8	0.04 5
11.2	0.28 9.8	0.56 6
7.4	0.7 9.4	1.9 5

only showing top 5 rows

```
# float형식으로 변환하기
def string_to_float(x):
    return float(x)

#
def condition(r):
    if (0 <= r <= 4):
        label = "low"
    elif(4 < r <= 6):
        label = "medium"
    else:
        label = "high"
    return label
```

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
string_to_float_udf = udf(string_to_float, DoubleType())
quality_udf = udf(lambda x: condition(x), StringType())
```

```
df = df.withColumn("quality", quality_udf("quality"))
df.show(5, True)
df.printSchema()
```

```
+----+-----+-----+-----+-----+-----+-----+-----+-----+
|fixed|volatile|citric|sugar|chlorides|free|total|density| ...
|pH|sulphates|alcohol|quality|
+----+-----+-----+-----+-----+-----+-----+-----+
| 7.4| 0.7| 0.0| 1.9| 0.076|11.0| 34.0| 0.9978|3.51| 0.56| ...
| 9.4| medium| | | | | | | |
| 7.8| 0.88| 0.0| 2.6| 0.098|25.0| 67.0| 0.9968| 3.2| 0.68| ...
| 9.8| medium| | | | | | | |
| 7.8| 0.76| 0.04| 2.3| 0.092|15.0| 54.0| 0.997|3.26| 0.65| ...
| 9.8| medium| | | | | | | |
| 11.2| 0.28| 0.56| 1.9| 0.075|17.0| 60.0| 0.998|3.16| 0.58| ...
| 9.8| medium| | | | | | | |
| 7.4| 0.7| 0.0| 1.9| 0.076|11.0| 34.0| 0.9978|3.51| 0.56| ...
| 9.4| medium| | | | | | | |
+----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
root
|-- fixed: double (nullable = true)
|-- volatile: double (nullable = true)
|-- citric: double (nullable = true)
|-- sugar: double (nullable = true)
|-- chlorides: double (nullable = true)
|-- free: double (nullable = true)
|-- total: double (nullable = true)
|-- density: double (nullable = true)
|-- pH: double (nullable = true)
|-- sulphates: double (nullable = true)
|-- alcohol: double (nullable = true)
|-- quality: string (nullable = true)
```

3. 데이터를 고밀도 벡터로 변환하기

노트:

여러분들께서 복잡한 데이터 셋에서 범주형 데이터를 처리하기 위해 저의 `get_dummy` 함수를 사용해 보시길 적극 권장합니다.

지도 학습 버전:

```

def get_dummy(df, indexCol, categoricalCols, continuousCols,
    labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer,
    OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_"
        indexed".format(c))
        for c in categoricalCols ]

    # 기본 설정: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.
        getOutputCol(),
        outputCol="{0}_encoded".format(indexer.
        getOutputCol()))
        for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.
        getOutputCol() for encoder in encoders]
        + continuousCols, outputCol=
        "features")

    pipeline = Pipeline(stages=indexers + encoders +_
        [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label', col(labelCol))

    return data.select(indexCol, 'features', 'label')

```

비지도학습 버전:

```

def get_dummy(df, indexCol, categoricalCols, continuousCols):
    """
    더미 변수를 가져와 비지도학습을 위해 연속형 변수와 결합합니다.

    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터의 이름 목록

    :반환 k: 특징 행렬

    :저자: Wenqiang Feng
    :이메일: von198@gmail.com
    """

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_"
        indexed".format(c))

```

(다음 페이지에 계속)

(o] 전 페이지에서 계속)

```

        for c in categoricalCols ] 

        # 기본 설정: dropLast=True
        encoders = [ OneHotEncoder(inputCol=indexer.
→getOutputCol(),
                                outputCol="{0}_encoded".format(indexer.
→getOutputCol())))
        for indexer in indexers ] 

        assembler = VectorAssembler(inputCols=[encoder.
→getOutputCol() for encoder in encoders]
                                + continuousCols, outputCol=
→"features") 

        pipeline = Pipeline(stages=indexers + encoders +_
→[assembler]) 

        model=pipeline.fit(df)
        data = model.transform(df)

        return data.select(indexCol,'features')
    
```

두 가지 버전을 한 번에 적용하기:

```

def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol,
→dropLast=False) :

    """
    더미 변수를 가져와 머신러닝 모델링을 위한 연속형 변수와 결합합니다.

    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터의 이름 목록
    :param labelCol: 라벨 열의 이름
    :param dropLast: 마지막 열을 버릴 것인지를 설정하는 옵션
    :반환: 특징 행렬

    :작자: Wenqiang Feng
    :이메일: von198@gmail.com

>>> df = spark.createDataFrame([
        (0, "a"),
        (1, "b"),
        (2, "c"),
        (3, "a"),
        (4, "a"),
        (5, "c")
    ], ["id", "category"])

>>> indexCol = 'id'
>>> categoricalCols = ['category']
>>> continuousCols = []
    
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

>>> labelCol = []

>>> mat = get_dummy(df, indexCol, categoricalCols, continuousCols,
↪labelCol)
>>> mat.show()

>>>
+---+-----+
| id|    features|
+---+-----+
| 0|[1.0,0.0,0.0]|
| 1|[0.0,0.0,1.0]|
| 2|[0.0,1.0,0.0]|
| 3|[1.0,0.0,0.0]|
| 4|[1.0,0.0,0.0]|
| 5|[0.0,1.0,0.0]|
+---+-----+
,,

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
↪VectorAssembler
from pyspark.sql.functions import col

indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
             for c in categoricalCols ]

# 기본 설정: dropLast=True
encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                           outputCol="{0}_encoded".format(indexer.
↪getOutputCol()), dropLast=dropLast)
              for indexer in indexers ]

assembler = VectorAssembler(inputCols=[encoder.getOutputCol()].
↪for encoder in encoders]
                           + continuousCols, outputCol="features"
↪)

pipeline = Pipeline(stages=indexers + encoders + [assembler])

model=pipeline.fit(df)
data = model.transform(df)

if indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select(indexCol, 'features', 'label')
elif not indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select('features', 'label')

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
elif indexCol and not labelCol:
    # 빠지도학습의 경우
    return data.select(indexCol,'features')
elif not indexCol and not labelCol:
    # 빠지도학습의 경우
    return data.select('features')
```

```
# !!!!주의: from pyspark.mllib.linalg import Vectors가 아닙니다.
from pyspark.ml.linalg import Vectors
from pyspark.ml import Pipeline
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1]),r[-1]]).toDF([
    'features','label'])
```

4. 데이터 셋을 데이터 프레임으로 변환하기

```
transformed = transData(df)
transformed.show(5)
```

```
+-----+-----+
|      features| label|
+-----+-----+
| [7.4,0.7,0.0,1.9,...|medium|
| [7.8,0.88,0.0,2.6...|medium|
| [7.8,0.76,0.04,2....|medium|
| [11.2,0.28,0.56,1...|medium|
| [7.4,0.7,0.0,1.9,...|medium|
+-----+-----+
only showing top 5 rows
```

5. 범주형 라벨과 변수들 처리하기

```
# 라벨들을 색인하고 메타데이터에 라벨 열 추가하기
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(transformed)
labelIndexer.transform(transformed).show(5, True)
```

```
+-----+-----+-----+
|      features| label|indexedLabel|
+-----+-----+-----+
| [7.4,0.7,0.0,1.9,...|medium|          0.0|
| [7.8,0.88,0.0,2.6...|medium|          0.0|
| [7.8,0.76,0.04,2....|medium|          0.0|
| [11.2,0.28,0.56,1...|medium|          0.0|
| [7.4,0.7,0.0,1.9,...|medium|          0.0|
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
+-----+-----+
only showing top 5 rows
```

```
# 범주형 특징을 자동으로 식별하고 인덱싱합니다.
# maxCategories를 지정하여 4개 이상의 고유 값을 가진 특징이 연속형으로 처리되도록 합니다.

featureIndexer = VectorIndexer(inputCol="features", \
                                outputCol="indexedFeatures", \
                                maxCategories=4).fit(transformed)
featureIndexer.transform(transformed).show(5, True)
```

```
+-----+-----+
|       features| label|      indexedFeatures|
+-----+-----+-----+
| [7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
| [7.8,0.88,0.0,2.6...|medium|[7.8,0.88,0.0,2.6...|
| [7.8,0.76,0.04,2....|medium|[7.8,0.76,0.04,2....|
| [11.2,0.28,0.56,1...|medium|[11.2,0.28,0.56,1...|
| [7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
+-----+-----+
only showing top 5 rows
```

6. 데이터를 훈련 및 테스트 셋으로 분할하기

```
# 데이터를 훈련 및 테스트 셋으로 분할합니다(40%는 테스트 셋)
(trainingData, testData) = transformed.randomSplit([0.6, 0.4])

trainingData.show(5)
testData.show(5)
```

```
+-----+-----+
|       features| label|
+-----+-----+
| [4.6,0.52,0.15,2....| low|
| [4.7,0.6,0.17,2.3...|medium|
| [5.0,1.02,0.04,1....| low|
| [5.0,1.04,0.24,1....|medium|
| [5.1,0.585,0.0,1....| high|
+-----+-----+
only showing top 5 rows

+-----+-----+
|       features| label|
+-----+-----+
| [4.9,0.42,0.0,2.1...| high|
| [5.0,0.38,0.01,1....|medium|
| [5.0,0.4,0.5,4.3,...|medium|
| [5.0,0.42,0.24,2....| high|
| [5.0,0.74,0.0,1.2...|medium|
+-----+-----+
only showing top 5 rows
```

7. 의사결정 나무 분류 모형 적합하기

```
from pyspark.ml.classification import DecisionTreeClassifier

# 의사결정 나무 모형 학습하기
dTree = DecisionTreeClassifier(labelCol='indexedLabel', featuresCol=
    'indexedFeatures')
```

8. 파이프라인 아키텍처

```
# 인덱싱된 라벨들을 본래 라벨들로 변환하기
labelConverter = IndexToString(inputCol="prediction", outputCol=
    "predictedLabel",
    labels=labelIndexer.labels)
```

```
# 파이프라인에 인덱서와 트리 묶기
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dTree,
    labelConverter])
```

```
# 모형을 훈련하기. 인덱서들도 같이 실행됩니다.
model = pipeline.fit(trainingData)
```

9. 예측값 만들기

```
# 예측값 만들기
predictions = model.transform(testData)
# 표시할 예제 행 선택하기
predictions.select("features", "label", "predictedLabel").show(5)
```

```
+-----+-----+-----+
|       features| label|predictedLabel|
+-----+-----+-----+
| [4.9, 0.42, 0.0, 2.1...| high|      high|
| [5.0, 0.38, 0.01, 1....|medium|      medium|
| [5.0, 0.4, 0.5, 4.3,...|medium|      medium|
| [5.0, 0.42, 0.24, 2....| high|      medium|
| [5.0, 0.74, 0.0, 1.2...|medium|      medium|
+-----+-----+-----+
only showing top 5 rows
```

10. 평가하기

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# (예측, 실제 라벨) 을 선택하고 검정 오차 계산하기
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
rfModel = model.stages[-2]
print(rfModel) # summary만 표시
```

```
Test Error = 0.45509
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_
˓→4545ac8dca9c8438ef2a)
of depth 5 with 59 nodes
```

11. 시각화하기

```
import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm, classes,
                         normalize=False,
                         title='Confusion matrix',
                         cmap=plt.cm.Blues):

    """
    이 함수는 혼동 행렬을 출력하고 그립니다. 정규화는 normalize=True' 를
    설정하여 적용할 수 있습니다. """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
class_temp = predictions.select("label").groupBy("label") \
    .count().sort('count', ascending=False).toPandas()
class_temp = class_temp["label"].values.tolist()
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
class_names = map(str, class_temp)
# # # print(class_name)
class_names
```

```
['medium', 'high', 'low']
```

```
from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()

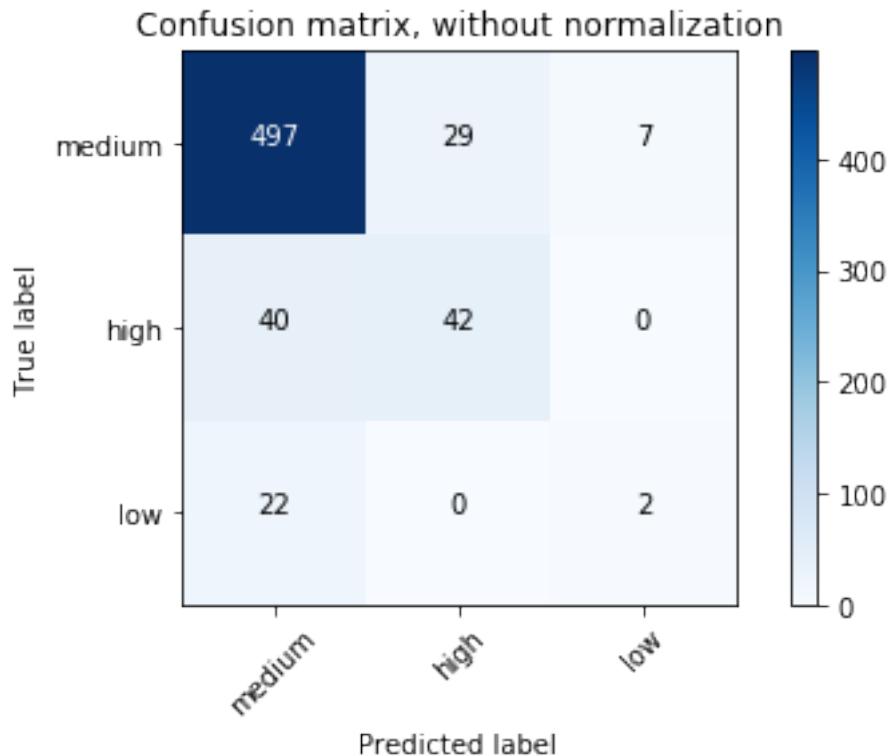
y_pred = predictions.select("predictedLabel")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred, labels=class_names)
cnf_matrix
```

```
array([[497,  29,    7],
       [ 40,  42,    0],
       [ 22,    0,    2]])
```

```
# 비정규화된 혼동 행렬 그리기
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')
plt.show()
```

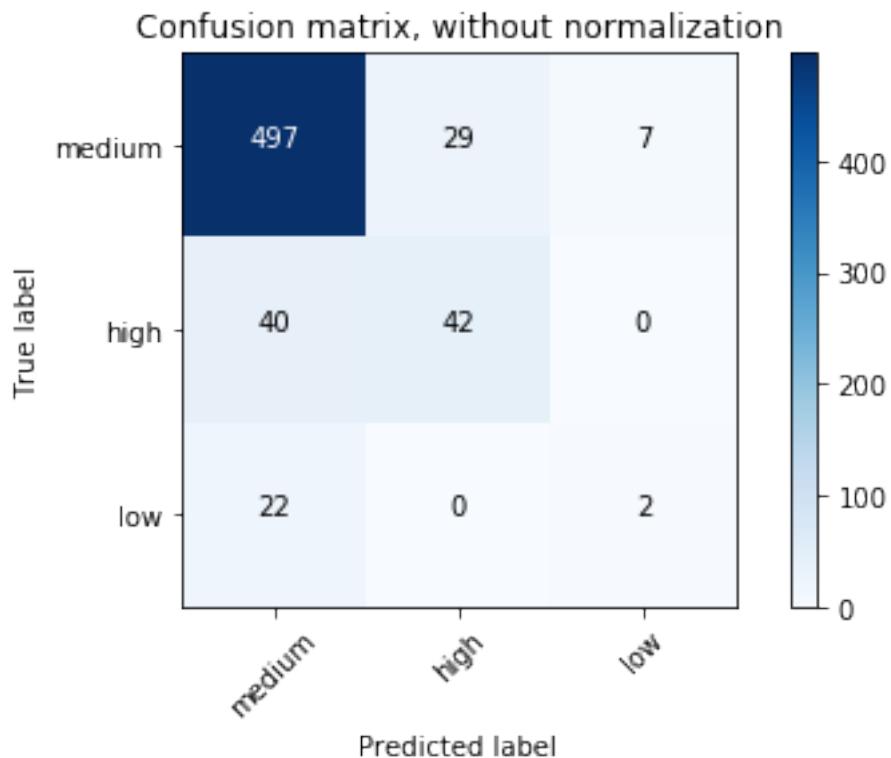
```
Confusion matrix, without normalization
[[497  29   7]
 [ 40  42   0]
 [ 22   0   2]]
```



```
# 정규화된 혼동 행렬 그리기
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Normalized confusion matrix
[[ 0.93245779  0.05440901  0.01313321]
 [ 0.48780488  0.51219512  0.          ]
 [ 0.91666667  0.          0.08333333]]
```



11.4 랜덤 포레스트 - 분류

11.4.1 도입

11.4.2 테모

- 주피터 노트북은 랜덤 포레스트 분류에서 다운로드할 수 있습니다.
 - 자세한 내용은 랜덤 포레스트 분류자 API를 참조하시길 바랍니다.
- 스파크 컨텍스트 및 스파크세션 설정

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark Decision Tree classification") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

- 데이터셋 로드하기

```
df = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', \
    inferSchema='true') \
```

(다음 페이지에서 계속)

(o) 전 페이지에서 계속)

```
.load("../data/WineData2.csv", header=True);
df.show(5, True)
```

```
+----+-----+-----+-----+-----+-----+-----+-----+
|fixed|volatile|citric|sugar|chlorides|free|total|density|
|pH|sulphates|alcohol|quality|
+----+-----+-----+-----+-----+-----+-----+-----+
| 7.4|    0.7|    0.0|   1.9|    0.076|11.0| 34.0| 0.9978|3.51|    0.56|
| 9.4|      5|          |          |          |          |          |          |          |
| 7.8|    0.88|    0.0|   2.6|    0.098|25.0| 67.0| 0.9968| 3.2|    0.68|
| 9.8|      5|          |          |          |          |          |          |
| 7.8|    0.76|    0.04|   2.3|    0.092|15.0| 54.0| 0.997|3.26|    0.65|
| 9.8|      5|          |          |          |          |          |          |
| 11.2|    0.28|    0.56|   1.9|    0.075|17.0| 60.0| 0.998|3.16|    0.58|
| 9.8|      6|          |          |          |          |          |          |
| 7.4|    0.7|    0.0|   1.9|    0.076|11.0| 34.0| 0.9978|3.51|    0.56|
| 9.4|      5|          |          |          |          |          |          |
+----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
# float 형식으로 변환하기
def string_to_float(x):
    return float(x)

#
def condition(r):
    if (0 <= r <= 4):
        label = "low"
    elif(4 < r <= 6):
        label = "medium"
    else:
        label = "high"
    return label
```

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType
string_to_float_udf = udf(string_to_float, DoubleType())
quality_udf = udf(lambda x: condition(x), StringType())
```

```
df = df.withColumn("quality", quality_udf("quality"))
df.show(5, True)
df.printSchema()
```

```
+----+-----+-----+-----+-----+-----+-----+-----+
|fixed|volatile|citric|sugar|chlorides|free|total|density|
|pH|sulphates|alcohol|quality|
+----+-----+-----+-----+-----+-----+-----+-----+
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
root
|-- fixed: double (nullable = true)
|-- volatile: double (nullable = true)
|-- citric: double (nullable = true)
|-- sugar: double (nullable = true)
|-- chlorides: double (nullable = true)
|-- free: double (nullable = true)
|-- total: double (nullable = true)
|-- density: double (nullable = true)
|-- pH: double (nullable = true)
|-- sulphates: double (nullable = true)
|-- alcohol: double (nullable = true)
|-- quality: string (nullable = true)
```

3. 데이터를 고밀도 벡터로 변환하기

10

여러분들께서 복잡한 데이터 셋에서 범주형 데이터를 처리하기 위해 저의 `get_dummy` 함수를 사용해 보시길 적극 권장합니다.

지도 학습 버전:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols,
             labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer,_
    OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{}_"_
    .format(c))
        for c in categoricalCols ]
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

# 기본 설정: dropLast=True
encoders = [ OneHotEncoder(inputCol=indexer.
    ↪getOutputCol(),
    outputCol="{0}_encoded".format(indexer.
    ↪getOutputCol())))
    for indexer in indexers ]

assembler = VectorAssembler(inputCols=[encoder.
    ↪getOutputCol() for encoder in encoders]
    + continuousCols, outputCol=
    ↪"features")

pipeline = Pipeline(stages=indexers + encoders +_
    ↪[assembler])

model=pipeline.fit(df)
data = model.transform(df)

data = data.withColumn('label', col(labelCol))

return data.select(indexCol,'features','label')

```

비지도학습 버전:

```

def get_dummy(df,indexCol,categoricalCols,continuousCols):
    """
    디미 변수를 가져와 비지도 학습을 위한 연속 변수와 결합합니다.

    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터의 이름 목록
    """

    :반환 k: 특징 행렬

    :저자: Wenqiang Feng
    :이메일: von198@gmail.com
    """

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_"
    ↪indexed".format(c))
        for c in categoricalCols ]

    # 기본 설정: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.
    ↪getOutputCol(),
    outputCol="{0}_encoded".format(indexer.
    ↪getOutputCol())))
        for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.
    ↪getOutputCol() for encoder in encoders])

```

(다음 페이지에서 계속)

(이전 페이지에서 계속)

```
+ continuousCols, outputCol=
↳ "features")

pipeline = Pipeline(stages=indexers + encoders +_
↳ [assembler])

model=pipeline.fit(df)
data = model.transform(df)

return data.select(indexCol, 'features')
```

두 가지 버전을 한 번에 적용하기:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols, labelCol,
↳ dropLast=False) :

    """
    더미 변수를 가져와 머신러닝 모델링을 위해 연속형 변수와 결합합니다.

    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터의 이름 목록
    :param labelCol: 라벨 열의 이름
    :param dropLast: 마지막 열을 버릴 것인지를 설정하는 옵션
    :반환: 특징 행렬

    :작자: Wenqiang Feng
    :이메일: von198@gmail.com

>>> df = spark.createDataFrame([
        (0, "a"),
        (1, "b"),
        (2, "c"),
        (3, "a"),
        (4, "a"),
        (5, "c")
    ], ["id", "category"])

>>> indexCol = 'id'
>>> categoricalCols = ['category']
>>> continuousCols = []
>>> labelCol = []

>>> mat = get_dummy(df, indexCol, categoricalCols, continuousCols,
↳ labelCol)
>>> mat.show()

>>>
+---+-----+
| id|    features|
+---+-----+
|  0|[1.0,0.0,0.0]|

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

| 1|[0.0,0.0,1.0]|
| 2|[0.0,1.0,0.0]|
| 3|[1.0,0.0,0.0]|
| 4|[1.0,0.0,0.0]|
| 5|[0.0,1.0,0.0]|
+---+-----+
''

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
→VectorAssembler
from pyspark.sql.functions import col

indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
→format(c))
             for c in categoricalCols ]

# 기본 설정: dropLast=True
encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                           outputCol="{0}_encoded".format(indexer.
→getOutputCol()), dropLast=dropLast)
              for indexer in indexers ]

assembler = VectorAssembler(inputCols=[encoder.getOutputCol()].
→for encoder in encoders]
                           + continuousCols, outputCol="features"
→""

pipeline = Pipeline(stages=indexers + encoders + [assembler])

model=pipeline.fit(df)
data = model.transform(df)

if indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select(indexCol, 'features', 'label')
elif not indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select('features', 'label')
elif indexCol and not labelCol:
    # 비지도학습의 경우
    return data.select(indexCol, 'features')
elif not indexCol and not labelCol:
    # 비지도학습의 경우
    return data.select('features')

```

!!!!주의: from pyspark.mllib.linalg import Vectors 가 아닙니다.
from pyspark.ml.linalg import Vectors

(다음 페이지에 계속)

(이전 페이지에 계속)

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1]), r[-1]]).toDF([
        'features', 'label'])
```

4. 데이터셋을 데이터프레임으로 변환하기

```
transformed = transData(df)
transformed.show(5)
```

```
+-----+-----+
|      features| label|
+-----+-----+
| [7.4,0.7,0.0,1.9,...|medium|
| [7.8,0.88,0.0,2.6...|medium|
| [7.8,0.76,0.04,2....|medium|
| [11.2,0.28,0.56,1...|medium|
| [7.4,0.7,0.0,1.9,...|medium|
+-----+-----+
only showing top 5 rows
```

5. 범주형 라벨과 변수들 처리하기

```
# 라벨 색인을 만들고 메타데이터에 라벨 컬럼을 추가하기
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(transformed)
labelIndexer.transform(transformed).show(5, True)
```

```
+-----+-----+-----+
|      features| label|indexedLabel|
+-----+-----+-----+
| [7.4,0.7,0.0,1.9,...|medium|          0.0|
| [7.8,0.88,0.0,2.6...|medium|          0.0|
| [7.8,0.76,0.04,2....|medium|          0.0|
| [11.2,0.28,0.56,1...|medium|          0.0|
| [7.4,0.7,0.0,1.9,...|medium|          0.0|
+-----+-----+-----+
only showing top 5 rows
```

범주형 특징을 자동으로 식별하고 인덱싱합니다.
maxCategories를 지정하여 4개 이상의 고유 값을 가진 특징이 연속형으로 처리되도록 합니다.

```
featureIndexer = VectorIndexer(inputCol="features", \
                                outputCol="indexedFeatures", \
                                maxCategories=4).fit(transformed)
featureIndexer.transform(transformed).show(5, True)
```

```
+-----+-----+
|       features| label|      indexedFeatures|
+-----+-----+
|[7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
|[7.8,0.88,0.0,2.6...|medium|[7.8,0.88,0.0,2.6...|
|[7.8,0.76,0.04,2....|medium|[7.8,0.76,0.04,2....|
|[11.2,0.28,0.56,1...|medium|[11.2,0.28,0.56,1...|
|[7.4,0.7,0.0,1.9,...|medium|[7.4,0.7,0.0,1.9,...|
+-----+-----+
only showing top 5 rows
```

6. 데이터를 훈련 및 테스트 셋으로 분할하기

```
# 데이터를 훈련 및 테스트 셋으로 분할합니다(40%는 테스트 셋)
(trainingData, testData) = transformed.randomSplit([0.6, 0.4])

trainingData.show(5)
testData.show(5)
```

```
+-----+-----+
|       features| label|
+-----+-----+
|[4.6,0.52,0.15,2....|   low|
|[4.7,0.6,0.17,2.3...|medium|
|[5.0,1.02,0.04,1....|   low|
|[5.0,1.04,0.24,1....|medium|
|[5.1,0.585,0.0,1....|   high|
+-----+-----+
only showing top 5 rows

+-----+-----+
|       features| label|
+-----+-----+
|[4.9,0.42,0.0,2.1...|   high|
|[5.0,0.38,0.01,1....|medium|
|[5.0,0.4,0.5,4.3,...|medium|
|[5.0,0.42,0.24,2....|   high|
|[5.0,0.74,0.0,1.2...|medium|
+-----+-----+
only showing top 5 rows
```

7. 랜덤 포레스트 분류 모형 적합하기

```
from pyspark.ml.classification import RandomForestClassifier

# 랜덤 포레스트 모형 훈련하기
rf = RandomForestClassifier(labelCol="indexedLabel", featuresCol=
    "indexedFeatures", numTrees=10)
```

8. 파이프라인 아키텍쳐

```
# 인덱싱된 라벨들을 본래 라벨들로 변환하기  
labelConverter = IndexToString(inputCol="prediction", outputCol=  
    "predictedLabel",  
    labels=labelIndexer.labels)
```

```
# 파이프라인에 인덱서와 트리 묶기  
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, rf, labelConverter])
```

```
# 모형을 훈련하기. 인덱서들도 같이 실행됩니다.  
model = pipeline.fit(trainingData)
```

9. 예측값 만들기

```
# 예측값 만들기  
predictions = model.transform(testData)  
# 표시할 예시 행 선택하기  
predictions.select("features", "label", "predictedLabel").show(5)
```

```
+-----+-----+-----+  
| features| label|predictedLabel|  
+-----+-----+-----+  
| [4.9,0.42,0.0,2.1...| high| high|  
| [5.0,0.38,0.01,1....|medium| medium|  
| [5.0,0.4,0.5,4.3,...|medium| medium|  
| [5.0,0.42,0.24,2....| high| medium|  
| [5.0,0.74,0.0,1.2...|medium| medium|  
+-----+-----+-----+  
only showing top 5 rows
```

10. 평가하기

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator  
  
# (예측, 실제 라벨) 을 선택하고 검정 오차 계산하기  
evaluator = MulticlassClassificationEvaluator(  
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")  
accuracy = evaluator.evaluate(predictions)  
print("Test Error = %g" % (1.0 - accuracy))  
  
rfModel = model.stages[-2]  
print(rfModel) # summary만 표시하기
```

```
Test Error = 0.173502  
RandomForestClassificationModel (uid=rfc_a3395531f1d2) with 10 trees
```

11. 시각화하기

```
import matplotlib.pyplot as plt  
import numpy as np
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

import itertools

def plot_confusion_matrix(cm, classes,
                         normalize=False,
                         title='Confusion matrix',
                         cmap=plt.cm.Blues):

    """
    이 함수는 혼동 행렬을 출력하고 그립니다. 정규화는 'normalize=True'를
    설정하여 적용할 수 있습니다.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

```

```

class_temp = predictions.select("label").groupBy("label") \
    .count().sort('count', ascending=False).toPandas()
class_temp = class_temp["label"].values.tolist()
class_names = map(str, class_temp)
# # # print(class_name)
class_names

```

```
['medium', 'high', 'low']
```

```

from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()

y_pred = predictions.select("predictedLabel")

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

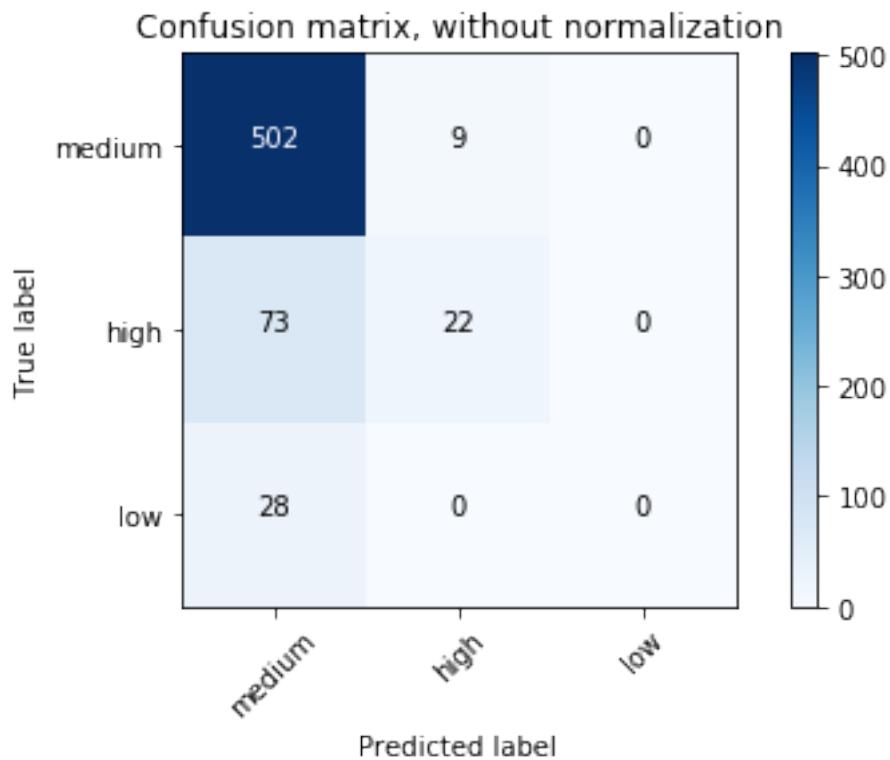
```
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred, labels=class_names)
cnf_matrix
```

```
array([[502,    9,    0],
       [ 73,   22,    0],
       [ 28,    0,    0]])
```

```
# 비정규화된 혼동 행렬 그리기
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')
plt.show()
```

```
Confusion matrix, without normalization
[[502  9  0]
 [ 73 22  0]
 [ 28  0  0]]
```



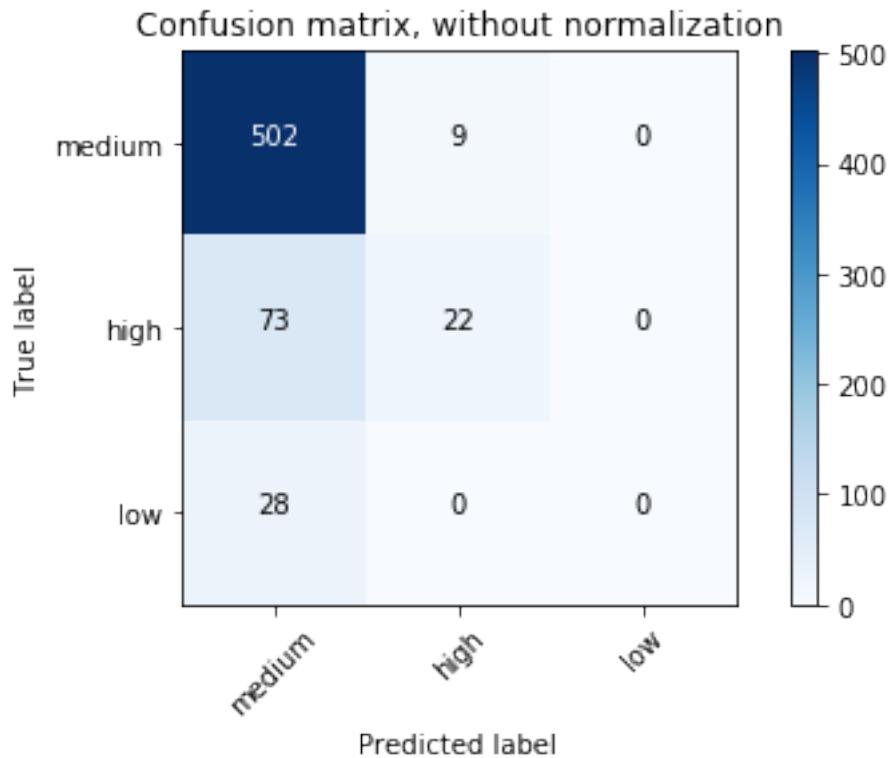
```
# 정규화된 혼동 행렬 그리기
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
plt.show()
```

```
Normalized confusion matrix
[[ 0.98238748  0.01761252  0.        ]
 [ 0.76842105  0.23157895  0.        ]
 [ 1.          0.          0.        ]]
```



11.5 그레디언트 부스팅 트리 - 분류

11.5.1 도입

11.5.2 테모

- 주피터 노트북은 그레디언트 부스팅 트리 분류에서 다운로드할 수 있습니다.
- 자세한 내용은 GBTClassifier API를 참조하시길 바랍니다.

경고: 안타깝게도 현재 GBTClassifier는 이항 라벨에 대해서만 지원하고 있습니다.

11.6 XGBoost: 그레디언트 부스팅 트리 분류

11.6.1 도입

11.6.2 테모

- 주피터 노트북은 그레디언트 부스팅 트리 분류에서 다운로드할 수 있습니다.
- 자세한 내용은 GBTClassifier API를 참조하시길 바랍니다.

경고: 불행히도 Spark에서 직접 XGBoost를 설치하는 방법을 찾지 못했지만 제 머신에서 pysparkling으로 XGBoost가 작동하는 방법을 찾았습니다.

1. Spark 환경에서 H2O 클러스터 시작하기

```
from pysparkling import *
hc = H2OContext.getOrCreate(spark)
```

```
Connecting to H2O server at http://192.168.0.102:54323... successful.
H2O cluster uptime:      07 secs
H2O cluster timezone:    America/Chicago
H2O data parsing timezone:    UTC
H2O cluster version:     3.22.1.3
H2O cluster version age:    20 days
H2O cluster name:        sparkling-water-dt21661_local-1550259209801
H2O cluster total nodes:   1
H2O cluster free memory:   848 Mb
H2O cluster total cores:   8
H2O cluster allowed cores: 8
H2O cluster status:       accepting new members, healthy
H2O connection url:      http://192.168.0.102:54323
H2O connection proxy:     None
H2O internal security:    False
H2O API Extensions:      XGBoost, Algos, AutoML, Core V3, Core V4
Python version:            3.7.1 final

Sparkling Water Context:
* H2O name: sparkling-water-dt21661_local-1550259209801
* cluster size: 1
* list of used nodes:
  (executorId, host, port)
  -----
  (driver, 192.168.0.102, 54323)
  -----
```

Open H2O Flow in browser: http://192.168.0.102:54323 (CMD + click in Mac
OSX)

2. H2O를 이용해 데이터 구문을 분석하고 분석된 데이터를 Spark 프레임으로 변환하기

```
import h2o
frame = h2o.import_file("https://raw.githubusercontent.com/h2oai/sparkling-
    ↪water/master/examples/smalldata/prostate/prostate.csv")
spark_frame = hc.as_spark_frame(frame)
```

Parse progress: |██████████| 100%

```
spark_frame.show(4)
```

ID	CAPSULE	AGE	RACE	DPROS	DCAPS	PSA	VOL	GLEASON
1	0	65	1	2	1	1.4	0.0	6
2	0	72	1	3	2	6.7	0.0	7
3	0	70	1	1	2	4.9	0.0	6
4	0	76	2	2	1	51.2	20.0	7

only showing top 4 rows

3. 모델 훈련하기

```
from pysparkling.ml import H2OXGBoost
estimator = H2OXGBoost(predictionCol="AGE")
model = estimator.fit(spark_frame)
```

4. 예측하기

```
predictions = model.transform(spark_frame)
predictions.show(4)
```

ID	CAPSULE	AGE	RACE	DPROS	DCAPS	PSA	VOL	GLEASON	prediction_output
1	0	65	1	2	1	1.4	0.0	6	[64.85852813720703]
2	0	72	1	3	2	6.7	0.0	7	[72.0611801147461]
3	0	70	1	1	2	4.9	0.0	6	[70.26496887207031]
4	0	76	2	2	1	51.2	20.0	7	[75.26521301269531]

only showing top 4 rows

11.7 나이브 베이즈 분류(Naive Bayes Classification)

11.7.1 도입

11.7.2 테모

- 주피터 노트북은 나이브 베이즈 분류에서 다운로드할 수 있습니다.

- 더 자세한 내용은 [나이브 베이즈 API](#)를 참조하시길 바랍니다.

1. 스파크 컨텍스트 및 스파크세션 설정

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark Naive Bayes classification") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. 데이터셋 로드하기

```
df = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', inferSchema='true') \
    .load("./data/WineData2.csv", header=True);
df.show(5)
```

```
+----+-----+-----+-----+-----+-----+-----+-----+
|fixed|volatile|citric|sugar|chlorides|free|total|density|
|pH|sulphates|alcohol|quality|
+----+-----+-----+-----+-----+-----+-----+-----+
| 7.4| 0.7| 0.0| 1.9| 0.076|11.0| 34.0| 0.9978|3.51| 0.56|
| 9.4| 5| |
| 7.8| 0.88| 0.0| 2.6| 0.098|25.0| 67.0| 0.9968| 3.2| 0.68|
| 9.8| 5| |
| 7.8| 0.76| 0.04| 2.3| 0.092|15.0| 54.0| 0.997|3.26| 0.65|
| 9.8| 5| |
| 11.2| 0.28| 0.56| 1.9| 0.075|17.0| 60.0| 0.998|3.16| 0.58|
| 9.8| 6| |
| 7.4| 0.7| 0.0| 1.9| 0.076|11.0| 34.0| 0.9978|3.51| 0.56|
| 9.4| 5| |
+----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
df.printSchema()
```

```
root
|-- fixed: double (nullable = true)
|-- volatile: double (nullable = true)
|-- citric: double (nullable = true)
|-- sugar: double (nullable = true)
|-- chlorides: double (nullable = true)
|-- free: double (nullable = true)
|-- total: double (nullable = true)
|-- density: double (nullable = true)
|-- pH: double (nullable = true)
|-- sulphates: double (nullable = true)
```

(다음 페이지에 계속)

(○] 전 페이지에서 계속)

```
|-- alcohol: double (nullable = true)
|-- quality: string (nullable = true)
```

```
# float형식으로 변환하기
def string_to_float(x):
    return float(x)

#
def condition(r):
    if (0 <= r <= 6):
        label = "low"
    else:
        label = "high"
    return label

from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType
string_to_float_udf = udf(string_to_float, DoubleType())
quality_udf = udf(lambda x: condition(x), StringType())

df = df.withColumn("quality", quality_udf("quality"))

df.show(5, True)
```

```
+----+-----+-----+-----+-----+-----+-----+-----+-----+
|fixed|volatile|citric|sugar|chlorides|free|total|density| 
|pH|sulphates|alcohol|quality|
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| 7.4| 0.7| 0.0| 1.9| 0.076|11.0| 34.0| 0.9978|3.51| 0.56| 
| 9.4| medium| 
| 7.8| 0.88| 0.0| 2.6| 0.098|25.0| 67.0| 0.9968| 3.2| 0.68| 
| 9.8| medium| 
| 7.8| 0.76| 0.04| 2.3| 0.092|15.0| 54.0| 0.997|3.26| 0.65| 
| 9.8| medium| 
| 11.2| 0.28| 0.56| 1.9| 0.075|17.0| 60.0| 0.998|3.16| 0.58| 
| 9.8| medium| 
| 7.4| 0.7| 0.0| 1.9| 0.076|11.0| 34.0| 0.9978|3.51| 0.56| 
| 9.4| medium| 
+----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
df.printSchema()
```

```
root
 |-- fixed: double (nullable = true)
 |-- volatile: double (nullable = true)
 |-- citric: double (nullable = true)
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
|-- sugar: double (nullable = true)
|-- chlorides: double (nullable = true)
|-- free: double (nullable = true)
|-- total: double (nullable = true)
|-- density: double (nullable = true)
|-- pH: double (nullable = true)
|-- sulphates: double (nullable = true)
|-- alcohol: double (nullable = true)
|-- quality: string (nullable = true)
```

3. 범주형 데이터를 처리하고 데이터를 고밀도 벡터로 변환하기

노트:

여러분들께서 복잡한 데이터 셋에서 범주형 데이터를 처리하기 위해 저의 `get_dummy` 함수를 사용해 보시길 적극 권장합니다.

지도 학습 버전:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols,
             labelCol):
    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer,
    OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_"
    .format(c))
        for c in categoricalCols ]

    # 기본 설정: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.
    .getOutputCol(),
        outputCol="{0}_encoded".format(indexer.
    .getOutputCol()))
        for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.
    .getOutputCol() for encoder in encoders]
        + continuousCols, outputCol=
    "features")

    pipeline = Pipeline(stages=indexers + encoders +
    [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label', col(labelCol))

    return data.select(indexCol,'features','label')
```

비지도학습 버전:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols):
    """
    더미 변수를 가져와 비지도 학습을 위한 연속 변수와 결합합니다.

    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터의 이름 목록

    :반환 k: 특징 행렬

    :작자: Wenqiang Feng
    :이메일: von198@gmail.com
    """

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_"
    ↪indexed".format(c))
        for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.
    ↪getOutputCol(),
        outputCol="{0}_encoded".format(indexer.
    ↪getOutputCol()))
        for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.
    ↪getOutputCol() for encoder in encoders]
        + continuousCols, outputCol=
    ↪"features")

    pipeline = Pipeline(stages=indexers + encoders +_
    ↪[assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    return data.select(indexCol, 'features')
```

두 가지 버전을 한 번에 적용하기:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols, labelCol,
    ↪dropLast=False):

    """
    더미 변수를 가져와 머신러닝 모델링을 위한 연속형 변수와 결합합니다.

    :param df: 데이터 프레임
    :param categoricalCols: 범주형 데이터의 이름 목록
    :param continuousCols: 수치형 데이터의 이름 목록
    :param labelCol: 라벨 열의 이름
```

(다음 페이지에 계속)

(이전 페이지에 계속)

```

:param dropLast: 마지막 열을 버릴 것인지를 설정하는 옵션
:return: 특징 행렬

:작자: Wenqiang Feng
:이메일: von198@gmail.com

>>> df = spark.createDataFrame([
    (0, "a"),
    (1, "b"),
    (2, "c"),
    (3, "a"),
    (4, "a"),
    (5, "c")
], ["id", "category"])

>>> indexCol = 'id'
>>> categoricalCols = ['category']
>>> continuousCols = []
>>> labelCol = []

>>> mat = get_dummy(df, indexCol, categoricalCols, continuousCols,
labelCol)
>>> mat.show()

>>>
+---+-----+
| id|    features|
+---+-----+
|  0|[1.0,0.0,0.0]|
|  1|[0.0,0.0,1.0]|
|  2|[0.0,1.0,0.0]|
|  3|[1.0,0.0,0.0]|
|  4|[1.0,0.0,0.0]|
|  5|[0.0,1.0,0.0]|
+---+-----+
```

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
VectorAssembler
from pyspark.sql.functions import col

indexers = [StringIndexer(inputCol=c, outputCol="{0}_indexed".
format(c))
 for c in categoricalCols]

기본 설정: dropLast=True
encoders = [OneHotEncoder(inputCol=indexer.getOutputCol(),
outputCol="{0}_encoded".format(indexer.
getOutputCol()), dropLast=dropLast)
 for indexer in indexers]

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

assembler = VectorAssembler(inputCols=[encoder.getOutputCol()_
↪for encoder in encoders]
 + continuousCols, outputCol="features"
↪")

pipeline = Pipeline(stages=indexers + encoders + [assembler])

model=pipeline.fit(df)
data = model.transform(df)

if indexCol and labelCol:
 # 지도 학습의 경우
 data = data.withColumn('label', col(labelCol))
 return data.select(indexCol, 'features', 'label')
elif not indexCol and labelCol:
 # 지도 학습의 경우
 data = data.withColumn('label', col(labelCol))
 return data.select('features', 'label')
elif indexCol and not labelCol:
 # 비지도학습의 경우
 return data.select(indexCol, 'features')
elif not indexCol and not labelCol:
 # 비지도학습의 경우
 return data.select('features')

```

```

def get_dummy(df,categoricalCols,continuousCols,labelCol):

 from pyspark.ml import Pipeline
 from pyspark.ml.feature import StringIndexer, OneHotEncoder,
↪VectorAssembler
 from pyspark.sql.functions import col

 indexers = [StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
 for c in categoricalCols]

 # 기본 설정: dropLast=True
 encoders = [OneHotEncoder(inputCol=indexer.getOutputCol(),
 outputCol="{0}_encoded".format(indexer.getOutputCol()))
 for indexer in indexers]

 assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder_
↪in encoders]
 + continuousCols, outputCol="features")

 pipeline = Pipeline(stages=indexers + encoders + [assembler])

 model=pipeline.fit(df)
 data = model.transform(df)

 data = data.withColumn('label', col(labelCol))

```

(다음 페이지에 계속)

(이전 페이지에 계속)

```
return data.select('features', 'label')
```

#### 4. Transform the dataset to DataFrame

```
from pyspark.ml.linalg import Vectors
!!!주의: from pyspark.mllib.linalg import Vectors이 아닙니다
from pyspark.ml import Pipeline
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

def transData(data):
 return data.rdd.map(lambda r: [Vectors.dense(r[:-1]), r[-1]]).toDF(['features',
 'label'])
```

```
transformed = transData(df)
transformed.show(5)
```

```
+-----+-----+
| features|label|
+-----+-----+
[7.4,0.7,0.0,1.9,...	low
[7.8,0.88,0.0,2.6...	low
[7.8,0.76,0.04,2....	low
[11.2,0.28,0.56,1...	low
[7.4,0.7,0.0,1.9,...	low
+-----+-----+
only showing top 5 rows
```

#### 4. 범주형 라벨과 변수들 처리하기

```
라벨 색인을 만들고 메타데이터에 라벨 컬럼을 추가합니다
labelIndexer = StringIndexer(inputCol='label',
 outputCol='indexedLabel').fit(transformed)
labelIndexer.transform(transformed).show(5, True)
```

```
+-----+-----+-----+
| features|label|indexedLabel|
+-----+-----+-----+
[7.4,0.7,0.0,1.9,...	low	0.0
[7.8,0.88,0.0,2.6...	low	0.0
[7.8,0.76,0.04,2....	low	0.0
[11.2,0.28,0.56,1...	low	0.0
[7.4,0.7,0.0,1.9,...	low	0.0
+-----+-----+-----+
only showing top 5 rows
```

```
범주형 특징을 자동으로 식별하고 인덱싱합니다.
maxCategories를 지정하여 4개 이상의 고유 값을 가진 특징이 연속형으로 처리되도록 합니다.
```

(다음 페이지에 계속)

(o) 전 페이지에서 계속)

```
featureIndexer = VectorIndexer(inputCol="features", \
 outputCol="indexedFeatures", \
 maxCategories=4).fit(transformed)
featureIndexer.transform(transformed).show(5, True)
```

| features                     | label | indexedFeatures              |
|------------------------------|-------|------------------------------|
| [7.4, 0.7, 0.0, 1.9, ...]    | low   | [7.4, 0.7, 0.0, 1.9, ...]    |
| [7.8, 0.88, 0.0, 2.6, ...]   | low   | [7.8, 0.88, 0.0, 2.6, ...]   |
| [7.8, 0.76, 0.04, 2.0, ...]  | low   | [7.8, 0.76, 0.04, 2.0, ...]  |
| [11.2, 0.28, 0.56, 1.0, ...] | low   | [11.2, 0.28, 0.56, 1.0, ...] |
| [7.4, 0.7, 0.0, 1.9, ...]    | low   | [7.4, 0.7, 0.0, 1.9, ...]    |

only showing top 5 rows

## 5. 데이터를 훈련 및 테스트 셋으로 분할하기

```
데이터를 훈련 및 테스트 셋으로 분할합니다(40%는 테스트 셋)
(trainingData, testData) = data.randomSplit([0.6, 0.4])

trainingData.show(5, False)
testData.show(5, False)
```

| features                                                            | label |
|---------------------------------------------------------------------|-------|
| [5.0, 0.38, 0.01, 1.6, 0.048, 26.0, 60.0, 0.99084, 3.7, 0.75, 14.0] | low   |
| [5.0, 0.42, 0.24, 2.0, 0.06, 19.0, 50.0, 0.9917, 3.72, 0.74, 14.0]  | high  |
| [5.0, 0.74, 0.0, 1.2, 0.041, 16.0, 46.0, 0.99258, 4.01, 0.59, 12.5] | low   |
| [5.0, 1.02, 0.04, 1.4, 0.045, 41.0, 85.0, 0.9938, 3.75, 0.48, 10.5] | low   |
| [5.0, 1.04, 0.24, 1.6, 0.05, 32.0, 96.0, 0.9934, 3.74, 0.62, 11.5]  | low   |

only showing top 5 rows

| features                                                            | label |
|---------------------------------------------------------------------|-------|
| [4.6, 0.52, 0.15, 2.1, 0.054, 8.0, 65.0, 0.9934, 3.9, 0.56, 13.1]   | low   |
| [4.7, 0.6, 0.17, 2.3, 0.058, 17.0, 106.0, 0.9932, 3.85, 0.6, 12.9]  | low   |
| [4.9, 0.42, 0.0, 2.1, 0.048, 16.0, 42.0, 0.99154, 3.71, 0.74, 14.0] | high  |
| [5.0, 0.4, 0.5, 4.3, 0.046, 29.0, 80.0, 0.9902, 3.49, 0.66, 13.6]   | low   |
| [5.2, 0.49, 0.26, 2.3, 0.09, 23.0, 74.0, 0.9953, 3.71, 0.62, 12.2]  | low   |

only showing top 5 rows

## 6. 나이브 베이즈 분류 모형 적합하기

```
from pyspark.ml.classification import NaiveBayes
nb = NaiveBayes(featuresCol='indexedFeatures', labelCol='indexedLabel')
```

### 7. 파일프라인 아키텍처

```
인덱싱된 라벨들을 본래 라벨들로 변환하기
labelConverter = IndexToString(inputCol="prediction", outputCol=
 "predictedLabel",
 labels=labelIndexer.labels)
```

```
파일프라인에 인덱서와 트리 묶기
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, nb, labelConverter])
```

```
모형을 훈련하기. 인덱서들도 같이 실행됩니다.
model = pipeline.fit(trainingData)
```

### 8. 예측값 만들기

```
예측값 만들기
predictions = model.transform(testData)
표시할 예제 행 선택하기
predictions.select("features", "label", "predictedLabel").show(5)
```

```
+-----+-----+-----+
| features|label|predictedLabel|
+-----+-----+-----+
[4.6,0.52,0.15,2....	low	low
[4.7,0.6,0.17,2.3...	low	low
[4.9,0.42,0.0,2.1...	high	low
[5.0,0.4,0.5,4.3,...	low	low
[5.2,0.49,0.26,2....	low	low
+-----+-----+-----+
only showing top 5 rows
```

### 9. 평가하기

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
 labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))
```

```
Test Error = 0.307339
```

```
lrModel = model.stages[2]
trainingSummary = lrModel.summary

매 반복에서 목표 함수 구하기
objectiveHistory = trainingSummary.objectiveHistory
print("objectiveHistory:")
for objective in objectiveHistory:
```

(다음 페이지에서 계속)

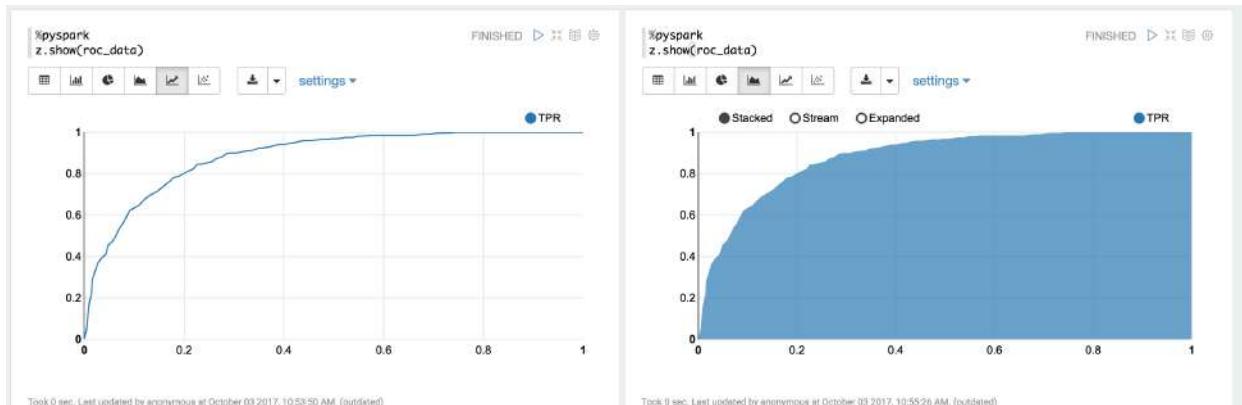
(이전 페이지에서 계속)

```
print(objective)

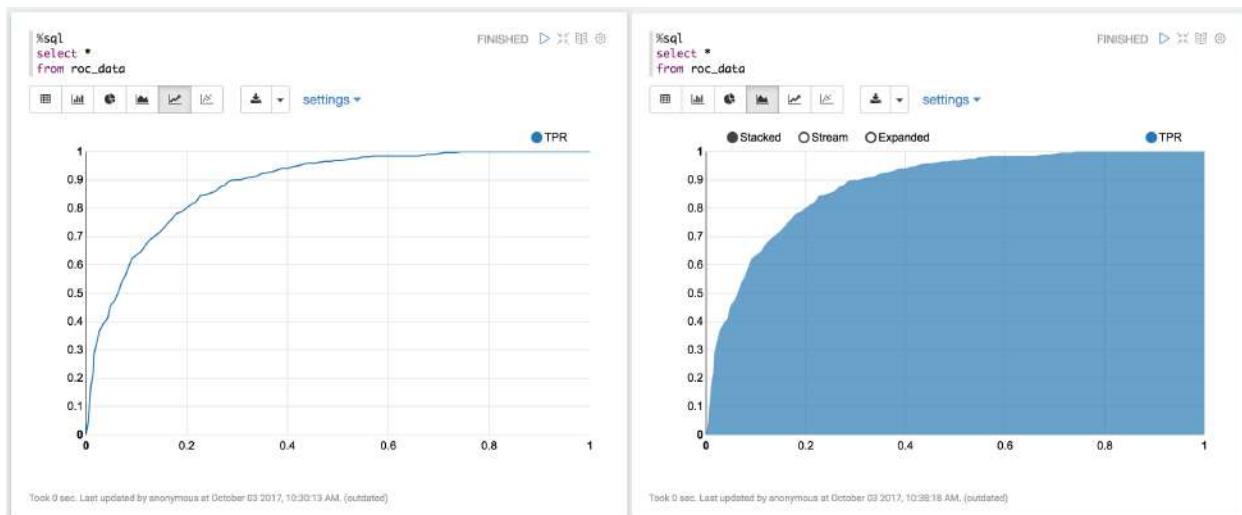
데이터 프레임과 areaUnderROC로 ROC 구하기
trainingSummary.roc.show(5)
print("areaUnderROC: " + str(trainingSummary.areaUnderROC))

F-척도를 최대화하는 임곗값을 모형으로 설정하기
fMeasure = trainingSummary.fMeasureByThreshold
maxFMeasure = fMeasure.groupBy().max('F-Measure').select('max(F-Measure)').head(5)
bestThreshold = fMeasure.where(fMeasure['F-Measure'] == maxFMeasure['max(F-
Measure)']) \
.select('threshold').head()['threshold']
lr.setThreshold(bestThreshold)
```

여러분들은 z.show()를 사용해 ROC 곡선들과 데이터를 얻을 수 있습니다.:



또한 임시 테이블 data.registerTempTable('roc\_data')을 생성하고 sql로 ROC 곡선을 그릴 수 있습니다:



## 10. 시각화하기

```

import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm, classes,
 normalize=False,
 title='Confusion matrix',
 cmap=plt.cm.Blues):

 """
 이 함수는 혼동 행렬을 출력하고 그립니다. 정규화는 'normalize=True'를
 설정하여 적용할 수 있습니다.
 """

 if normalize:
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
 print("Normalized confusion matrix")
 else:
 print('Confusion matrix, without normalization')

 print(cm)

 plt.imshow(cm, interpolation='nearest', cmap=cmap)
 plt.title(title)
 plt.colorbar()
 tick_marks = np.arange(len(classes))
 plt.xticks(tick_marks, classes, rotation=45)
 plt.yticks(tick_marks, classes)

 fmt = '.2f' if normalize else 'd'
 thresh = cm.max() / 2.
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
 plt.text(j, i, format(cm[i, j], fmt),
 horizontalalignment="center",
 color="white" if cm[i, j] > thresh else "black")

 plt.tight_layout()
 plt.ylabel('True label')
 plt.xlabel('Predicted label')

```

```

class_temp = predictions.select("label").groupBy("label") \
 .count().sort('count', ascending=False).toPandas()
class_temp = class_temp["label"].values.tolist()
class_names = map(str, class_temp)
print(class_name)
class_names

```

```
['low', 'high']
```

```

from sklearn.metrics import confusion_matrix
y_true = predictions.select("label")
y_true = y_true.toPandas()

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

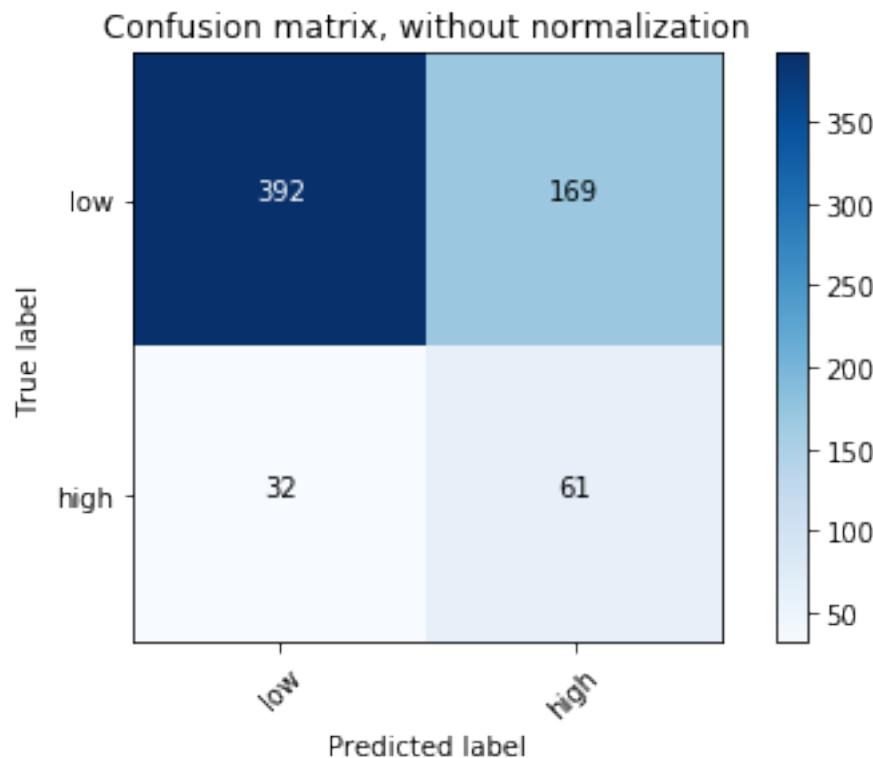
```
y_pred = predictions.select("predictedLabel")
y_pred = y_pred.toPandas()

cnf_matrix = confusion_matrix(y_true, y_pred, labels=class_names)
cnf_matrix
```

```
array([[392, 169],
 [32, 61]])
```

```
비정규화된 혼동 행렬 그리기
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
 title='Confusion matrix, without normalization')
plt.show()
```

```
Confusion matrix, without normalization
[[392 169]
 [32 61]]
```



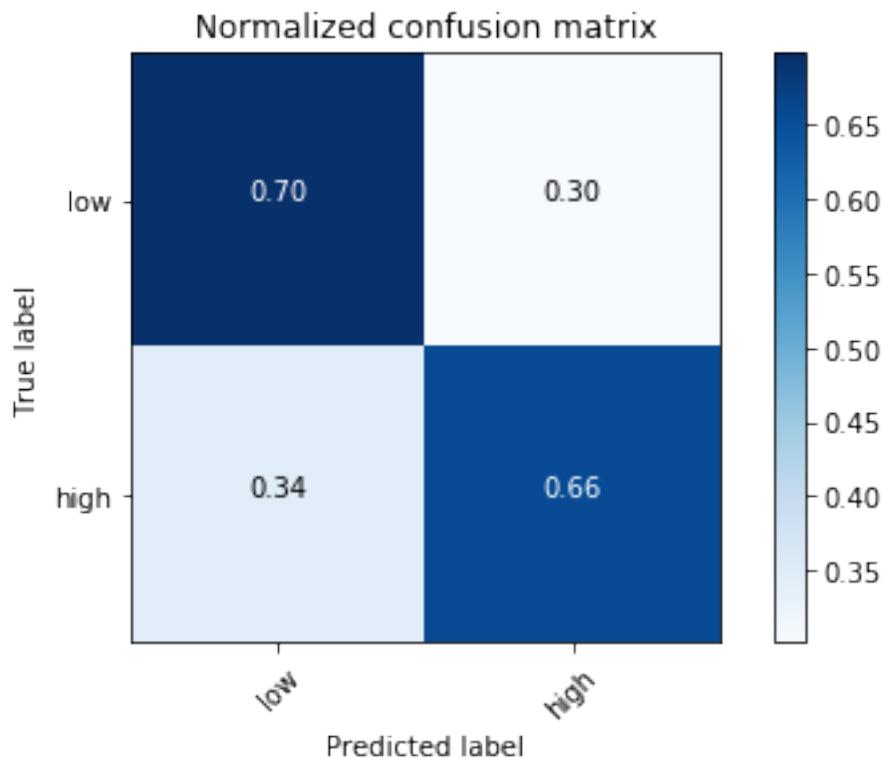
```
정규화된 혼동 행렬 그리기
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
 title='Normalized confusion matrix')
```

(다음 페이지에서 계속)

(○] 전 페이지에서 계속)

```
plt.show()
```

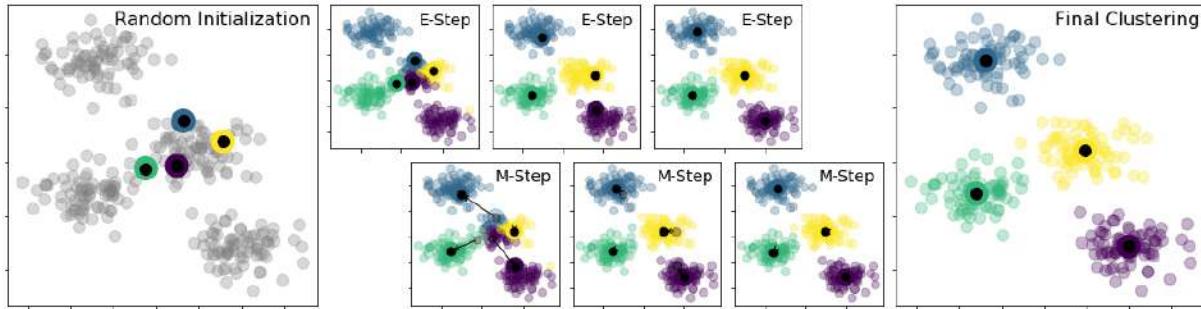
```
Normalized confusion matrix
[[0.69875223 0.30124777]
 [0.34408602 0.65591398]]
```



## 군집 분류

중국 속담

칼을 오래 가는 것은 장작을 더 쉽게 부러뜨릴 수 있게 한다 – 중국 옛 속담



위의 그림은 Python 데이터 사이언스 안내서에 나와 있는 코드로 생성한 것입니다.

## 12.1 K-Means 모델

### 12.1.1 도입

k-means 군집 분류는 원래 신호 처리에서 나온 벡터 양자화 방법 중 하나로 데이터 마이닝에서 군집 분석에 널리 사용됩니다. k-means 접근법은 기댓값 최대화(**Expectation-Maximization**) 알고리즘을 해결하는 것에 근간을 둡니다:

1. 군집의 중심을 할당합니다
2. 수렴할 때까지 다음을 반복합니다
  - E-단계: 점들을 가장 가까운 군집 중심에 할당합니다
  - M-단계: 군집 중심을 평균으로 설정합니다.

관측 집합이 주어졌을 때  $(x_1, x_2, \dots, x_m)$ . 목적 함수는 다음과 같습니다.

$$J = \sum_{i=1}^m \sum_{k=1}^K w_{ik} \|x_i - c_k\|^2$$

여기서 만약  $x_i$  가 군집  $k$ 에 속하면  $w_{ik} = 1$  이며  $c_k$  가  $x_i$  의 군집의 중심이면  $w_{ik} = 0$  입니다.

수리적으로, k-means는 두 부분을 최소화하는 문제입니다: 첫째로  $c_k$  가 고정되어 있을 때  $w_{ik}$ 에 관해  $J$ 를 최소화합니다. 그리고 나서  $w_{ik}$  가 고정되어 있을 때  $c_k$ 에 관해  $J$ 를 최소화합니다.

**E-단계:**

$$\begin{aligned} \frac{\partial J}{\partial w_{ik}} &= \sum_{i=1}^m \sum_{k=1}^K \|x_i - c_k\|^2 \\ \Rightarrow w_{ik} &= \begin{cases} 1, & \text{만약 } k = \operatorname{argmin}_j \|x_i - c_j\|^2 \\ 0, & \text{그렇지 않으면} \end{cases} \end{aligned}$$

**M-단계:**

$$\frac{\partial J}{\partial c_k} = 2 \sum_{i=1}^m m w_{ik} (x_i - c_k) = 0 \Rightarrow c_k = \frac{\sum_{i=1}^m w_{ik} x_i}{\sum_{i=1}^m w_{ik}}$$

### 12.1.2 데모

#### 1. 스파크 컨텍스트 및 스파크세션 설정하기

```
from pyspark.sql import SparkSession

spark = SparkSession \
 .builder \
 .appName("Python Spark K-means example") \
 .config("spark.some.config.option", "some-value") \
 .getOrCreate()
```

#### 2. 데이터 셋 로드하기

```
df = spark.read.format('com.databricks.spark.csv') \
 .options(header='true', \
 inferschema='true') \
 .load("../data/iris.csv", header=True);
```

데이터 셋을 확인합니다.

```
df.show(5, True)
df.printSchema()
```

여러분은 아래와 같은 결과를 얻을 수 있습니다.

```
+-----+-----+-----+-----+
| sepal_length|sepal_width|petal_length|petal_width|species|
+-----+-----+-----+-----+
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
+-----+-----+-----+-----+
only showing top 5 rows

root
|-- sepal_length: double (nullable = true)
|-- sepal_width: double (nullable = true)
|-- petal_length: double (nullable = true)
|-- petal_width: double (nullable = true)
|-- species: string (nullable = true)
```

여러분은 데이터 프레임으로부터 통계를 얻을 수 있습니다(이는 수치형 변수에만 해당됩니다).

```
df.describe().show()
```

여러분은 아래와 같은 결과를 얻을 수 있습니다.

```
+-----+-----+-----+-----+
+-----+-----+
| summary| sepal_length| sepal_width| petal_length| petal_width| species|
+-----+-----+-----+-----+
+-----+-----+
| count| 150| 150| 150| 150|
+-----+-----+
| mean| 5.843333333333335| 3.0540000000000007|3.7586666666666693|1.
+-----+-----+
| stddev|0.8280661279778637|0.43359431136217375| 1.764420419952262|0.
+-----+-----+
| min| 4.3| 2.0| 1.0| 0.1|
+-----+-----+
| max| 7.9| 4.4| 6.9| 2.5|
+-----+-----+
| |virginica|
```

### 3. 데이터를 고밀도 벡터로 변환하기(특징)

```
convert the data to dense vector
from pyspark.mllib.linalg import Vectors
def transData(data):
 return data.rdd.map(lambda r: [Vectors.dense(r[:-1])]).toDF(['features'])
```

---

노트:

여러분들께서 복잡한 데이터 셋에서 범주형 데이터를 처리하기 위해 저의 get\_dummy 함수를 사용해 보시길 적극 권장합니다.

지도 학습 버전:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols,
 labelCol):

 from pyspark.ml import Pipeline
 from pyspark.ml.feature import StringIndexer,
 OneHotEncoder, VectorAssembler
 from pyspark.sql.functions import col

 indexers = [StringIndexer(inputCol=c, outputCol="{0}_"
 .format(c))
 for c in categoricalCols]

 # 기본 설정: dropLast=True
 encoders = [OneHotEncoder(inputCol=indexer.
 .getOutputCol(),
 outputCol="{0}_encoded".format(indexer.
 .getOutputCol()))
 for indexer in indexers]

 assembler = VectorAssembler(inputCols=[encoder.
 .getOutputCol() for encoder in encoders]
 + continuousCols, outputCol=
 "features")

 pipeline = Pipeline(stages=indexers + encoders +
 [assembler])

 model=pipeline.fit(df)
 data = model.transform(df)

 data = data.withColumn('label', col(labelCol))

 return data.select(indexCol, 'features', 'label')
```

비지도학습 버전:

```
def get_dummy(df, indexCol, categoricalCols, continuousCols):
 '''

 더미 변수를 가져와 비지도 학습을 위한 연속 변수와 결합합니다.

 :param df: 데이터 프레임
 :param categoricalCols: 범주형 데이터의 이름 목록
 :param continuousCols: 수치형 데이터의 이름 목록

 :반환 k: 특징 행렬

 :저자: Wenqiang Feng
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

:이메일: von198@gmail.com
''

indexers = [StringIndexer(inputCol=c, outputCol="{0}_"
 .format(c))
 for c in categoricalCols]

기본 설정: dropLast=True
encoders = [OneHotEncoder(inputCol=indexer.
 .getOutputCol(),
 outputCol="{0}_encoded".format(indexer.
 .getOutputCol()))
 for indexer in indexers]

assembler = VectorAssembler(inputCols=[encoder.
 .getOutputCol() for encoder in encoders]
 + continuousCols, outputCol=
 "features")

pipeline = Pipeline(stages=indexers + encoders +_
 [assembler])

model=pipeline.fit(df)
data = model.transform(df)

return data.select(indexCol,'features')

```

두 가지 버전을 한 번에 적용하기:

```

def get_dummy(df,indexCol,categoricalCols,continuousCols,labelCol,
 dropLast=False):
 ...

더미 변수를 가져와 머신러닝 모델링을 위해 연속형 변수와 결합합니다.

:param df: 데이터 프레임
:param categoricalCols: 범주형 데이터의 이름 목록
:param continuousCols: 수치형 데이터의 이름 목록
:param labelCol: 라벨 열의 이름
:param dropLast: 마지막 열을 버릴 것인지를 설정하는 옵션
:반환: 특징 행렬

:저자: Wenqiang Feng
:이메일: von198@gmail.com

>>> df = spark.createDataFrame([
 (0, "a"),
 (1, "b"),
 (2, "c"),
 (3, "a"),
 (4, "a"),
 (5, "c")]

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

], ["id", "category"])

>>> indexCol = 'id'
>>> categoricalCols = ['category']
>>> continuousCols = []
>>> labelCol = []

>>> mat = get_dummy(df, indexCol, categoricalCols, continuousCols,
↪labelCol)
>>> mat.show()

>>>
+---+-----+
| id| features|
+---+-----+
0	[1.0,0.0,0.0]
1	[0.0,0.0,1.0]
2	[0.0,1.0,0.0]
3	[1.0,0.0,0.0]
4	[1.0,0.0,0.0]
5	[0.0,1.0,0.0]
+---+-----+
...
```

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
↪VectorAssembler
from pyspark.sql.functions import col

indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".
↪format(c))
             for c in categoricalCols ]

# 기본 설정: dropLast=True
encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                           outputCol="{0}_encoded".format(indexer.
↪getOutputCol()), dropLast=dropLast)
             for indexer in indexers ]

assembler = VectorAssembler(inputCols=[encoder.getOutputCol()].
↪for encoder in encoders]
                           + continuousCols, outputCol="features
↪")

pipeline = Pipeline(stages=indexers + encoders + [assembler])

model=pipeline.fit(df)
data = model.transform(df)

if indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

    return data.select(indexCol, 'features', 'label')
elif not indexCol and labelCol:
    # 지도 학습의 경우
    data = data.withColumn('label', col(labelCol))
    return data.select('features', 'label')
elif indexCol and not labelCol:
    # 비지도학습의 경우
    return data.select(indexCol, 'features')
elif not indexCol and not labelCol:
    # 비지도학습의 경우
    return data.select('features')

```

4. 데이터셋을 데이터프레임으로 변환하기

```
transformed= transData(df)
transformed.show(5, False)
```

```
+-----+
| features      |
+-----+
| [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] |
+-----+
only showing top 5 rows
```

5. 범주형 변수 처리하기

```

from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

# 범주형 특징을 자동으로 식별하고 인덱싱합니다.
# maxCategories를 지정하여 4개 이상의 고유 값을 가진 특징이 연속형으로 처리되도록 합니다. ↴

featureIndexer = VectorIndexer(inputCol="features", \
                                outputCol="indexedFeatures", \
                                maxCategories=4).fit(transformed)

data = featureIndexer.transform(transformed)

```

이제 여러분의 데이터셋을 확인해봅니다.

```
data.show(5, True)
```

여러분은 아래와 같은 결과를 얻을 수 있습니다.

```
+-----+-----+
|       features| indexedFeatures|
+-----+-----+
|[5.1,3.5,1.4,0.2]| [5.1,3.5,1.4,0.2] |
|[4.9,3.0,1.4,0.2]| [4.9,3.0,1.4,0.2] |
|[4.7,3.2,1.3,0.2]| [4.7,3.2,1.3,0.2] |
|[4.6,3.1,1.5,0.2]| [4.6,3.1,1.5,0.2] |
|[5.0,3.6,1.4,0.2]| [5.0,3.6,1.4,0.2] |
+-----+-----+
only showing top 5 rows
```

노트: k-means를 포함해서 군집 분류 알고리즘은 데이터 점들 사이 유사도를 결정하기 위해 거리 기반의 측도를 사용하기 때문에 평균이 0이고 표준편차가 1을 갖기 위해 데이터를 정규화하는 것을 강력하게 추천합니다.

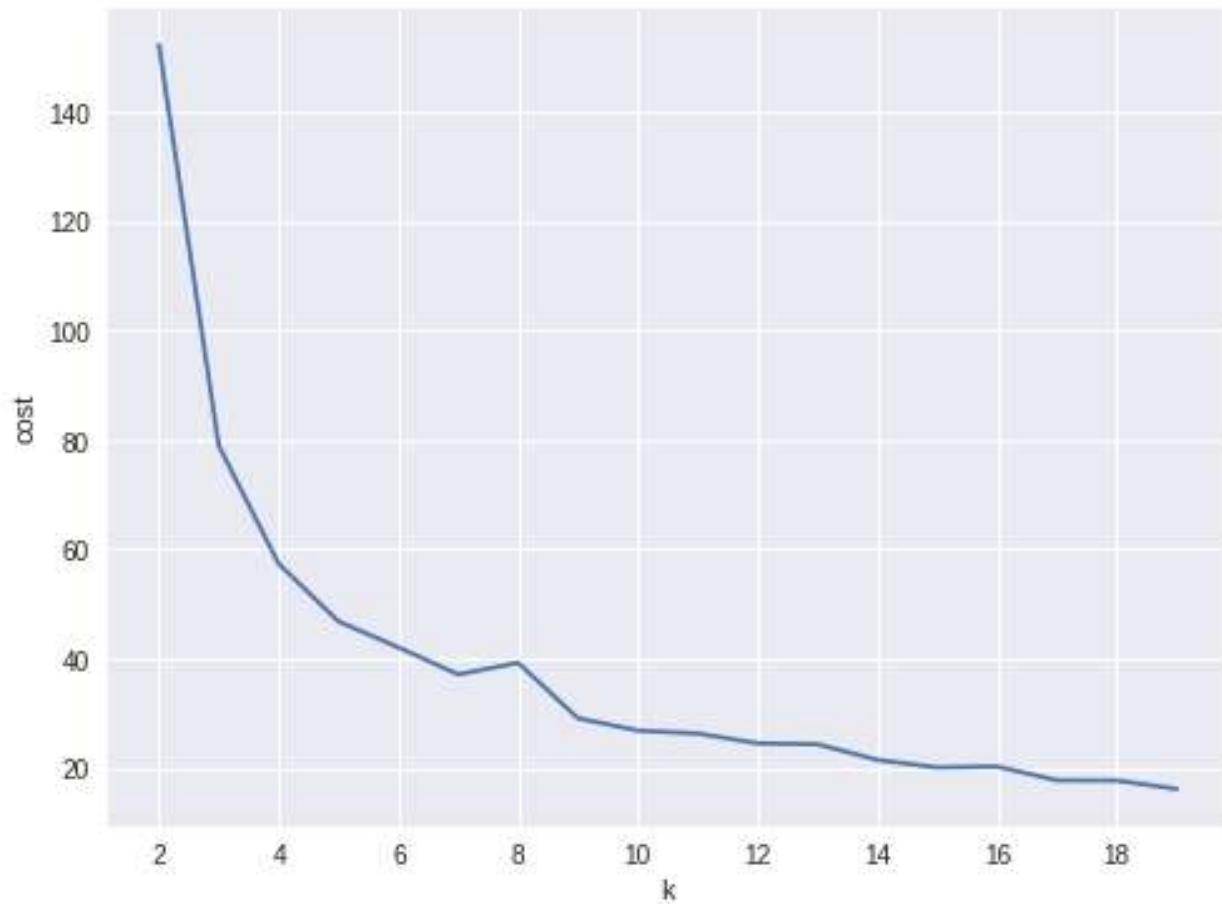
6. Elbow 방법으로 k-means 군집 분류의 최적의 군집 개수를 결정하기

```
import numpy as np
cost = np.zeros(20)
for k in range(2,20):
    kmeans = KMeans() \
        .setK(k) \
        .setSeed(1) \
        .setFeaturesCol("indexedFeatures") \
        .setPredictionCol("cluster")

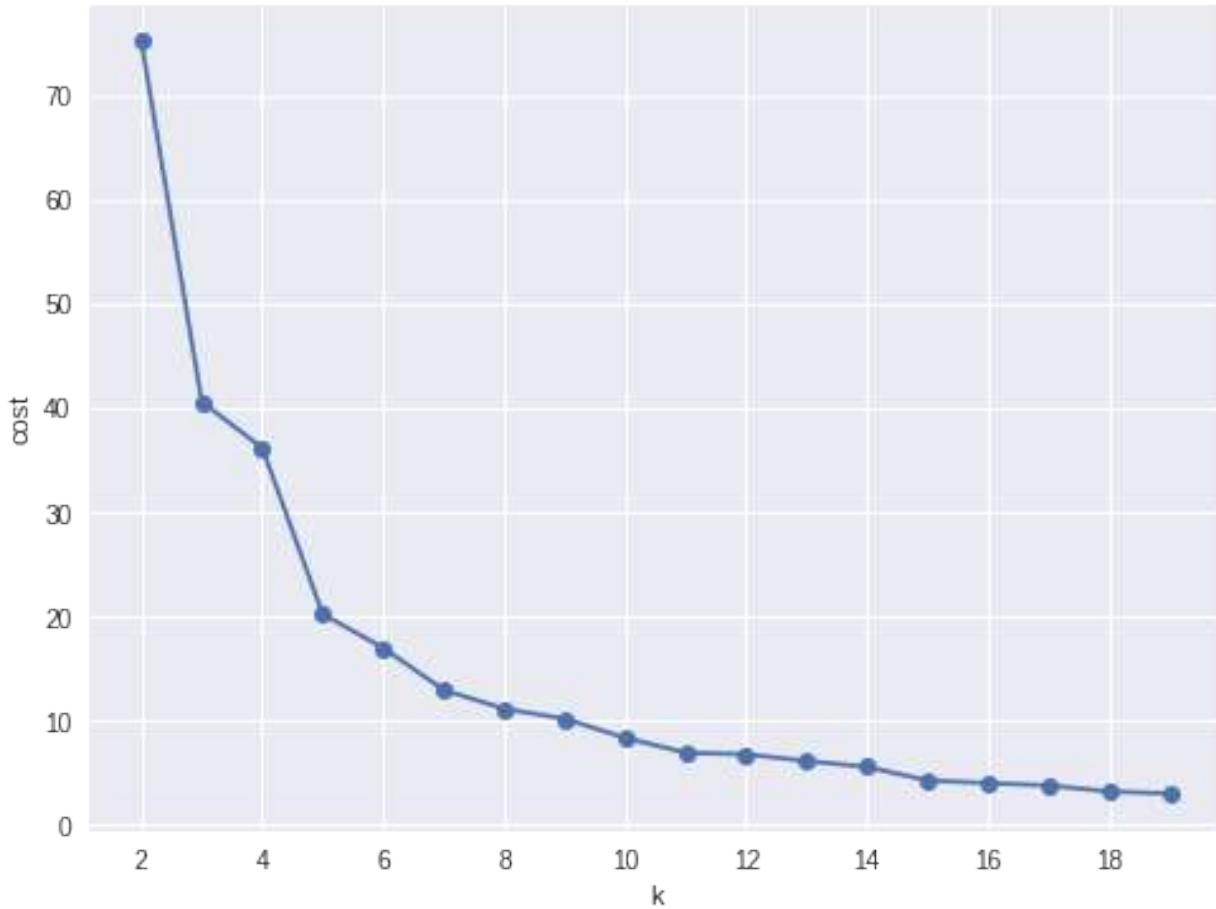
    model = kmeans.fit(data)
    cost[k] = model.computeCost(data) # Spark 2.0 혹은 그 이후 버전이 요구됩니다
```

```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.ticker import MaxNLocator

fig, ax = plt.subplots(1,1, figsize =(8,6))
ax.plot(range(2,20),cost[2:20])
ax.set_xlabel('k')
ax.set_ylabel('cost')
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()
```



개인적인 의견에선 때때로 elbow 방법으로 최적의 군집 개수를 선택하는 것이 어렵습니다. 그럼에서 보시다시피 여러분은 3, 5 혹은 8을 선택할 수 있습니다. 저는 데모에서 3을 선택하였습니다.



- 실루엣(Silhouette) 분석

```
#PySpark 라이브러리
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.sql.functions import col, percent_rank, lit
from pyspark.sql.window import Window
from pyspark.sql import DataFrame, Row
from pyspark.sql.types import StructType
from functools import reduce # For Python 3.x

from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

def optimal_k(df_in, index_col, k_min, k_max, num_runs):
    """
    Silhouette 점수 분석을 사용해 최적의 군집 개수를 결정합니다.
    :param df_in: 입력 데이터 프레임
    :param index_col: 인덱스 열의 이름
    :param k_min: 군집의 최소 개수
    :param k_max: 군집의 최대 개수
    :param num_runs: 각 고정된 군집들을 실행한 횟수
    """
    # Pipeline 구조 정의
    pipeline = Pipeline().setStages([
        StringIndexer(inputCol=index_col, outputCol='label'),
        OneHotEncoder(inputCols=['label'], outputCols=['label']),
        VectorAssembler(inputCols=['label'] + [col for col in df_in.columns if col != index_col], outputCol='features')
    ])
    # Pipeline 적용
    df_out = pipeline.fit(df_in).transform(df_in)
    # Window Function 설정
    window_spec = Window.partitionBy('label').orderBy('label')
    # 평균과 표준편차 계산
    df_out = df_out.withColumn('mean', percent_rank().over(window_spec))
    df_out = df_out.withColumn('std', lit(1.0))
    # Silhouette Score 계산
    df_out = df_out.withColumn('silhouette', (df_out['mean'] - col('mean')) / (col('std') * 2))
    # 평균 및 표준편차 계산
    df_out = df_out.groupby('label').agg(
        mean=col('silhouette').mean(),
        std=col('silhouette').std()
    )
    # 평균과 표준편차로 정규화
    df_out = df_out.withColumn('silhouette', (col('silhouette') - col('mean')) / col('std'))
    # 군집 개수별 평균 Silhouette Score 계산
    df_out = df_out.groupby('label').agg(silhouette_mean=col('silhouette').mean())
    # 평균 Silhouette Score 계산
    df_out = df_out.agg(silhouette_mean=df_out['silhouette_mean'].mean())
    # 최적의 군집 개수 결정
    optimal_k = None
    max_silhouette_mean = -float('inf')
    for k in range(k_min, k_max+1):
        df_out_k = df_out.filter(col('label') == k)
        silhouette_mean_k = df_out_k['silhouette_mean'].mean()
        if silhouette_mean_k > max_silhouette_mean:
            max_silhouette_mean = silhouette_mean_k
            optimal_k = k
    return optimal_k
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

:반환 k: 최적의 군집 개수
:반환 silh_lst: 실루엣 점수
:반환 r_table: 실행한 결과 테이블

:저자: Wenqiang Feng
:이메일: von198@gmail.com
''

start = time.time()
silh_lst = []
k_lst = np.arange(k_min, k_max+1)

r_table = df_in.select(index_col).toPandas()
r_table = r_table.set_index(index_col)
centers = pd.DataFrame()

for k in k_lst:
    silh_val = []
    for run in np.arange(1, num_runs+1):

        # k-means 모델 훈련
        kmeans = KMeans() \
            .setK(k) \
            .setSeed(int(np.random.randint(100, size=1)))
        model = kmeans.fit(df_in)

        # 예측값 생성
        predictions = model.transform(df_in)
        r_table['cluster_{k}_{run}'.format(k=k, run=run)] = predictions.
        ↪select('prediction').toPandas()

        # 실루엣 점수를 계산해 군집 분류 평가
        evaluator = ClusteringEvaluator()
        silhouette = evaluator.evaluate(predictions)
        silh_val.append(silhouette)

    silh_array=np.asarray(silh_val)
    silh_lst.append(silh_array.mean())

elapsed = time.time() - start

silhouette = pd.DataFrame(list(zip(k_lst,silh_lst)),columns = ['k',
    ↪'silhouette'])

print('+'-----+')
print("|      The finding optimal k phase took %8.0f s.      |"
    ↪%(elapsed))
print('+'-----+')

return k_lst[np.argmax(silh_lst, axis=0)], silhouette , r_table

```

```
k, silh_lst, r_table = optimal_k(scaledData, index_col, k_min, k_max, num_runs)

+-----+
|           The finding optimal k phase took      1783 s. |
+-----+
```

```
spark.createDataFrame(silh_lst).show()

+---+-----+
| k | silhouette |
+---+-----+
| 3 | 0.8045154385557953 |
| 4 | 0.6993528775512052 |
| 5 | 0.6689286654221447 |
| 6 | 0.6356184024841809 |
| 7 | 0.7174102265711756 |
| 8 | 0.6720861758298997 |
| 9 | 0.601771359881241 |
| 10 | 0.6292447334578428 |
+---+-----+
```

실루엣 목록으로부터 우리는 최적 군집 개수를 3으로 선택할 수 있습니다.

경고: pyspark.ml.evaluation의 ClusteringEvaluator는 Spark 2.4 혹은 그 이후 버전을 요구합니다!!

7. 파이프라인 아키텍처

```
from pyspark.ml.clustering import KMeans, KMeansModel

kmeans = KMeans() \
    .setK(3) \
    .setFeaturesCol("indexedFeatures") \
    .setPredictionCol("cluster")

# 파일에 인덱서와 트리 뮤기
pipeline = Pipeline(stages=[featureIndexer, kmeans])

model = pipeline.fit(transformed)

cluster = model.transform(transformed)
```

8. k-means 군집

```
cluster = model.transform(transformed)
```

```
+-----+-----+-----+
|       features | indexedFeatures | cluster |
+-----+-----+-----+
|[5.1,3.5,1.4,0.2] | [5.1,3.5,1.4,0.2] |      1 |
|[4.9,3.0,1.4,0.2] | [4.9,3.0,1.4,0.2] |      1 |
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
| [4.7,3.2,1.3,0.2] | [4.7,3.2,1.3,0.2] |    1 |
| [4.6,3.1,1.5,0.2] | [4.6,3.1,1.5,0.2] |    1 |
| [5.0,3.6,1.4,0.2] | [5.0,3.6,1.4,0.2] |    1 |
| [5.4,3.9,1.7,0.4] | [5.4,3.9,1.7,0.4] |    1 |
| [4.6,3.4,1.4,0.3] | [4.6,3.4,1.4,0.3] |    1 |
| [5.0,3.4,1.5,0.2] | [5.0,3.4,1.5,0.2] |    1 |
| [4.4,2.9,1.4,0.2] | [4.4,2.9,1.4,0.2] |    1 |
| [4.9,3.1,1.5,0.1] | [4.9,3.1,1.5,0.1] |    1 |
| [5.4,3.7,1.5,0.2] | [5.4,3.7,1.5,0.2] |    1 |
| [4.8,3.4,1.6,0.2] | [4.8,3.4,1.6,0.2] |    1 |
| [4.8,3.0,1.4,0.1] | [4.8,3.0,1.4,0.1] |    1 |
| [4.3,3.0,1.1,0.1] | [4.3,3.0,1.1,0.1] |    1 |
| [5.8,4.0,1.2,0.2] | [5.8,4.0,1.2,0.2] |    1 |
| [5.7,4.4,1.5,0.4] | [5.7,4.4,1.5,0.4] |    1 |
| [5.4,3.9,1.3,0.4] | [5.4,3.9,1.3,0.4] |    1 |
| [5.1,3.5,1.4,0.3] | [5.1,3.5,1.4,0.3] |    1 |
| [5.7,3.8,1.7,0.3] | [5.7,3.8,1.7,0.3] |    1 |
| [5.1,3.8,1.5,0.3] | [5.1,3.8,1.5,0.3] |    1 |
+-----+-----+-----+
only showing top 20 rows
```


RFM 분석

Segment	RFM	Description	Marketing
Best Customers	111	Bought most recently and most often, and spend the most	No price incentives, new products, and loyalty programs
Loyal Customers	X1X	Buy most frequently	Use R and M to further segment
Big Spenders	XX1	Spend the most	Market your most expensive products
Almost Lost	311	Haven't purchased for some time, but purchased frequently and spend the most	Aggressive price incentives
Lost Customers	411	Haven't purchased for some time, but purchased frequently and spend the most	Aggressive price incentives
Lost Cheap Customers	444	Last purchased long ago, purchased few, and spent little	Don't spend too much trying to re-acquire

위 출처는 Blast Analytics Marketing입니다.

RFM은 고객의 가치를 분석하는 방법 중 하나입니다. 이것은 흔히 데이터베이스와 직접 마케팅에 사용되며 유통업과 전문 서비스 산업분야에서 주목받고 있습니다. 더 자세한 내용은 위

키디아 RFM_wikipedia를 참조하시길 바랍니다.

RFM은 다음과 같은 세 가지 단어의 약자를 나타냅니다.

- 죄신성 (Recency) – 고객이 얼마나 최근에 구매했는가? 즉, 마지막 구매 이후 기간
- 빈도성 (Frequency) – 고객이 얼마나 자주 구매하는가? 즉, 총 구매 회수
- 규모성 (Monetary Value) – 고객이 얼마나 소비하는가? 즉, 고객이 지출한 총 금액

13.1 RFM 분석 기법

RFM 분석은 다음과 같이 세 가지 주요 단계가 있습니다:

13.1.1 1. 각 고객에 대한 RFM 특징 행렬을 만듭니다

CustomerID	Recency	Frequency	Monetary
14911	1	248	132572.62
12748	0	224	29072.1
17841	1	169	40340.78
14606	1	128	11713.85
15311	0	118	59419.34

only showing top 5 rows

13.1.2 2. 각 특징의 절삭점을(cutting points) 결정합니다

CustomerID	Recency	Frequency	Monetary	r_seg	f_seg	m_seg
17420	50	3	598.83	2	3	2
16861	59	3	151.65	3	3	1
16503	106	5	1421.43	3	2	3
15727	16	7	5178.96	1	1	4
17389	0	43	31300.08	1	1	4

only showing top 5 rows

13.1.3 3. RFM 점수를 결정하고 이와 대응되는 비즈니스 가치를 요약합니다

CustomerID	Recency	Frequency	Monetary	r_seg	f_seg	m_seg	RFMScore
17988	11	8	191.17	1	1	1	111
16892	1	7	496.84	1	1	2	112
16668	15	6	306.72	1	1	2	112
16554	3	7	641.55	1	1	2	112
16500	4	6	400.86	1	1	2	112

only showing top 5 rows

비즈니스 설명과 마케팅 가치는 다음과 같습니다:

Segment	RFM	Description	Marketing
Best Customers	111	Bought most recently and most often, and spend the most	No price incentives, new products, and loyalty programs
Loyal Customers	X1X	Buy most frequently	Use R and M to further segment
Big Spenders	XX1	Spend the most	Market your most expensive products
Almost Lost	311	Haven't purchased for some time, but purchased frequently and spend the most	Aggressive price incentives
Lost Customers	411	Haven't purchased for some time, but purchased frequently and spend the most	Aggressive price incentives
Lost Cheap Customers	444	Last purchased long ago, purchased few, and spent little	Don't spend too much trying to re-acquire

그림. 1: 출처: Blast Analytics Marketing

13.2 데모

- 주피터 노트북은 데이터 탐색에서 다운로드할 수 있습니다.
- 데이터는 German Credit에서 다운로드 할 수 있습니다.

13.2.1 데이터 읽기 및 정리

1. 스파크 컨텍스트 및 스파크세션 설정하기

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark RFM example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. 데이터 셋 로드하기

```
df_raw = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', \
             inferschema='true') \
    .load("Online Retail.csv", header=True);
```

데이터 셋을 확인합니다.

```
df_raw.show(5)
df_raw.printSchema()
```

여러분은 아래와 같은 결과를 얻을 수 있습니다.

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
536365	85123A	WHITE HANGING HEA...	6	12/1/10 8:26	2.55		
17850	United Kingdom						
536365	71053	WHITE METAL LANTERN	6	12/1/10 8:26	3.39		
17850	United Kingdom						
536365	84406B	CREAM CUPID HEART...	8	12/1/10 8:26	2.75		
17850	United Kingdom						
536365	84029G	KNITTED UNION FLA...	6	12/1/10 8:26	3.39		
17850	United Kingdom						
536365	84029E	RED WOOLLY HOTTIE...	6	12/1/10 8:26	3.39		
17850	United Kingdom						

only showing top 5 rows

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
root
| -- InvoiceNo: string (nullable = true)
| -- StockCode: string (nullable = true)
| -- Description: string (nullable = true)
| -- Quantity: integer (nullable = true)
| -- InvoiceDate: string (nullable = true)
| -- UnitPrice: double (nullable = true)
| -- CustomerID: integer (nullable = true)
| -- Country: string (nullable = true)
```

3. 데이터를 정리하고 처리하기
4. null 값들을 확인하고 제거하기

```
from pyspark.sql.functions import count

def my_count(df_in):
    df_in.agg( *[ count(c).alias(c) for c in df_in.columns ] ).show()
```

```
my_count(df_raw)
```

```
+-----+-----+-----+-----+-----+-----+-----+
|<--+ InvoiceNo | StockCode | Description | Quantity | InvoiceDate | UnitPrice | CustomerID |
Country |-----+-----+-----+-----+-----+-----+
+-----+---+
|<----+ 541909 | 541909 | 540455 | 541909 | 541909 | 541909 | 406829 |
|<--+ 541909 |-----+-----+-----+-----+-----+-----+
+-----+---+
```

각 열 값들의 개수를 확인 결과 우리는 CustomerID 열에서 null 값을 가진 것을 알 수 있습니다. 따라서 데이터 셋에서 해당 행들을 삭제할 수 있습니다.

```
df = df_raw.dropna(how='any')
my_count(df)
```

```
+-----+-----+-----+-----+-----+-----+-----+
|<--+ InvoiceNo | StockCode | Description | Quantity | InvoiceDate | UnitPrice | CustomerID |
Country |-----+-----+-----+-----+-----+-----+
+-----+---+
|<----+ 406829 | 406829 | 406829 | 406829 | 406829 | 406829 | 406829 |
|<--+ 406829 |-----+-----+-----+-----+-----+-----+
+-----+---+
```

- InvoiceDate 처리하기

```
from pyspark.sql.functions import to_utc_timestamp, unix_timestamp, lit, \
    datediff, col

timeFmt = "MM/dd/yy HH:mm"

df = df.withColumn('NewInvoiceDate'
    , to_utc_timestamp(unix_timestamp(col('InvoiceDate'),
    timeFmt).cast('timestamp')
    , 'UTC'))
```

```
df.show(5)
```

InvoiceNo	StockCode	Description	Quantity	NewInvoiceDate
InvoiceDate	UnitPrice	CustomerID	Country	
536365	85123A	WHITE HANGING HEA...	6	12/1/10 8:26 2.55
17850	United Kingdom	2010-12-01 08:26:...	6	12/1/10 8:26 3.39
536365	71053	WHITE METAL LANTERN	8	12/1/10 8:26 2.75
17850	United Kingdom	2010-12-01 08:26:...	6	12/1/10 8:26 3.39
536365	84406B	CREAM CUPID HEART...	8	12/1/10 8:26 3.39
17850	United Kingdom	2010-12-01 08:26:...	6	12/1/10 8:26 3.39
536365	84029G	KNITTED UNION FLA...	6	12/1/10 8:26 3.39
17850	United Kingdom	2010-12-01 08:26:...	6	12/1/10 8:26 3.39
536365	84029E	RED WOOLLY HOTTIE...	6	12/1/10 8:26 3.39
17850	United Kingdom	2010-12-01 08:26:...	6	12/1/10 8:26 3.39

only showing top 5 rows

경고: spark는 데이터 포맷에 예민합니다!

- 총 가격 계산하기

```
from pyspark.sql.functions import round

df = df.withColumn('TotalPrice', round(df.Quantity * df.UnitPrice, 2))
```

- 현재 시간과 NewInvoiceDate간 시간 차 계산하기

```
from pyspark.sql.functions import mean, min, max, sum, datediff, to_date

date_max = df.select(max('NewInvoiceDate')).toPandas()
current = to_utc_timestamp(unix_timestamp(lit(str(date_max.iloc[0][0])), \
    'yy-MM-dd HH:mm').cast('timestamp'), 'UTC')

# 시간 계산하기
df = df.withColumn('Duration', datediff(lit(current), 'NewInvoiceDate'))
```

- 최신성(Recency), 빈도성(Frequency), 규모성(Monetary) 생성하기

```
recency = df.groupBy('CustomerID').agg(min('Duration').alias('Recency'))
frequency = df.groupBy('CustomerID', 'InvoiceNo').count() \
    .groupBy('CustomerID') \
    .agg(count('*').alias("Frequency"))
monetary = df.groupBy('CustomerID').agg(round(sum('TotalPrice'), 2).alias(
    'Monetary'))
rfm = recency.join(frequency, 'CustomerID', how = 'inner') \
    .join(monetary, 'CustomerID', how = 'inner')
```

```
rfm.show(5)
```

CustomerID	Recency	Frequency	Monetary
17420	50	3	598.83
16861	59	3	151.65
16503	106	5	1421.43
15727	16	7	5178.96
17389	0	43	31300.08

only showing top 5 rows

13.2.2 RFM 세분화

5. 절삭점 결정하기

이번 섹션에선 여러분은 [데이터 탐색](#) 섹션에서 다룬 기법(통계적 결과와 시각화)들을 사용해 각 속성의 절삭점을 결정할 수 있습니다. 제 의견으로 절삭점은 주로 비즈니스에 의존합니다. 여러분께서 마케팅 직원들과 상의해 보거나 절삭점들에 대해 제안하고 피드백을 받아보는 것이 더 좋은 방법일 것입니다. 저는 이번 데모에서 절삭점으로서 4분위수를 사용할 것입니다.

```
cols = ['Recency', 'Frequency', 'Monetary']
rfm.select(cols).summary().show()
```

summary	Recency	Frequency	Monetary
count	4372.0	4372.0	4372.0
mean	91.58119853613907	5.07548032936871	1898.4597003659655
stddev	100.7721393138483	9.338754163574727	8219.345141139722
min	0.0	1.0	-4287.63
max	373.0	248.0	279489.02
25%	16.0	1.0	293.36249999999995
50%	50.0	3.0	648.075
75%	143.0	5.0	1611.725

절삭점을 사용한 사용자 정의 함수:

```

def RScore(x):
    if x <= 16:
        return 1
    elif x<= 50:
        return 2
    elif x<= 143:
        return 3
    else:
        return 4

def FScore(x):
    if x <= 1:
        return 4
    elif x <= 3:
        return 3
    elif x <= 5:
        return 2
    else:
        return 1

def MScore(x):
    if x <= 293:
        return 4
    elif x <= 648:
        return 3
    elif x <= 1611:
        return 2
    else:
        return 1

from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType

R_udf = udf(lambda x: RScore(x), StringType())
F_udf = udf(lambda x: FScore(x), StringType())
M_udf = udf(lambda x: MScore(x), StringType())

```

6. RFM 세분화

```

rfm_seg = rfm.withColumn("r_seg", R_udf("Recency"))
rfm_seg = rfm_seg.withColumn("f_seg", F_udf("Frequency"))
rfm_seg = rfm_seg.withColumn("m_seg", M_udf("Monetary"))
rfm_seg.show(5)

```

CustomerID	Recency	Frequency	Monetary	r_seg	f_seg	m_seg
17420	50	3	598.83	2	3	2
16861	59	3	151.65	3	3	1
16503	106	5	1421.43	3	2	3
15727	16	7	5178.96	1	1	4
17389	0	43	31300.08	1	1	4

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
rfm_seg = rfm_seg.withColumn('RFMScore',
                             F.concat(F.col('r_seg'), F.col('f_seg'), F.col('m_
                             ↵seg'))))
rfm_seg.sort(F.col('RFMScore')).show(5)
```

```
+-----+-----+-----+-----+-----+-----+
|CustomerID|Recency|Frequency|Monetary|r_seg|f_seg|m_seg|RFMScore|
+-----+-----+-----+-----+-----+-----+
|    17988|     11|        8|   191.17|     1|     1|     1|     111|
|    16892|      1|        7|   496.84|     1|     1|     2|     112|
|    16668|     15|        6|   306.72|     1|     1|     2|     112|
|    16554|      3|        7|   641.55|     1|     1|     2|     112|
|    16500|      4|        6|   400.86|     1|     1|     2|     112|
+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

13.2.3 통계 요약

7. 통계 요약

- 간단한 요약

```
rfm_seg.groupBy('RFMScore')\
    .agg({'Recency': 'mean',
          'Frequency': 'mean',
          'Monetary': 'mean'})\
    .sort(F.col('RFMScore')).show(5)
```

```
+-----+-----+-----+-----+
|RFMScore|avg(Recency)|avg(Monetary)|avg(Frequency)|
+-----+-----+-----+-----+
|    111|      11.0|    191.17|       8.0|
|    112|       8.0|   505.9775|       7.5|
|    113|7.237113402061856|1223.3604123711339| 7.752577319587629|
|    114|6.035123966942149| 8828.888595041324|18.882231404958677|
|    121|       9.6|    207.24|       4.4|
+-----+-----+-----+-----+
only showing top 5 rows
```

- 복잡한 요약

```
grp = 'RFMScore'
num_cols = ['Recency', 'Frequency', 'Monetary']
df_input = rfm_seg

quantile_grouped = quantile_agg(df_input, grp, num_cols)
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
quantile_grouped.toPandas().to_csv(output_dir+'quantile_grouped.csv')  
  
deciles_grouped = deciles_agg(df_input,grp,num_cols)  
deciles_grouped.toPandas().to_csv(output_dir+'deciles_grouped.csv')
```

13.3 확장

여러분들은 군집 분류에서 K-means 군집 분류를 적용할 수 있습니다.

13.3.1 특징 행렬 생성하기

1. 고밀도 특징 행렬 생성하기

```
from pyspark.sql import Row  
from pyspark.ml.linalg import Vectors  
  
# 방법 1 (적은 특징에 적합):  
def transData(row):  
    #     return Row(label=row["Sales"],  
    #                 features=Vectors.dense([row["TV"],  
    #                                         row["Radio"],  
    #                                         row["Newspaper"]]))  
  
# 방법 2 (많은 특징에 적합):  
def transData(data):  
    return data.rdd.map(lambda r: [r[0], Vectors.dense(r[1:])]).toDF([  
    'CustomerID', 'rfm'])
```

```
transformed= transData(rfm)  
transformed.show(5)
```

```
+-----+-----+  
|CustomerID|      rfm|  
+-----+-----+  
| 17420| [50.0,3.0,598.83]|  
| 16861| [59.0,3.0,151.65]|  
| 16503| [106.0,5.0,1421.43]|  
| 15727| [16.0,7.0,5178.96]|  
| 17389| [0.0,43.0,31300.08]|  
+-----+-----+  
only showing top 5 rows
```

2. 특징 행렬 스케일러

```
from pyspark.ml.feature import MinMaxScaler  
  
scaler = MinMaxScaler(inputCol="rfm", \
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
        outputCol="features")
scalerModel = scaler.fit(transformed)
scaledData = scalerModel.transform(transformed)
scaledData.show(5, False)
```

```
+-----+-----+
|CustomerID | rfm           | features
+-----+-----+
|17420      | [50.0, 3.0, 598.83] | [0.13404825737265416, 0.008097165991902834, 0.
|01721938714830836] |
|16861      | [59.0, 3.0, 151.65]  | [0.1581769436997319, 0.008097165991902834, 0.
|01564357039241953] |
|16503      | [106.0, 5.0, 1421.43] | [0.28418230563002683, 0.016194331983805668, 0.
|02011814573186342] |
|15727      | [16.0, 7.0, 5178.96]  | [0.04289544235924933, 0.024291497975708502, 0.
|03335929858922501] |
|17389      | [0.0, 43.0, 31300.08] | [0.0, 0.1700404858299595, 0.12540746393334334] |
+-----+-----+
only showing top 5 rows
```

13.3.2 K-means 군집 분류

3. 최적 군집 개수 찾기

군집의 최적 개수를 정의하는 두 가지 유명한 방법을 제시하겠습니다.

- elbow 분석

```
#PySpark 라이브러리
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.sql.functions import col, percent_rank, lit
from pyspark.sql.window import Window
from pyspark.sql import DataFrame, Row
from pyspark.sql.types import StructType
from functools import reduce # For Python 3.x

from pyspark.ml.clustering import KMeans
#from pyspark.ml.evaluation import ClusteringEvaluator # Spark 2.4 혹은 그 이후 버전
#이 요구됩니다.

import numpy as np
cost = np.zeros(20)
for k in range(2,20):
    kmeans = KMeans() \
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
.setK(k) \
.setSeed(1) \
.setFeaturesCol("features") \
.setPredictionCol("cluster")

model = kmeans.fit(scaledData)
cost [k] = model.computeCost(scaledData) # Spark 2.0 혹은 그 이후 버전이 요구
#됩니다.
```

```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.ticker import MaxNLocator

fig, ax = plt.subplots(1,1, figsize =(8,6))
ax.plot(range(2,20),cost[2:20], marker = "o")
ax.set_xlabel('k')
ax.set_ylabel('cost')
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()
```

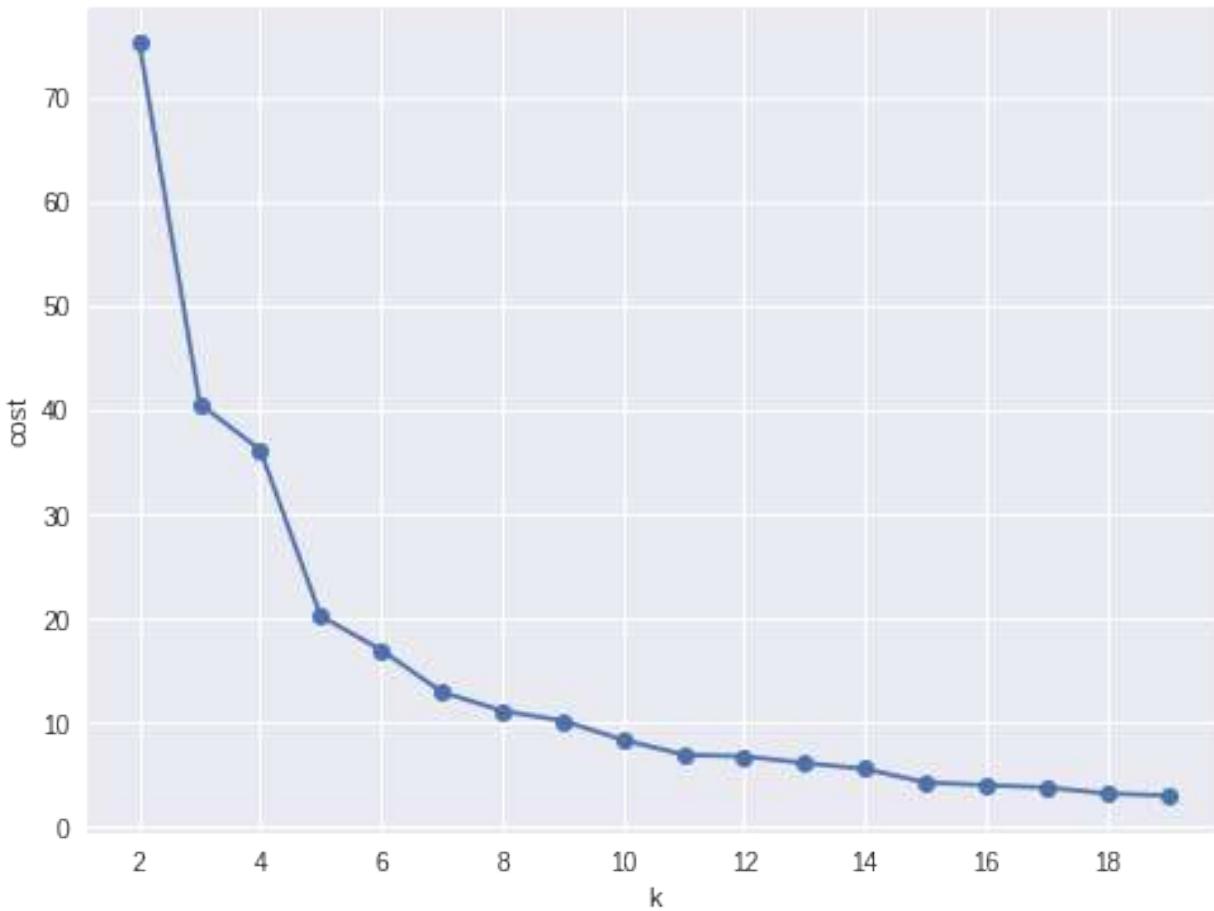


그림. 2: 비용 v.s. 군집 개수

제 생각에 때때로 군집 개수를 정하는 것은 어렵습니다. 그럼 비용 v.s. 군집 개수에서 나타나듯이 여러분들은 3, 5 혹은 8을 선택하실 수 있겠습니다. 저는 데모에서 3을 선택하겠습니다.

- 실루엣(Silhouette) 분석

```
#PySpark 라이브러리
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.sql.functions import col, percent_rank, lit
from pyspark.sql.window import Window
from pyspark.sql import DataFrame, Row
from pyspark.sql.types import StructType
from functools import reduce # For Python 3.x

from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

def optimal_k(df_in, index_col, k_min, k_max, num_runs):
    """
    실루엣 점수 분석을 통해 최적 군집 개수를 결정합니다
    """
    pass
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

:param df_in: 입력 데이터 프레임
:param index_col: 인덱스 열의 이름
:param k_min: 훈련 데이터 셋
:param k_min: 최소 군집 개수
:param k_max: 최대 군집 개수
:param num_runs: 각 고정된 군집들을 실행한 수

:반환 k: 최적 군집 개수
:반환 silh_lst: 실루엣 점수
:반환 r_table: 실행한 결과 테이블

:저자: Wenqiang Feng
:이메일: von198@gmail.com.com
'',

start = time.time()
silh_lst = []
k_lst = np.arange(k_min, k_max+1)

r_table = df_in.select(index_col).toPandas()
r_table = r_table.set_index(index_col)
centers = pd.DataFrame()

for k in k_lst:
    silh_val = []
    for run in np.arange(1, num_runs+1):

        # k-means 모델 훈련
        kmeans = KMeans() \
            .setK(k) \
            .setSeed(int(np.random.randint(100, size=1)))
        model = kmeans.fit(df_in)

        # 예측값 생성
        predictions = model.transform(df_in)
        r_table['cluster_{k}_{run}'.format(k=k, run=run)] = predictions.
        ↪select('prediction').toPandas()

        # 실루엣 점수를 계산하여 군집 분류 평가
        evaluator = ClusteringEvaluator()
        silhouette = evaluator.evaluate(predictions)
        silh_val.append(silhouette)

    silh_array=np.asarray(silh_val)
    silh_lst.append(silh_array.mean())

elapsed = time.time() - start

silhouette = pd.DataFrame(list(zip(k_lst,silh_lst)),columns = ['k',
    ↪'silhouette'])

print ('-----+')

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
print("|
˓→% (elapsed))
    print('+'-----+')
```

```
return k_lst[np.argmax(silh_lst, axis=0)], silhouette , r_table
```

```
k, silh_lst, r_table = optimal_k(scaledData,index_col,k_min, k_max,num_runs)

+-----+
|      The finding optimal k phase took   1783 s.      |
+-----+
```

```
spark.createDataFrame(silh_lst).show()
```

```
+---+-----+
|  k |      silhouette|
+---+-----+
|  3 | 0.8045154385557953|
|  4 | 0.6993528775512052|
|  5 | 0.6689286654221447|
|  6 | 0.6356184024841809|
|  7 | 0.7174102265711756|
|  8 | 0.6720861758298997|
|  9 | 0.601771359881241|
| 10 | 0.6292447334578428|
+---+-----+
```

실루엣 목록에서 우리는 최적의 군집 개수로 3을 선택할 수 있습니다.

경고: ClusteringEvaluator in pyspark.ml.evaluation 는 Spark 2.4 혹은 그 이후 버전을 요구합니다!!

4. K-means 군집 분류

```
k = 3
kmeans = KMeans().setK(k).setSeed(1)
model = kmeans.fit(scaledData)
# 예측값 생성
predictions = model.transform(scaledData)
predictions.show(5, False)
```

CustomerID	rfm	features	prediction
17420	[50.0, 3.0, 598.83]	[0.13404825737265...]	0
16861	[59.0, 3.0, 151.65]	[0.15817694369973...]	0
16503	[106.0, 5.0, 1421.43]	[0.28418230563002...]	2
15727	[16.0, 7.0, 5178.96]	[0.04289544235924...]	0
17389	[0.0, 43.0, 31300.08]	[0.0, 0.1700404858...]	0

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
+-----+-----+-----+-----+
only showing top 5 rows
```

13.3.3 통계 요약

5. 통계 요약

```
results = rfm.join(predictions.select('CustomerID', 'prediction'), 'CustomerID',
                     how='left')
results.show(5)
```

```
+-----+-----+-----+-----+
| CustomerID | Recency | Frequency | Monetary | prediction |
+-----+-----+-----+-----+
| 13098 | 1 | 41 | 28658.88 | 0 |
| 13248 | 124 | 2 | 465.68 | 2 |
| 13452 | 259 | 2 | 590.0 | 1 |
| 13460 | 29 | 2 | 183.44 | 0 |
| 13518 | 85 | 1 | 659.44 | 0 |
+-----+-----+-----+-----+
only showing top 5 rows
```

- 간단한 요약

```
results.groupBy('prediction') \
    .agg({'Recency': 'mean',
          'Frequency': 'mean',
          'Monetary': 'mean'}) \
    .sort(F.col('prediction')).show(5)
```

```
+-----+-----+-----+-----+
| prediction | avg(Recency) | avg(Monetary) | avg(Frequency) |
+-----+-----+-----+-----+
| 0 | 30.966337980278816 | 2543.0355321319284 | 6.514450867052023 |
| 1 | 296.02403846153845 | 407.16831730769206 | 1.5592948717948718 |
| 2 | 154.40148698884758 | 702.5096406443623 | 2.550185873605948 |
+-----+-----+-----+-----+
```

- 복잡한 요약

```
grp = 'RFMScore'
num_cols = ['Recency', 'Frequency', 'Monetary']
df_input = results

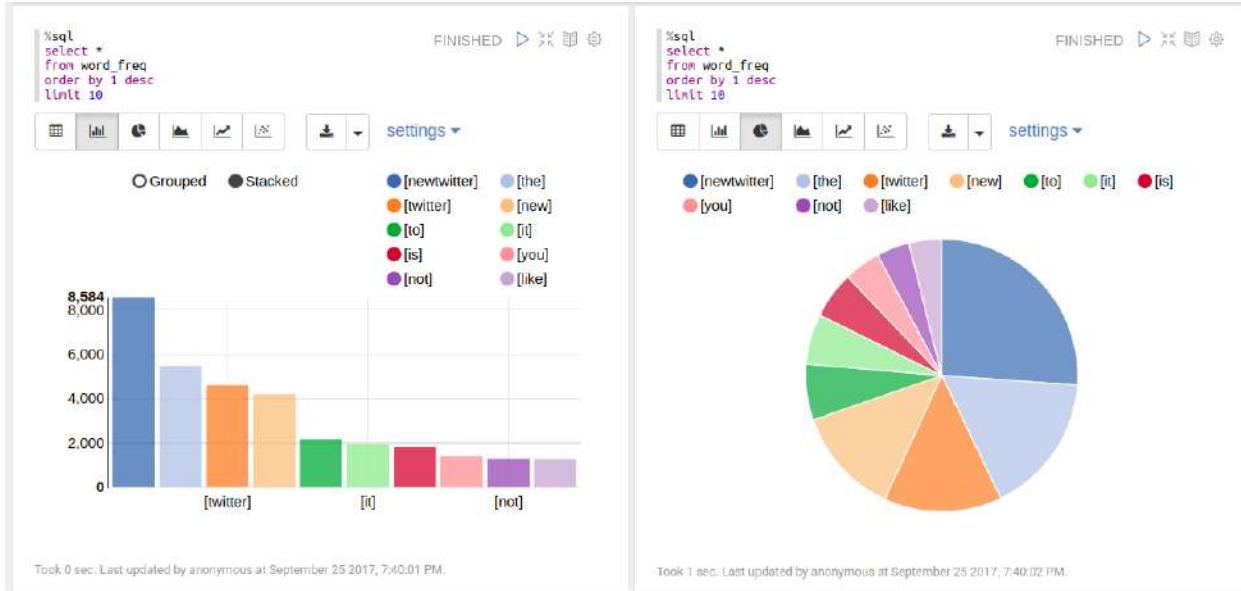
quantile_grouped = quantile_agg(df_input, grp, num_cols)
quantile_grouped.toPandas().to_csv(output_dir+'quantile_grouped.csv')

deciles_grouped = deciles_agg(df_input, grp, num_cols)
deciles_grouped.toPandas().to_csv(output_dir+'deciles_grouped.csv')
```

텍스트 마이닝

중국 속담

기사는 의도한 것 이상을 보여준다 – 양룡선 (Xianglong Shen)



14.1 텍스트 모음

14.1.1 이미지를 텍스트로

- img2txt 함수

```
def img2txt(img_dir):
    """
    이미지를 텍스트로 변환
    """
    import os, PythonMagick
```

(다음 페이지에서 계속)

(이전 페이지에서 계속)

```
from datetime import datetime
import PyPDF2

from PIL import Image
import pytesseract

f = open('doc4img.txt', 'wa')
for img in [img_file for img_file in os.listdir(img_dir)
            if (img_file.endswith('.png') or
                img_file.endswith('.jpg') or
                img_file.endswith('.jpeg'))]:
    start_time = datetime.now()

    input_img = img_dir + "/" + img

    print('-----')
    print(img)
    print('Converting ' + img + '.....')
    print('-----')

# 이미지에서 텍스트 정보 추출
text = pytesseract.image_to_string(Image.open(input_img))
print(text)

# 출력 텍스트 파일
f.write(img + "\n")
f.write(text.encode('utf-8'))

print "CPU Time for converting" + img + ":" + str(datetime.now() - start_time) + "\n"
f.write("\n-----\n")

f.close()
```

- 예제

Image 폴더의 이미지에 저의 img2txt 함수를 적용해 보겠습니다.

```
image_dir = r"Image"

img2txt(image_dir)
```

다음과 같은 결과를 얻습니다:

```
-----
feng.pdf_0.png
Converting feng.pdf_0.png.....
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
l I l w
```

Wenqiang Feng
Data Scientist
DST APPLIED ANALYTICS GROUP

Wenqiang Feng is Data Scientist **for** DST's Applied Analytics Group. Dr. Feng's responsibilities include providing DST clients with access to cutting--edge skills and technologies, including Big Data analytic solutions, advanced analytic and data enhancement techniques and modeling.

Dr. Feng has deep analytic expertise in data mining, analytic systems, machine learning algorithms, business intelligence, and applying Big Data tools to strategically solve industry problems in a cross--functional business. Before joining the DST Applied Analytics Group, Dr. Feng holds a MA Data Science Fellow at The Institute **for** Mathematics and Its Applications (IMA) at the University of Minnesota. While there, he helped startup companies make marketing decisions based on deep predictive analytics.

Dr. Feng graduated from University of Tennessee, Knoxville with PhD in Computational mathematics and Master's degree in Statistics. He also holds Master's degree in Computational Mathematics at Missouri University of Science and Technology (MST) and Master's degree in Applied Mathematics at University of science and technology of China (USTC). CPU Time **for** convertingfeng.pdf_0.png:0:00:02.061208

14.1.2 강화된 이미지를 텍스트로

- img2txt_enhance 함수

```
def img2txt_enhance(img_dir, scaler):
    """
    이미지 파일을 텍스트로 변환
    """

    import numpy as np
    import os, PythonMagick
    from datetime import datetime
    import PyPDF2
```

(다음 페이지에 계속)

(이전 페이지에 계속)

```
from PIL import Image, ImageEnhance, ImageFilter
import pytesseract

f = open('doc4img.txt', 'wa')
for img in [img_file for img_file in os.listdir(img_dir)
            if (img_file.endswith(".png") or
                img_file.endswith(".jpg") or
                img_file.endswith(".jpeg"))]:
    start_time = datetime.now()

    input_img = img_dir + "/" + img
    enhanced_img = img_dir + "/" + "Enhanced" + "/" + img

    im = Image.open(input_img) # the second one
    im = im.filter(ImageFilter.MedianFilter())
    enhancer = ImageEnhance.Contrast(im)
    im = enhancer.enhance(1)
    im = im.convert('1')
    im.save(enhanced_img)

    for scale in np.ones(scaler):
        im = Image.open(enhanced_img) # the second one
        im = im.filter(ImageFilter.MedianFilter())
        enhancer = ImageEnhance.Contrast(im)
        im = enhancer.enhance(scale)
        im = im.convert('1')
        im.save(enhanced_img)

    print('-----')
    print(img)
    print('Converting ' + img + '.....')
    print('-----')

# 이미지에서 텍스트 정보 추출
text = pytesseract.image_to_string(Image.open(enhanced_img))
print(text)

# 출력 텍스트 파일
f.write(img + "\n")
f.write(text.encode('utf-8'))

print "CPU Time for converting" + img + ":" + str(datetime.now() - start_time) + "\n"
f.write("\n-----\n")
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
f.close()
```

- 데모

Enhance 폴더의 다음과 같은 잡음이 있는 이미지에 저의 img2txt_enhance 함수를 적용해 보겠습니다.



```
image_dir = r"Enhance"  
pdf2txt_enhance(image_dir)
```

다음과 같은 결과를 얻습니다:

```
-----  
noised.jpg  
Converting noised.jpg.....  
-----  
zHHH  
CPU Time for convertingnoised.jpg:0:00:00.135465
```

반면 해당 데모에 기준 img2txt 함수를 적용하면

```
-----  
noised.jpg  
Converting noised.jpg.....  
-----  
,2 WW  
CPU Time for convertingnoised.jpg:0:00:00.133508
```

그 결과가 적절하지 않음을 확인할 수 있습니다.

14.1.3 PDF 를 텍스트로

- pdf2txt 함수

```
def pdf2txt (pdf_dir,image_dir) :  
    """  
    PDF를 텍스트로 변환  
    """  
  
    import os, PythonMagick  
    from datetime import datetime  
    import PyPDF2
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
from PIL import Image
import pytesseract

f = open('doc.txt', 'wa')
for pdf in [pdf_file for pdf_file in os.listdir(pdf_dir) if pdf_file.endswith('.pdf')]:
    start_time = datetime.now()

    input_pdf = pdf_dir + "/" + pdf

    pdf_im = PyPDF2.PdfFileReader(file(input_pdf, "rb"))
    npage = pdf_im.getNumPages()

    print('-----')
    print(pdf)
    print('Converting %d pages.' % npage)
    print('-----')

    f.write("\n-----\n")

    for p in range(npage):
        pdf_file = input_pdf + '[' + str(p) + ']'
        image_file = image_dir + "/" + pdf + '_' + str(p) + '.png'

        # PDF 파일들을 이미지들로 변환
        im = PythonMagick.Image()
        im.density('300')
        im.read(pdf_file)
        im.write(image_file)

        # 이미지들에서 텍스트 정보 추출
        text = pytesseract.image_to_string(Image.open(image_file))

        #print(text)

        # 출력 텍스트 파일
        f.write(pdf + "\n")
        f.write(text.encode('utf-8'))

    print "CPU Time for converting" + pdf + ":" + str(datetime.now() - start_time) + "\n"

f.close()
```

- 데모

pdf 폴더에 스캔한 bio pdf 파일에 저의 pdf2txt함수를 적용하겠습니다.

```
pdf_dir = r"pdf"
image_dir = r"Image"

pdf2txt(pdf_dir,image_dir)
```

다음과 같은 결과를 얻습니다:

```
-----
feng.pdf
Converting 1 pages.
-----
l l l w

Wenqiang Feng
Data Scientist
DST APPLIED ANALYTICS GROUP

Wenqiang Feng is Data Scientist for DST's Applied Analytics Group. Dr. Feng's responsibilities include providing DST clients with access to cutting--edge skills and technologies, including Big Data analytic solutions, advanced analytic and data enhancement techniques and modeling. Dr. Feng has deep analytic expertise in data mining, analytic systems, machine learning algorithms, business intelligence, and applying Big Data tools to strategically solve industry problems in a cross--functional business. Before joining the DST Applied Analytics Group, Dr. Feng holds a MA Data Science Fellow at The Institute for Mathematics and Its Applications (IMA) at the University of Minnesota. While there, he helped startup companies make marketing decisions based on deep predictive analytics.

Dr. Feng graduated from University of Tennessee, Knoxville with PhD in Computational mathematics and Master's degree in Statistics. He also holds Master's degree in Computational Mathematics at Missouri University of Science and Technology (MST) and Master's degree in Applied Mathematics at University of science and technology of China (USTC). CPU Time for convertingfeng.pdf:0:00:03.143800
```

14.1.4 오디오를 텍스트로

- audio2txt 함수

```
def audio2txt(audio_dir):
    ''' 오디오를 텍스트로 변환'''

    import speech_recognition as sr
    r = sr.Recognizer()

    f = open('doc.txt', 'w')
    for audio_n in [audio_file for audio_file in os.listdir(audio_dir) \
                    if audio_file.endswith('.wav')]:
        filename = audio_dir + "/" + audio_n

        # 오디오 데이터 읽기
        with sr.AudioFile(filename) as source:
            audio = r.record(source) # read the entire audio file

        # Google Speech Recognition
        text = r.recognize_google(audio)

        # 출력 텍스트 파일
        f.write( audio_n + ": ")
        f.write(text.encode('utf-8'))
        f.write("\n")

    print('You said: ' + text)

f.close()
```

- 데모

audio 폴더의 저의 오디오 기록들에 audio2txt 함수를 적용하겠습니다.

```
audio_dir = r"audio"

audio2txt(audio_dir)
```

다음과 같은 결과를 얻습니다:

```
You said: hello this is George welcome to my tutorial
You said: mathematics is important in daily life
You said: call me tomorrow
You said: do you want something to eat
You said: I want to speak with him
You said: nice to see you
You said: can you speak slowly
You said: have a good day
```

어쨌든, 여러분은 다음 파이썬 코드를 사용해 커맨드 라인 python record.py "demo2.wav"에 audio2txt 함수로 여러분 자신의 오디오를 녹음하고 재생해 볼 수 있습니다:

```

import sys, getopt

import speech_recognition as sr

audio_filename = sys.argv[1]

r = sr.Recognizer()
with sr.Microphone() as source:
    r.adjust_for_ambient_noise(source)
    print("Hey there, say something, I am recording!")
    audio = r.listen(source)
    print("Done listening!")

with open(audio_filename, "wb") as f:
    f.write(audio.get_wav_data())

```

14.2 텍스트 전처리

- 행에 공백만 있는지 확인하기

```

def check_blanks(data_str):
    is_blank = str(data_str.isspace())
    return is_blank

```

- 텍스트 내용의 언어가 영어인지 아닌지 확인하기: 영어 langid에 대한 올바른 정리 작업을 하고 있는지 확인하기 위해 언어를 분류하는 langid 모듈을 사용합니다.

```

import langid
def check_lang(data_str):
    predict_lang = langid.classify(data_str)
    if predict_lang[1] >= .9:
        language = predict_lang[0]
    else:
        language = 'NA'
    return language

```

- 특징들 제거하기

```

def remove_features(data_str):
    # 정규식을 컴파일 합니다
    url_re = re.compile('https?://(www.)?\w+\.\w+(/|\w+)*/?')
    punc_re = re.compile('[%s]' % re.escape(string.punctuation))
    num_re = re.compile('(\d+)')
    mention_re = re.compile('@(\w+)')
    alpha_num_re = re.compile("^[a-z0-9_.]+\$")
    # 소문자로 변환합니다
    data_str = data_str.lower()
    # 하이퍼링크를 제거합니다
    data_str = url_re.sub(' ', data_str)
    # @mentions를 제거합니다
    data_str = mention_re.sub(' ', data_str)

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
# 구두점을 제거합니다
data_str = punc_re.sub(' ', data_str)
# 하나 혹은 그 이상 연결된 숫자들을 제거합니다
data_str = num_re.sub(' ', data_str)
# a-z 0-9가 아닌 문자와 3자 미만의 단어를 제거합니다
list_pos = 0
cleaned_str = ''
for word in data_str.split():
    if list_pos == 0:
        if alpha_num_re.match(word) and len(word) > 2:
            cleaned_str = word
        else:
            cleaned_str = ' '
    else:
        if alpha_num_re.match(word) and len(word) > 2:
            cleaned_str = cleaned_str + ' ' + word
        else:
            cleaned_str += ' '
    list_pos += 1
return cleaned_str
```

- 불용어를(stop words) 제거하기

```
def remove_stops(data_str):
    # 문자열에 적용합니다
    stops = set(stopwords.words("english"))
    list_pos = 0
    cleaned_str = ''
    text = data_str.split()
    for word in text:
        if word not in stops:
            # cleaned_str를 다시 생성합니다
            if list_pos == 0:
                cleaned_str = word
            else:
                cleaned_str = cleaned_str + ' ' + word
        list_pos += 1
    return cleaned_str
```

- 텍스트 태깅하기

```
def tag_and_remove(data_str):
    cleaned_str = ''
    # 명사 태그들
    nn_tags = ['NN', 'NNP', 'NNP', 'NNPS', 'NNS']
    # 형용사
    jj_tags = ['JJ', 'JJR', 'JJS']
    # 동사
    vb_tags = ['VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ']
    nltk_tags = nn_tags + jj_tags + vb_tags
```

(다음 페이지에 계속)

(이전 페이지에 계속)

```
# 문자열을 '단어들'로 나눕니다
text = data_str.split()

# 텍스트를 태그하고 올바른 태그들만 보관합니다
tagged_text = pos_tag(text)
for tagged_word in tagged_text:
    if tagged_word[1] in nltk_tags:
        cleaned_str += tagged_word[0] + ' '

return cleaned_str
```

- 표제어 추출

```
def lemmatize(data_str):
    # 문자열에 적용합니다
    list_pos = 0
    cleaned_str = ''
    lmtzr = WordNetLemmatizer()
    text = data_str.split()
    tagged_words = pos_tag(text)
    for word in tagged_words:
        if 'v' in word[1].lower():
            lemma = lmtzr.lemmatize(word[0], pos='v')
        else:
            lemma = lmtzr.lemmatize(word[0], pos='n')
        if list_pos == 0:
            cleaned_str = lemma
        else:
            cleaned_str = cleaned_str + ' ' + lemma
        list_pos += 1
    return cleaned_str
```

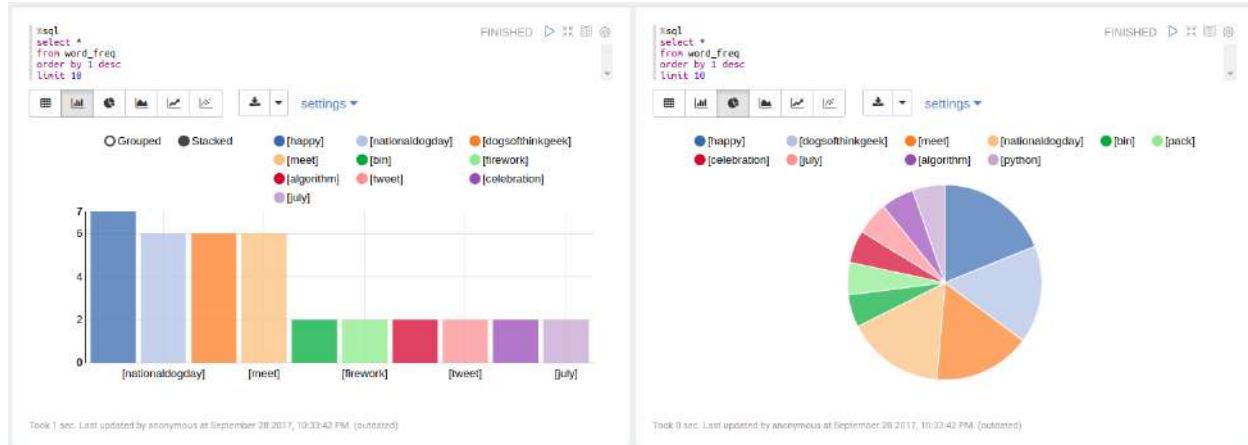
PySpark에서 전처리 함수를 정의하기

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
import preproc as pp

check_lang_udf = udf(pp.check_lang, StringType())
remove_stops_udf = udf(pp.remove_stops, StringType())
remove_features_udf = udf(pp.remove_features, StringType())
tag_and_remove_udf = udf(pp.tag_and_remove, StringType())
lemmatize_udf = udf(pp.lemmatize, StringType())
check_blanks_udf = udf(pp.check_blanks, StringType())
```

14.3 텍스트 분류

이론적으로 여러분은 분류를 하기 위해 어느 분류 알고리즘을 적용할 수도 있습니다. 다음에서 나이브 베이즈 방법에 대해서만 소개하겠습니다.



14.3.1 도입

14.3.2 테모

1. 스파크 컨텍스트를 생성하기

```
import pyspark
from pyspark.sql import SQLContext

# 스파크 컨텍스트를 생성합니다
sc = pyspark.SparkContext()
sqlContext = SQLContext(sc)
```

2. 데이터 로드하기

```
# 텍스트 파일을 로드하고 한 줄씩 하나의 행으로 변환합니다
data_rdd = sc.textFile("../data/raw_data.txt")
parts_rdd = data_rdd.map(lambda l: l.split("\t"))

# 잘못된 행들을 걸러냅니다
garantee_col_rdd = parts_rdd.filter(lambda l: len(l) == 3)
typed_rdd = guarantee_col_rdd.map(lambda p: (p[0], p[1], float(p[2])))

# 데이터 프레임을 생성합니다
data_df = sqlContext.createDataFrame(typed_rdd, ["text", "id", "label"])

# 원시 열을 가져옵니다
raw_cols = data_df.columns

#data_df.show()
data_df.printSchema()
```

```
root
| -- text: string (nullable = true)
| -- id: string (nullable = true)
| -- label: double (nullable = true)
```

	text	id	label
Fresh install of ...	1018769417	1.0	
Well. Now I know ...	10284216536	1.0	
"Literally six we...	10298589026	1.0	
Mitsubishi i MiEV...	109017669432377344	1.0	

only showing top 4 rows

3. pyspark udf 함수를 설정하기

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
import preproc as pp

# 스파크 컨텍스트로 Preproc의 모든 함수를 등록합니다
check_lang_udf = udf(pp.check_lang, StringType())
remove_stops_udf = udf(pp.remove_stops, StringType())
remove_features_udf = udf(pp.remove_features, StringType())
tag_and_remove_udf = udf(pp.tag_and_remove, StringType())
lemmatize_udf = udf(pp.lemmatize, StringType())
check_blanks_udf = udf(pp.check_blanks, StringType())
```

4. 언어 식별하기

```
lang_df = data_df.withColumn("lang", check_lang_udf(data_df["text"]))
en_df = lang_df.filter(lang_df["lang"] == "en")
en_df.show(4)
```

	text	id	label	lang
RT @goeentertain:...	665305154954989568	1.0		en
Teforia Uses Mach...	660668007975268352	1.0		en
Apple TV or Roku?	25842461136	1.0		en
Finished http://t...	9412369614	1.0		en

only showing top 4 rows

5. 불용어 제거하기

```
rm_stops_df = en_df.select(raw_cols) \
    .withColumn("stop_text", remove_stops_udf(en_df["text"]))
rm_stops_df.show(4)
```

```
+-----+-----+-----+
|       text|      id|label|      stop_text|
+-----+-----+-----+
|RT @goeentertain:...|665305154954989568| 1.0|RT @goeentertain:...|
|Teforia Uses Mach...|660668007975268352| 1.0|Teforia Uses Mach...|
| Apple TV or Roku?| 25842461136| 1.0| Apple TV Roku?|
|Finished http://t...| 9412369614| 1.0|Finished http://t...|
+-----+-----+-----+
only showing top 4 rows
```

6. 무관한 특징들 제거하기

```
rm_features_df = rm_stops_df.select(raw_cols+["stop_text"])\n    .withColumn("feat_text", \n        remove_features_udf(rm_stops_df["stop_text"]))\nrm_features_df.show(4)
```

```
+-----+-----+-----+
↔-----+
|       text|      id|label|      stop_text| ↳
↔feat_text| ↳
+-----+-----+-----+
↔-----+
|RT @goeentertain:...|665305154954989568| 1.0|RT @goeentertain:...| future ↳
↔blase ...| ↳
|Teforia Uses Mach...|660668007975268352| 1.0|Teforia Uses Mach...|teforia ↳
↔uses mach...| ↳
| Apple TV or Roku?| 25842461136| 1.0| Apple TV Roku?| ↳
↔apple roku| ↳
|Finished http://t...| 9412369614| 1.0|Finished http://t...| ↳
↔ finished| ↳
+-----+-----+-----+
↔-----+
only showing top 4 rows
```

7. 단어 태그하기

```
tagged_df = rm_features_df.select(raw_cols+["feat_text"])\n    .withColumn("tagged_text", \n        tag_and_remove_udf(rm_features_df.feat_text))\n\ntagged_df.show(4)
```

```
+-----+-----+-----+
↔-----+
|       text|      id|label|      feat_text| ↳
↔tagged_text| ↳
+-----+-----+-----+
↔-----+
|RT @goeentertain:...|665305154954989568| 1.0| future blase ...| future ↳
↔blase vic...| ↳
+-----+
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
| Teforia Uses Mach...|660668007975268352| 1.0|teforia uses mach...| teforia_
˓→uses mac...|
| Apple TV or Roku?| 25842461136| 1.0| apple roku|
˓→apple roku |
|Finished http://t...| 9412369614| 1.0| finished|
˓→ finished |
+-----+-----+-----+-----+
˓→-----+
only showing top 4 rows
```

8. 단어들의 표제어 찾기

```
lemm_df = tagged_df.select(raw_cols+["tagged_text"]) \
    .withColumn("lemm_text", lemmatize_udf(tagged_df["tagged_"
˓→text"]))
lemm_df.show(4)
```

```
+-----+-----+-----+-----+
˓→-----+
| text | id | label | tagged_text |
˓→ lemm_text |
+-----+-----+-----+-----+
˓→-----+
| RT @goeentertain:...|665305154954989568| 1.0| future blase vic...|future_
˓→blase vice...
| Teforia Uses Mach...|660668007975268352| 1.0| teforia uses mac...|teforia_
˓→use machi...
| Apple TV or Roku?| 25842461136| 1.0| apple roku |
˓→apple roku |
|Finished http://t...| 9412369614| 1.0| finished |
˓→ finish |
+-----+-----+-----+-----+
˓→-----+
only showing top 4 rows
```

9. 빈 행과 중복 개체들 제거하기

```
check_blanks_df = lemm_df.select(raw_cols+["lemm_text"]) \
    .withColumn("is_blank", check_blanks_udf(lemm_df[
˓→"lemm_text"]))
    # 빈 행들을 제거합니다
    no_blanks_df = check_blanks_df.filter(check_blanks_df["is_blank"] ==
˓→"False")

# 중복 개체들을 제거합니다
dedup_df = no_blanks_df.dropDuplicates(['text', 'label'])

dedup_df.show(4)
```

```
+-----+-----+-----+-----+
| text | id | label | lemm_text | is_blank |
+-----+-----+-----+-----+
(다음 페이지에 계속)
```

(이전 페이지에서 계속)

RT @goeentertain:...	665305154954989568	1.0 future blase vice...	False		
Teforia Uses Mach...	660668007975268352	1.0 teforia use machi...	False		
Apple TV or Roku?	25842461136	1.0	apple roku	False	
Finished http://t...	9412369614	1.0	finish	False	
+-----+-----+-----+-----+-----+					
only showing top 4 rows					

10. 유일한 ID 추가하기

```
from pyspark.sql.functions import monotonically_increasing_id
# Create Unique ID
dedup_df = dedup_df.withColumn("uid", monotonically_increasing_id())
dedup_df.show(4)
```

	text	id label	lemm_text is_blank		
	uid				
+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+
	dragon	1546813742 1.0	dragon	False	
	85899345920				
	hurt much	1558492525 1.0	hurt much		
	False 111669149696				
	seth blog word se...	383221484023709697 1.0 seth blog word se...			
	False 128849018880				
	teforia use machi...	660668007975268352 1.0 teforia use machi...			
	False 137438953472				
+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+
only showing top 4 rows					

11. 최종 데이터 셋 생성하기

```
data = dedup_df.select('uid', 'id', 'text', 'label')
data.show(4)
```

	uid	id	text label		
+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+
85899345920	1546813742	dragon 1.0			
111669149696	1558492525	hurt much 1.0			
128849018880	383221484023709697	seth blog word se... 1.0			
137438953472	660668007975268352	teforia use machi... 1.0			
+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+
only showing top 4 rows					

12. 훈련과 검증 셋 생성하기

```
# 데이터를 훈련 및 테스트 셋으로 분할합니다(40%는 테스트 셋)
(trainingData, testData) = data.randomSplit([0.6, 0.4])
```

13. NaiveBayes 파이프라인

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer
from pyspark.ml import Pipeline
from pyspark.ml.classification import NaiveBayes, RandomForestClassifier
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
from pyspark.ml.feature import CountVectorizer

# 트리 단계로 구성된 ML 파이프라인 구성: tokenizer, hashingTF, 및 nb.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol=
˓→"rawFeatures")
# vectorizer = CountVectorizer(inputCol= "words", outputCol="rawFeatures")
idf = IDF(minDocFreq=3, inputCol="rawFeatures", outputCol="features")

# 나이브 베이즈 모델
nb = NaiveBayes()

# 파이프라인 아키텍처
pipeline = Pipeline(stages=[tokenizer, hashingTF, idf, nb])

# 모델을 훈련합니다. 인덱서들도 같이 실행됩니다
model = pipeline.fit(trainingData)
```

14. 예측값 만들기

```
predictions = model.transform(testData)

# 표시할 예제 행을 선택합니다
predictions.select("text", "label", "prediction").show(5, False)
```

text	label	prediction
finish	1.0	1.0
meet rolo dogsofthinkgeek happy nationaldogday	1.0	1.0
pumpkin family	1.0	1.0
meet jet dogsofthinkgeek happy nationaldogday	1.0	1.0
meet vixie dogsofthinkgeek happy nationaldogday	1.0	1.0

only showing top 5 rows

15. 평가하기

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")
evaluator.evaluate(predictions)
```

0.912655971479501

14.4 감정 분석



14.4.1 도입

감정 분석(때로는 오피니언 마이닝 또는 감정 AI)은 자연어 처리, 텍스트 분석, 계산 언어학 및 생체 인식을 사용하여 정서적 상태 및 주관적 정보를 체계적으로 식별, 추출, 정량화 및 연구하는 것을 말합니다. 감정 분석은 마케팅에서 고객 서비스와 임상 의학에 이르는 예를 들어 리뷰와 설문 응답, 온라인 및 소셜 미디어 상의 고객의 소리와 헬스케어 애플리케이션에 광범위하게 적용됩니다.

일반적으로, 감정 분석은 어떤 주제에 대한 화자, 작가 또는 다른 주제의 태도 또는 문서, 상호 작용 또는 사건에 대한 전반적인 맥락적 극성 또는 감정적 반응을 결정하는 것을 목표로 합니다. 태도는 판단 또는 평가(평가 이론 참조), 정서적 상태(즉, 저자 또는 화자의 감정 상태) 또는 의도된 감정 커뮤니케이션(즉, 저자 또는 대화자가 의도한 감정 효과)일 것입니다.

오피니언 마이닝이라고도 하는 비즈니스에서의 감정 분석은 텍스트가 전달하는 톤에 따라 텍스트를 식별하고 목록화하는 과정입니다. 이는 광범위한 응용 분야를 가지고 있습니다:

- 비즈니스 인텔리전스 구축에 대한 감정 분석
- 경쟁우위를 위한 기업의 감정 분석

- 기업의 감정 분석을 통한 고객경험 향상

14.4.2 파이프라인



그림. 1: 감정 분석 파이프라인

14.4.3 데모

1. 스파크 컨텍스트 및 스파크세션 설정

```

from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark Sentiment Analysis example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
  
```

2. 데이터셋 로드하기

```

df = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', \
    inferschema='true') \
    .load("../data/newtwitter.csv", header=True);
  
```

	text	id	pubdate
10	Things Missing...	2602860537	18536
RT @ <u>NATURALBWINN</u> ...		2602850443	18536
RT @HBO24	yo the ...	2602761852	18535
Aaaaaaaaand I have...		2602738438	18535
I can I please have...		2602684185	18535

only showing top 5 rows

3. Text Preprocessing

3. 텍스트 전처리

- ASCII 문자가 아닌 것 제거하기

```

from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
  
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk import pos_tag
import string
import re

# ASCII 문자가 아닌 것 제거합니다
def strip_non_ascii(data_str):
    ''' ASCII 문자가 아닌 문자열을 반환합니다'''
    stripped = (c for c in data_str if 0 < ord(c) < 127)
    return ''.join(stripped)

# pyspark udf 함수를 설정합니다
strip_non_ascii_udf = udf(strip_non_ascii, StringType())
```

확인:

```
df = df.withColumn('text_non_ascii', strip_non_ascii_udf(df['text']))
df.show(5, True)
```

출력:

	text	id	pubdate	text_non_ascii
10	Things Missing...	2602860537	18536	Things Missing...
RT @_NATURALBWINN...		2602850443	18536	RT @_NATURALBWINN...
RT @HBO24 yo the ...		2602761852	18535	RT @HBO24 yo the ...
Aaaaaaaand I have...		2602738438	18535	Aaaaaaaand I have...
I can I please have...		2602684185	18535	I can I please have...

only showing top 5 rows

- 축약어 수정

```
# 축약어를 수정합니다
def fix_abbreviation(data_str):
    data_str = data_str.lower()
    data_str = re.sub(r'\bthat\b', 'that is', data_str)
    data_str = re.sub(r'\bive\b', 'i have', data_str)
    data_str = re.sub(r'\bim\b', 'i am', data_str)
    data_str = re.sub(r'\bya\b', 'yeah', data_str)
    data_str = re.sub(r'\bcant\b', 'can not', data_str)
    data_str = re.sub(r'\bdont\b', 'do not', data_str)
    data_str = re.sub(r'\bwont\b', 'will not', data_str)
    data_str = re.sub(r'\bid\b', 'i would', data_str)
    data_str = re.sub(r'\wtf', 'what the fuck', data_str)
    data_str = re.sub(r'\bwth\b', 'what the hell', data_str)
    data_str = re.sub(r'\br\b', 'are', data_str)
    data_str = re.sub(r'\bu\b', 'you', data_str)
    data_str = re.sub(r'\bk\b', 'OK', data_str)
    data_str = re.sub(r'\bsux\b', 'sucks', data_str)
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

data_str = re.sub(r'\bno+\b', 'no', data_str)
data_str = re.sub(r'\bcoo+\b', 'cool', data_str)
data_str = re.sub(r'rt\b', '', data_str)
data_str = data_str.strip()
return data_str

fix_abbreviation_udf = udf(fix_abbreviation, StringType())

```

확인:

```

df = df.withColumn('fixed_abbrev', fix_abbreviation_udf(df['text_non_ascii'
    ↪'])))
df.show(5, True)

```

출력:

text	id	pubdate	text_non_ascii	fixed_abbrev
10 Things Missing... 2602860537 18536 10 Things Missing... 10 things_				
missing...				
RT @_NATURALBWINN... 2602850443 18536 RT @_NATURALBWINN... @_				
naturalbwinner ...				
RT @HBO24 yo the ... 2602761852 18535 RT @HBO24 yo the ... @hbo24 yo the				
#ne.../				
Aaaaaaaaand I have... 2602738438 18535 Aaaaaaaaand I have... aaaaaaaand i_				
have...				
I can I please have... 2602684185 18535 I can I please have... can i please_				
have...				
only showing top 5 rows				

- 불필요한 특징들을 제거하기

```

def remove_features(data_str):
    # 정규식을 컴파일합니다
    url_re = re.compile('https?://(www.)?\w+\.\w+(/(\w+)*/?')
    punc_re = re.compile('[%s]' % re.escape(string.punctuation))
    num_re = re.compile('(\d+)')
    mention_re = re.compile('@(\w+)')
    alpha_num_re = re.compile("^[a-z0-9_.]+$")
    # 소문자로 변환합니다
    data_str = data_str.lower()
    # 하이퍼링크를 제거합니다
    data_str = url_re.sub(' ', data_str)
    # @mentions을 제거합니다
    data_str = mention_re.sub(' ', data_str)

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
# 구두점을 제거합니다
data_str = punc_re.sub(' ', data_str)
# 하나 혹은 그 이상 연결된 숫자들을 제거합니다
data_str = num_re.sub(' ', data_str)
# a-z 0-9 가 아닌 문자와 1자 미만의 단어를 제거합니다
list_pos = 0
cleaned_str = ''
for word in data_str.split():
    if list_pos == 0:
        if alpha_num_re.match(word) and len(word) > 1:
            cleaned_str = word
        else:
            cleaned_str = ' '
    else:
        if alpha_num_re.match(word) and len(word) > 1:
            cleaned_str = cleaned_str + ' ' + word
        else:
            cleaned_str += ' '
    list_pos += 1
# 원치 않는 스페이스를 제거합니다, *.split()가 공백을 자동으로 분할하고
# 중복을 삭제합니다. " ".join()가 결과 목록을 하나의 문자열로 결합합니다.

return " ".join(cleaned_str.split())
# pyspark udf 함수를 설정합니다
remove_features_udf = udf(remove_features, StringType())
```

확인:

```
df = df.withColumn('removed', remove_features_udf(df['fixed_abbrev']))
df.show(5, True)
```

출력:

```
+-----+-----+-----+-----+
|          text|      id|pubdate|      text_non_asci|      fixed_
|abbrev|           removed|
+-----+-----+-----+-----+
| 10 Things Missing...|2602860537| 18536|10 Things Missing...|10 things_
|missing...|things missing in...|
|RT @_NATURALBWINN...|2602850443| 18536|RT @_NATURALBWINN...|@_
|naturalbwinner ...|oh and do not lik...|
|RT @HBO24 yo the ...|2602761852| 18535|RT @HBO24 yo the ...|@hbo24 yo the
|#ne.../yo the newtwitter.../
|Aaaaaaaaand I have...|2602738438| 18535|Aaaaaaaaand I have...|aaaaaaaaand i_
|have...|aaaaaaaaand have t...|
|can I please have...|2602684185| 18535|can I please have...|can i please_
|have...|can please have t...|
+-----+-----+-----+-----+
|only showing top 5 rows
```

4. 감정 분석 주요 함수

```
from pyspark.sql.types import FloatType

from textblob import TextBlob

def sentiment_analysis(text):
    return TextBlob(text).sentiment.polarity

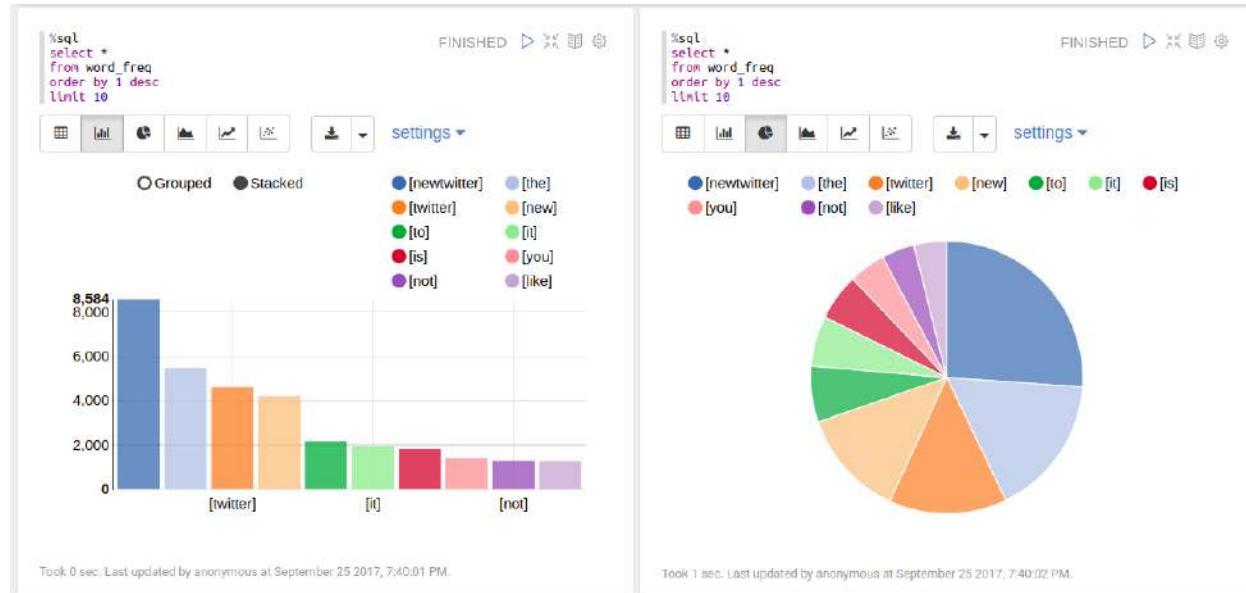
sentiment_analysis_udf = udf(sentiment_analysis, FloatType())
```

```
df = df.withColumn("sentiment_score", sentiment_analysis_udf(df['removed']))
df.show(5, True)
```

- 감정 점수(Sentiment score)

```
+-----+-----+
|       removed|sentiment_score|
+-----+-----+
|things missing in...| -0.03181818 |
|oh and do not lik...| -0.03181818 |
|yo the newtwitter...|  0.3181818 |
|aaaaaaaaand have t...|  0.11818182 |
|can please have t...|  0.13636364 |
+-----+-----+
only showing top 5 rows
```

- 단어 빈도



- 감성 분류

```

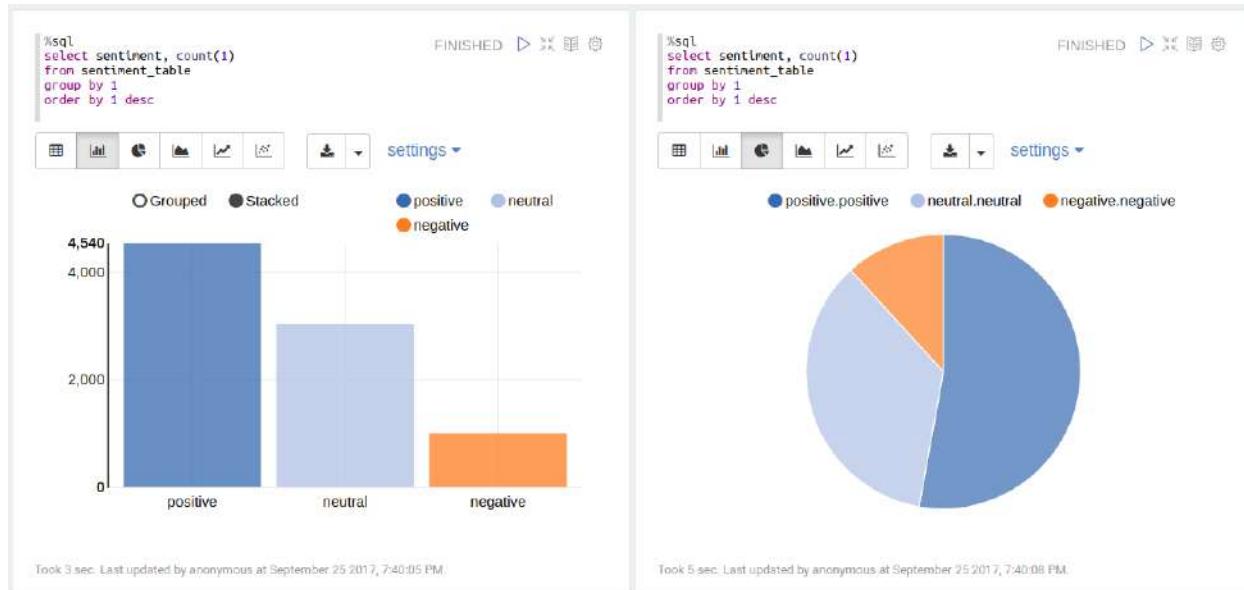
def condition(r):
    if (r >= 0.1):
        label = "positive"
    elif(r <= -0.1):
        label = "negative"
    else:
        label = "neutral"
    return label

sentiment_udf = udf(lambda x: condition(x), StringType())

```

5. 출력

- 감성 계급(class)



- 각 감성의 상위 트윗들

```

+-----+-----+-----+
|          text | sentiment_score | sentiment |
+-----+-----+-----+
| and this #newtwit... /      1.0 | positive | |
| "RT @SarahsJokes:... |      1.0 | positive |
| #newtwitter using... /      1.0 | positive |
| The #NewTwitter h... /      1.0 | positive |
| You can now undo ... |      1.0 | positive |
+-----+-----+-----+
only showing top 5 rows

```

```

+-----+-----+-----+
|          text | sentiment_score | sentiment |
+-----+-----+-----+
| Lists on #NewTwit... /      -0.1 | neutral |
| Too bad most of m... |      -0.1 | neutral |

```

(다음 페이지에 계속)

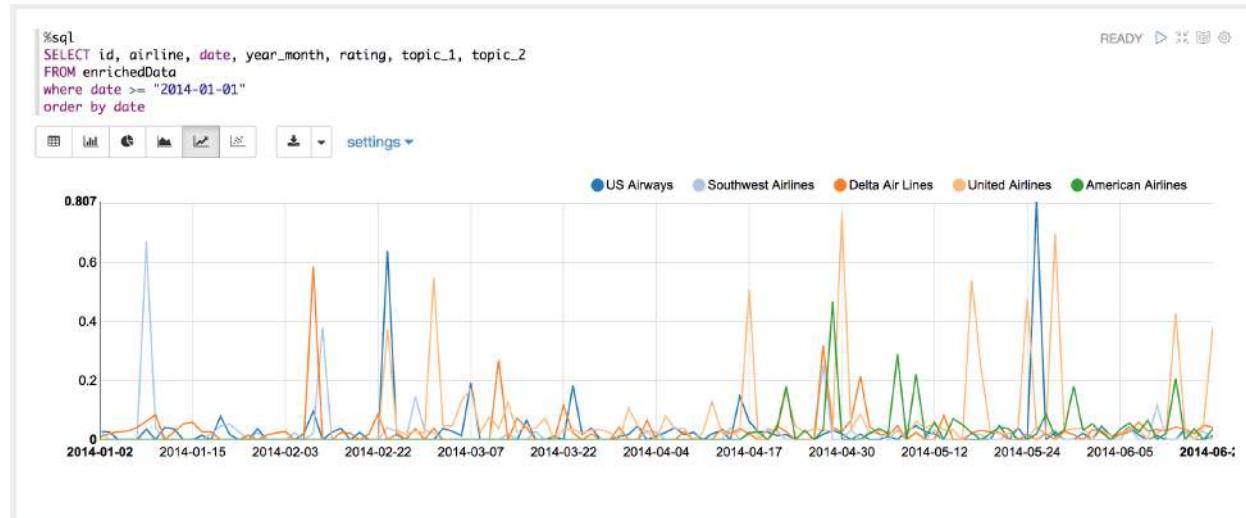
(이전 페이지에 계속)

```
|the #newtwitter i...| -0.1| neutral|
|Looks like our re...| -0.1| neutral|
|i switched to the...| -0.1| neutral|
+-----+-----+-----+
only showing top 5 rows
```

```
+-----+-----+-----+
|           text | sentiment_score | sentiment |
+-----+-----+-----+
|oh. #newtwitter i...| -1.0| negative|
|RT @chqwn: #NewTw...| -1.0| negative|
|Copy that - its W...| -1.0| negative|
|RT @chqwn: #NewTw...| -1.0| negative|
|#NewTwitter has t...| -1.0| negative|
+-----+-----+-----+
only showing top 5 rows
```

14.5 N-grams과 상관관계

14.6 토픽 모델: Latent Dirichlet Allocation



14.6.1 도입

텍스트 마이닝에서 토픽 모델은 문서 모음에서 발생하는 추상적인 "토픽"을 발견하기 위한 비지도 학습 모델입니다.

LDA(Latent Dirichlet Allocation)는 이 두 가지를 동시에 추정하는 수학적 방법입니다: 각 토픽과 관련된 단어들의 혼합(mixture)을 찾는 동시에 각 문서를 설명하는 토픽들의 혼합을 결정합니다.

14.6.2 데모

1. 데이터 로드하기

```
rawdata = spark.read.load("../data/airlines.csv", format="csv",  
    header=True)  
rawdata.show(5)
```

id	airline	date	location	rating	review
cabin	value	recommended			
10001	Delta Air Lines	21-Jun-14	Thailand	7	Economy 4
	YES Flew Mar 30 NRT t...				
10002	Delta Air Lines	19-Jun-14	USA	0	Economy 2
	NO Flight 2463 leavi...				
10003	Delta Air Lines	18-Jun-14	USA	0	Economy 1
	NO Delta Website fro...				
10004	Delta Air Lines	17-Jun-14	USA	9	Business 4
	YES "I just returned ...				
10005	Delta Air Lines	17-Jun-14	Ecuador	7	Economy 3
	YES "Round-trip flight...				

1. 텍스트 전처리

테이블을 간결하게 유지하기 위해 다음과 같은 원시 열 이름을 사용합니다:

```
raw_cols = rawdata.columns  
raw cols
```

```
['id', 'airline', 'date', 'location', 'rating', 'cabin', 'value',
 ↵'recommended', 'review']
```

```
rawdata = rawdata.dropDuplicates(['review'])
```

```
from pyspark.sql.functions import udf, col
from pyspark.sql.types import StringType, DoubleType, DateType

from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk import pos_tag
import langid
import string
import re
```

- ASCII 문자가 아닌 것 제거하기

```
# ASCII 문자가 아닌 것 제거하기
def strip_non_ascii(data_str):
    ''' ASCII 문자가 아닌 문자열을 반환합니다 '''
    stripped = (c for c in data_str if 0 < ord(c) < 127)
    return ''.join(stripped)
```

- 빈 줄인지 아닌지 확인하기

```
# 행에 공백만 있는지 확인합니다
def check_blanks(data_str):
    is_blank = str(data_str.isspace())
    return is_blank
```

- 언어 확인하기

```
# 언어를 확인합니다(영어만 적용됩니다)
def check_lang(data_str):
    from langid.langid import LanguageIdentifier, model
    identifier = LanguageIdentifier.from_modelstring(model, norm_
→probs=True)
    predict_lang = identifier.classify(data_str)

    if predict_lang[1] >= .9:
        language = predict_lang[0]
    else:
        language = predict_lang[0]
    return language
```

- 축약어 수정

```
# 축약어를 수정합니다
def fix_abbreviation(data_str):
    data_str = data_str.lower()
    data_str = re.sub(r'\bthats\b', 'that is', data_str)
    data_str = re.sub(r'\bive\b', 'i have', data_str)
    data_str = re.sub(r'\bim\b', 'i am', data_str)
    data_str = re.sub(r'\bya\b', 'yeah', data_str)
    data_str = re.sub(r'\bcant\b', 'can not', data_str)
    data_str = re.sub(r'\bdont\b', 'do not', data_str)
    data_str = re.sub(r'\bwont\b', 'will not', data_str)
    data_str = re.sub(r'\bid\b', 'i would', data_str)
    data_str = re.sub(r'wtf', 'what the fuck', data_str)
    data_str = re.sub(r'\bwth\b', 'what the hell', data_str)
    data_str = re.sub(r'\br\b', 'are', data_str)
    data_str = re.sub(r'\bu\b', 'you', data_str)
    data_str = re.sub(r'\bk\b', 'OK', data_str)
    data_str = re.sub(r'\bsux\b', 'sucks', data_str)
    data_str = re.sub(r'\bno+\b', 'no', data_str)
    data_str = re.sub(r'\bcoo+\b', 'cool', data_str)
    data_str = re.sub(r'\rt\b', '', data_str)
    data_str = data_str.strip()
    return data_str
```

파이썬으로 아파치 스파크 학습하기

- 불필요한 특징들 제거하기

```
# 불필요한 특징들을 제거합니다
def remove_features(data_str):
    # 정규식을 컴파일합니다
    url_re = re.compile('https?://(www.)?\w+\.\w+(/w+)*/?')
    punc_re = re.compile('[%s]' % re.escape(string.punctuation))
    num_re = re.compile('(\d+)')
    mention_re = re.compile('@(\w+)')
    alpha_num_re = re.compile("^[a-z0-9_.]+$")
    # 소문자로 변환합니다
    data_str = data_str.lower()
    # 하이퍼링크를 제거합니다
    data_str = url_re.sub(' ', data_str)
    # @mentions을 제거합니다
    data_str = mention_re.sub(' ', data_str)
    # 구두점을 제거합니다
    data_str = punc_re.sub(' ', data_str)
    # 하나 혹은 그 이상 연결된 숫자들을 제거합니다
    data_str = num_re.sub(' ', data_str)
    # a-z 0-9가 아닌 문자와 1자 미만의 단어를 제거합니다

    list_pos = 0
    cleaned_str = ''
    for word in data_str.split():
        if list_pos == 0:
            if alpha_num_re.match(word) and len(word) > 1:
                cleaned_str = word
            else:
                cleaned_str = ' '
        else:
            if alpha_num_re.match(word) and len(word) > 1:
                cleaned_str = cleaned_str + ' ' + word
            else:
                cleaned_str += ' '
        list_pos += 1
    # 원치 않는 스페이스를 제거합니다, *.split() 가 공백을 자동으로 분할
    # 하고 중복을 삭제합니다, ".join()" 가 결과 목록을 하나의 문자열로
    # 결합합니다
    return " ".join(cleaned_str.split())
```

- 불용어 제거하기

```
# 불용어를 제거합니다
def remove_stops(data_str):
    # 문자열에 적용합니다
    stops = set(stopwords.words("english"))
    list_pos = 0
    cleaned_str = ''
    text = data_str.split()
    for word in text:
        if word not in stops:
            # cleaned_str 다시 생성합니다
```

(이전 페이지에서 계속)

(이전 페이지에서 계속)

```

if list_pos == 0:
    cleaned_str = word
else:
    cleaned_str = cleaned_str + ' ' + word
list_pos += 1
return cleaned_str

```

- 품사 태깅

```

# 품사 태깅
def tag_and_remove(data_str):
    cleaned_str = ''
    # 명사 태그들
    nn_tags = ['NN', 'NNP', 'NNP', 'NNPS', 'NNS']
    # 형용사
    jj_tags = ['JJ', 'JJR', 'JJS']
    # 동사
    vb_tags = ['VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ']
    nltk_tags = nn_tags + jj_tags + vb_tags

    # 문자열을 '단어들'로 나눕니다
    text = data_str.split()

    # 텍스트를 태그하고 올바른 태그들만 보관합니다
    tagged_text = pos_tag(text)
    for tagged_word in tagged_text:
        if tagged_word[1] in nltk_tags:
            cleaned_str += tagged_word[0] + ' '

    return cleaned_str

```

- 표제어 추출

```

# 표제어 추출
def lemmatize(data_str):
    # 문자열에 적용합니다
    list_pos = 0
    cleaned_str = ''
    lmtzr = WordNetLemmatizer()
    text = data_str.split()
    tagged_words = pos_tag(text)
    for word in tagged_words:
        if 'v' in word[1].lower():
            lemma = lmtzr.lemmatize(word[0], pos='v')
        else:
            lemma = lmtzr.lemmatize(word[0], pos='n')
        if list_pos == 0:
            cleaned_str = lemma
        else:
            cleaned_str = cleaned_str + ' ' + lemma
        list_pos += 1
    return cleaned_str

```

- pyspark udf 함수 설정하기

```
# pyspark udf 함수를 설정합니다
strip_non_ascii_udf = udf(strip_non_ascii, StringType())
check_blanks_udf = udf(check_blanks, StringType())
check_lang_udf = udf(check_lang, StringType())
fix_abbreviation_udf = udf(fix_abbreviation, StringType())
remove_stops_udf = udf(remove_stops, StringType())
remove_features_udf = udf(remove_features, StringType())
tag_and_remove_udf = udf(tag_and_remove, StringType())
lemmatize_udf = udf(lemmatize, StringType())
```

1. 텍스트 처리

- 데이터 스키마 수정하기

```
rawdata = rawdata.withColumn('rating', rawdata.rating.cast('float'))
```

```
rawdata.printSchema()
```

```
root
|--- id: string (nullable = true)
|--- airline: string (nullable = true)
|--- date: string (nullable = true)
|--- location: string (nullable = true)
|--- rating: float (nullable = true)
|--- cabin: string (nullable = true)
|--- value: string (nullable = true)
|--- recommended: string (nullable = true)
|--- review: string (nullable = true)
```

```
from datetime import datetime
from pyspark.sql.functions import col

# https://docs.python.org/2/library/datetime.html#strftime-and-
# strptime-behavior
# 21-Jun-14 <----> %d-%b-%y
to_date = udf(lambda x: datetime.strptime(x, '%d-%b-%y'), DateType())

rawdata = rawdata.withColumn('date', to_date(col('date')))
```

```
rawdata.printSchema()
```

```
root
|--- id: string (nullable = true)
|--- airline: string (nullable = true)
|--- date: date (nullable = true)
|--- location: string (nullable = true)
|--- rating: float (nullable = true)
|--- cabin: string (nullable = true)
```

(다음 페이지에 계속)

(이전 페이지에 계속)

```
|-- value: string (nullable = true)
|-- recommended: string (nullable = true)
|-- review: string (nullable = true)
```

```
rawdata.show(5)
```

	id	airline	date	location	rating	review
10551	Southwest Airlines	2013-11-06	USA	1.0	Business	2 ...
10298	US Airways	2014-03-31	UK	1.0	Business	0 ...
10564	Southwest Airlines	2013-09-06	USA	10.0	Economy	5 ...
10134	Delta Air Lines	2013-12-10	USA	8.0	Economy	4 ...
10912	United Airlines	2014-04-07	USA	3.0	Economy	1 ...

only showing top 5 rows

```
rawdata = rawdata.withColumn('non_ascii', strip_non_ascii_udf(rawdata['review']))

+-----+-----+-----+-----+-----+
| id | airline | date | location | rating | review | non_ascii |
+-----+-----+-----+-----+-----+
| 10551 | Southwest Airlines | 2013-11-06 | USA | 1.0 | Business | 2 | ... |
| 10298 | US Airways | 2014-03-31 | UK | 1.0 | Business | 0 | ... |
| 10564 | Southwest Airlines | 2013-09-06 | USA | 10.0 | Economy | 5 | ... |
| 10134 | Delta Air Lines | 2013-12-10 | USA | 8.0 | Economy | 4 | ... |
| 10912 | United Airlines | 2014-04-07 | USA | 3.0 | Economy | 1 | ... |
+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
rawdata = rawdata.select(raw_cols+['non_ascii'])\n                .withColumn('fixed_abbrev',fix_abbreviation_\n                ~udf(rawdata['non_ascii']))\n\n+-----+-----+-----+-----+-----+-----+\n| id | airline | date | location | rating | review | non_ascii |\n+-----+-----+-----+-----+-----+-----+\n| cabin | value | recommended |          |          |          |\n+-----+-----+-----+-----+-----+-----+\n| fixed_abbrev |\n+-----+-----+-----+-----+-----+-----+\n| 10551 | Southwest Airlines | 2013-11-06 | USA | 1.0 | Business | 2 |\n| NO | Flight 3246 from ... | Flight 3246 from ... | flight 3246 |\n| from ... |\n| 10298 | US Airways | 2014-03-31 | UK | 1.0 | Business | 0 |\n| NO | Flight from Manch... | Flight from Manch... | flight from |\n| manch... |\n| 10564 | Southwest Airlines | 2013-09-06 | USA | 10.0 | Economy | 5 |\n| YES | I'm Executive Pla... | I'm Executive Pla... | i'm executive |\n| pla... |\n| 10134 | Delta Air Lines | 2013-12-10 | USA | 8.0 | Economy | 4 |\n| YES | MSP-JFK-MXP and r... | MSP-JFK-MXP and r... | msp-jfk-mxp |\n| and r... |\n| 10912 | United Airlines | 2014-04-07 | USA | 3.0 | Economy | 1 |\n| NO | Worst airline I h... | Worst airline I h... | worst airline |\n| i h... |\n+-----+-----+-----+-----+-----+-----+\n|          |\n+-----+-----+-----+-----+-----+-----+\n|          |\n+-----+-----+-----+-----+-----+-----+\nonly showing top 5 rows
```

```
rawdata = rawdata.select(raw_cols+['fixed_abbrev'])\n                .withColumn('stop_text', remove_stops_udf(rawdata[\n        'fixed_abbrev']))\n\n+-----+-----+-----+-----+-----+\n| id | airline | date | location | rating | \n| cabin | value | recommended | review | fixed_abbrev | \n| stop_text | \n+-----+-----+-----+-----+-----+\n| 10551 | Southwest Airlines | 2013-11-06 | USA | 1.0 | Business | 2 |\n| NO | Flight 3246 from ... | flight 3246 from ... | flight 3246 |\n| chicago... | \n| 10298 | US Airways | 2014-03-31 | UK | 1.0 | Business | 0 |\n| NO | Flight from Manch... | flight from manch... | flight |\n| manchester... | \n
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
| 10564 | Southwest Airlines | 2013-09-06 | USA | 10.0 | Economy | 5 | ↵
↳ YES | I'm Executive Pla... | i'm executive pla... | i'm executive |
↳ pla... |
| 10134 | Delta Air Lines | 2013-12-10 | USA | 8.0 | Economy | 4 | ↵
↳ YES | MSP-JFK-MXP and r... | msp-jfk-mxp and r... | msp-jfk-mxp |
↳ retur... |
| 10912 | United Airlines | 2014-04-07 | USA | 3.0 | Economy | 1 | ↵
↳ NO | Worst airline I h... | worst airline i h... | worst airline |
↳ eve... |
+-----+-----+-----+-----+-----+-----+-----+
↳-----+
↳-----+
only showing top 5 rows
```

```
rawdata = rawdata.select(raw_cols + ['stop_text']) \
    .withColumn('feat_text', remove_features_udf(rawdata[
↳ 'stop_text']))
```

id	airline	date	location	rating	review	stop_text
	cabin value recommended feat_text					
10551	Southwest Airlines	2013-11-06	USA	1.0	Business	2 ↵ ↳ NO Flight 3246 from ... flight 3246 chica... flight ↳ chicago mi...
10298	US Airways	2014-03-31	UK	1.0	Business	0 ↵ ↳ NO Flight from Manch... flight manchester... flight ↳ manchester...
10564	Southwest Airlines	2013-09-06	USA	10.0	Economy	5 ↵ ↳ YES I'm Executive Pla... i'm executive pla... executive ↳ platinu...
10134	Delta Air Lines	2013-12-10	USA	8.0	Economy	4 ↵ ↳ YES MSP-JFK-MXP and r... msp-jfk-mxp retur... msp jfk mxp ↳ retur...
10912	United Airlines	2014-04-07	USA	3.0	Economy	1 ↵ ↳ NO Worst airline I h... worst airline eve... worst airline ↳ eve...

```
+-----+-----+-----+-----+-----+-----+
↳-----+
↳-----+
only showing top 5 rows
```

```
rawdata = rawdata.select(raw_cols + ['feat_text']) \
    .withColumn('tagged_text', tag_and_remove_ \
udf(rawdata['feat_text']))
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

<code> id </code>	<code>airline </code>	<code>date location rating </code>	<code>review </code>	<code>feat_text </code>
<code> cabin value recommended </code>				
<code> tagged_text </code>				
<code> 10551 Southwest Airlines 2013-11-06 </code>	<code>USA </code>	<code>1.0 Business </code>	<code>2 </code>	
<code> ↳ NO Flight 3246 from ... </code>	<code>flight chicago mi... </code>	<code>flight</code>		
<code> ↳ chicago mi... </code>				
<code> 10298 US Airways 2014-03-31 </code>	<code>UK </code>	<code>1.0 Business </code>	<code>0 </code>	
<code> ↳ NO Flight from Manch... </code>	<code>flight manchester... </code>	<code>flight</code>		
<code> ↳ mancheste... </code>				
<code> 10564 Southwest Airlines 2013-09-06 </code>	<code>USA </code>	<code>10.0 Economy </code>	<code>5 </code>	
<code> ↳ YES I'm Executive Pla... </code>	<code>executive platinu... </code>	<code>executive</code>		
<code> ↳ platinu... </code>				
<code> 10134 Delta Air Lines 2013-12-10 </code>	<code>USA </code>	<code>8.0 Economy </code>	<code>4 </code>	
<code> ↳ YES MSP-JFK-MXP and r... </code>	<code>msp jfk mxp retur... </code>	<code>msp jfk mxp</code>		
<code> ↳ retur... </code>				
<code> 10912 United Airlines 2014-04-07 </code>	<code>USA </code>	<code>3.0 Economy </code>	<code>1 </code>	
<code> ↳ NO Worst airline I h... </code>	<code>worst airline eve... </code>	<code>worst</code>		
<code> ↳ airline ua... </code>				
<code>+-----+-----+-----+-----+-----+</code>				
<code> ↳ -----+-----+-----+-----+-----+</code>				
<code> ↳ -----+-----+-----+-----+-----+</code>				
<code>only showing top 5 rows</code>				

```
rawdata = rawdata.select(raw_cols+['tagged_text']) \
    .withColumn('lemm_text', lemmatize_udf(rawdata['tagged_text']))
```

<code> id </code>	<code>airline </code>	<code>date location rating </code>	<code>review </code>	<code>tagged_text </code>
<code> cabin value recommended </code>				
<code> lemm_text </code>				
<code> 10551 Southwest Airlines 2013-11-06 </code>	<code>USA </code>	<code>1.0 Business </code>	<code>2 </code>	
<code> ↳ NO Flight 3246 from ... </code>	<code>flight chicago mi... </code>	<code>flight</code>		
<code> ↳ chicago mi... </code>				
<code> 10298 US Airways 2014-03-31 </code>	<code>UK </code>	<code>1.0 Business </code>	<code>0 </code>	
<code> ↳ NO Flight from Manch... </code>	<code>flight manchester... </code>	<code>flight</code>		
<code> ↳ manchester... </code>				
<code> 10564 Southwest Airlines 2013-09-06 </code>	<code>USA </code>	<code>10.0 Economy </code>	<code>5 </code>	
<code> ↳ YES I'm Executive Pla... </code>	<code>executive platinu... </code>	<code>executive</code>		
<code> ↳ platinu... </code>				

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
| 10134 | Delta Air Lines|2013-12-10|      USA|     8.0| Economy|    4 |_
↳ YES|MSP-JFK-MXP and r...| msp jfk mxp retur...|msp jfk mxp|
↳ return...
| 10912 | United Airlines|2014-04-07|      USA|     3.0| Economy|    1 |_
↳ NO|Worst airline I h...| worst airline ua...|worst airline|
↳  ual...
+-----+-----+-----+-----+-----+-----+-----+
↳ -----
↳ -----
only showing top 5 rows
```

```
rawdata = rawdata.select(raw_cols+[ 'lemm_text' ]) \
    .withColumn("is_blank", check_blanks_udf(rawdata[
↳ "lemm_text"]))
```

```
+-----+-----+-----+-----+-----+-----+
↳ -----
| id|airline|date|location|rating|_
↳ cabin|value|recommended|           review|       lemm_
↳ text|is_blank|
+-----+-----+-----+-----+-----+-----+
↳ -----
| 10551|Southwest Airlines|2013-11-06|      USA|     1.0|Business|    2 |_
↳ NO|Flight 3246 from ...|flight chicago mi...| False|
| 10298 | US Airways|2014-03-31|      UK|     1.0|Business|    0 |_
↳ NO|Flight from Manch...|flight manchester...| False|
| 10564|Southwest Airlines|2013-09-06|      USA|    10.0| Economy|    5 |_
↳ YES|I'm Executive Pla...|executive platinu...| False|
| 10134 | Delta Air Lines|2013-12-10|      USA|     8.0| Economy|    4 |_
↳ YES|MSP-JFK-MXP and r...| msp jfk mxp retur...| False|
| 10912 | United Airlines|2014-04-07|      USA|     3.0| Economy|    1 |_
↳ NO|Worst airline I h...| worst airline ual...| False|
+-----+-----+-----+-----+-----+-----+
↳ -----
only showing top 5 rows
```

```
from pyspark.sql.functions import monotonically_increasing_id
# 유일한 ID를 생성합니다
rawdata = rawdata.withColumn("uid", monotonically_increasing_id())
data = rawdata.filter(rawdata["is_blank"] == "False")
```

```
+-----+-----+-----+-----+-----+-----+
↳ -----
| id|airline|date|location|rating|_
↳ cabin|value|recommended|           review|       lemm_
↳ text|is_blank|uid|
+-----+-----+-----+-----+-----+-----+
↳ -----
| 10551|Southwest Airlines|2013-11-06|      USA|     1.0|Business|    2 |_
↳ NO|Flight 3246 from ...|flight chicago mi...| False| 0 |
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

10298 US Airways 2014-03-31 UK 1.0 Business 0 ↵
↳ NO Flight from Manch... flight manchester... False 1 ↵
10564 Southwest Airlines 2013-09-06 USA 10.0 Economy 5 ↵
↳ YES I'm Executive Pla... executive platinu... False 2 ↵
10134 Delta Air Lines 2013-12-10 USA 8.0 Economy 4 ↵
↳ YES MSP-JFK-MXP and r... msp jfk mxp return... False 3 ↵
10912 United Airlines 2014-04-07 USA 3.0 Economy 1 ↵
↳ NO Worst airline I h... worst airline ual... False 4 ↵
+-----+-----+-----+-----+-----+-----+-----+
↳-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

LDA 모델을 위한 파이프 라인

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer
from pyspark.ml import Pipeline
from pyspark.ml.classification import NaiveBayes, ↵
    RandomForestClassifier
from pyspark.ml.clustering import LDA
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.feature import IndexToString, StringIndexer, ↵
    VectorIndexer
from pyspark.ml.feature import CountVectorizer

# 트리 단계로 구성된 ML 파이프라인을 구성합니다: tokenizer, hashingTF, 및 nb.
tokenizer = Tokenizer(inputCol="lemm_text", outputCol="words")
#data = tokenizer.transform(data)
vectorizer = CountVectorizer(inputCol= "words", outputCol=
    "rawFeatures")
idf = IDF(inputCol="rawFeatures", outputCol="features")
#idfModel = idf.fit(data)

lda = LDA(k=20, seed=1, optimizer="em")

pipeline = Pipeline(stages=[tokenizer, vectorizer,idf, lda])

model = pipeline.fit(data)
```

1. 결과 제시

- 토픽

topic	termIndices	termWeights
0 [60, 7, 12, 483, ...	[0.01349507958269...	
1 [363, 29, 187, 55...	[0.01247250144447...	

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

| 2|[46, 107, 672, 27...|[0.01188684264641...
| 3|[76, 43, 285, 152...|[0.01132638300115...
| 4|[201, 13, 372, 69...|[0.01337529863256...
| 5|[122, 103, 181, 4...|[0.00930415977117...
| 6|[14, 270, 18, 74,...|[0.01253817708163...
| 7|[111, 36, 341, 10...|[0.01269584954257...
| 8|[477, 266, 297, 1...|[0.01017486869509...
| 9|[10, 73, 46, 1, 2...|[0.01050875237546...
| 10|[57, 29, 411, 10,...|[0.01777350667863...
| 11|[293, 119, 385, 4...|[0.01280305149305...
| 12|[116, 218, 256, 1...|[0.01570714218509...
| 13|[433, 171, 176, 3...|[0.00819684813575...
| 14|[74, 84, 45, 108,...|[0.01700630002172...
| 15|[669, 215, 14, 58...|[0.00779310974971...
| 16|[198, 21, 98, 164...|[0.01030577084202...
| 17|[96, 29, 569, 444...|[0.01297142577633...
| 18|[18, 60, 140, 64,...|[0.01306356985169...
| 19|[33, 178, 95, 2, ...|[0.00907425683229...
+-----+

```

- 토픽 용어

```

from pyspark.sql.types import ArrayType, StringType

def termsIdx2Term(vocabulary):
    def termsIdx2Term(termIndices):
        return [vocabulary[int(index)] for index in termIndices]
    return udf(termsIdx2Term, ArrayType(StringType()))

vectorizerModel = model.stages[1]
vocabList = vectorizerModel.vocabulary
final = ldatopics.withColumn("Terms", termsIdx2Term(vocabList) (
    "termIndices"))

```

topic	termIndices	Terms
0	[60, 7, 12, 483, 292, 326, 88, 4, 808, 32]	[pm, plane, board, kid, online, lga, schedule, get, memphis, arrive]
1	[363, 29, 187, 55, 48, 647, 30, 9, 204, 457]	[dublin, class, th, sit, entertainment, express, say, delay, dl, son]
2	[46, 107, 672, 274, 92, 539, 23, 27, 279, 8]	[economy, sfo, milwaukee, decent, comfortable, iad, return, united, average, airline]

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
|3   | [76, 43, 285, 152, 102, 34, 300, 113, 24, 31] | [didn't, pay, ▾
→ lose, different, extra, bag, mile, baggage, leave, day] ▾
→ |
|4   | [201, 13, 372, 692, 248, 62, 211, 187, 105, 110] | [houston, ▾
→ crew, heathrow, louisville, london, great, denver, th, land, jfk] ▾
→ |
|5   | [122, 103, 181, 48, 434, 10, 121, 147, 934, 169] | [lhr, serve, ▾
→ screen, entertainment, ny, delta, excellent, atl, sin, newark] ▾
→ |
|6   | [14, 270, 18, 74, 70, 37, 16, 450, 3, 20] | [check, ▾
→ employee, gate, line, change, wait, take, flt, time, tell] ▾
→ |
|7   | [111, 36, 341, 10, 320, 528, 844, 19, 195, 524] | [atlanta, ▾
→ first, toilet, delta, washington, card, global, staff, route, ▾
→ amsterdam] |
|8   | [477, 266, 297, 185, 1, 33, 22, 783, 17, 908] | [fuel, group, ▾
→ pas, boarding, seat, trip, minute, orleans, make, select] ▾
→ |
|9   | [10, 73, 46, 1, 248, 302, 213, 659, 48, 228] | [delta, lax, ▾
→ economy, seat, london, detroit, comfo, weren't, entertainment, wife] ▾
→ |
|10  | [57, 29, 411, 10, 221, 121, 661, 19, 805, 733] | [business, ▾
→ class, fra, delta, lounge, excellent, syd, staff, nov, mexico] ▾
→ |
|11  | [293, 119, 385, 481, 503, 69, 13, 87, 176, 545] | [march, ua, ▾
→ manchester, phx, envoy, drink, crew, american, aa, canada] ▾
→ |
|12  | [116, 218, 256, 156, 639, 20, 365, 18, 22, 136] | [san, clt, ▾
→ francisco, second, text, tell, captain, gate, minute, available] ▾
→ |
|13  | [433, 171, 176, 339, 429, 575, 10, 26, 474, 796] | [daughter, ▾
→ small, aa, ba, segment, proceed, delta, passenger, size, similar] ▾
→ |
|14  | [74, 84, 45, 108, 342, 111, 315, 87, 52, 4] | [line, agent, ▾
→ next, hotel, standby, atlanta, dallas, american, book, get] ▾
→ |
|15  | [669, 215, 14, 58, 561, 59, 125, 179, 93, 5] | [fit, carry, ▾
→ check, people, bathroom, ask, thing, row, don, fly] ▾
→ |
|16  | [198, 21, 98, 164, 57, 141, 345, 62, 121, 174] | [ife, good, ▾
→ nice, much, business, lot, dfw, great, excellent, carrier] ▾
→ |
|17  | [96, 29, 569, 444, 15, 568, 21, 103, 657, 505] | [phl, class, ▾
→ diego, lady, food, wheelchair, good, serve, miami, mia] ▾
→ |
|18  | [18, 60, 140, 64, 47, 40, 31, 35, 2, 123] | [gate, pm, ▾
→ phoenix, connection, cancel, connect, day, airpo, hour, charlotte] ▾
→ |
|19  | [33, 178, 95, 2, 9, 284, 42, 4, 89, 31] | [trip, ▾
→ counter, philadelphia, hour, delay, stay, way, get, southwest, ▾
→ day] |
+-----+-----+-----+
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

• LDA 결과

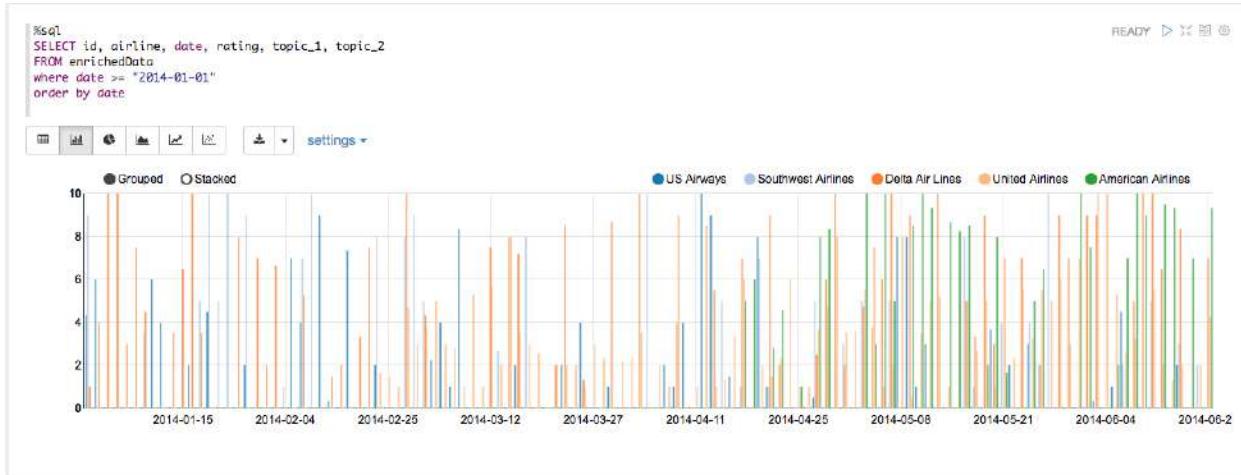
id	airline	date	cabin	rating	
words	features	topicDistribution			
10551 Southwest Airlines 2013-11-06 Business 1.0 [flight, ↵chicago, ... (4695, [0, 2, 3, 6, 11... [0.03640342580508...					
10298 US Airways 2014-03-31 Business 1.0 [flight, ↵manchest... (4695, [0, 1, 2, 6, 7, ... [0.01381306271470...					
10564 Southwest Airlines 2013-09-06 Economy 10.0 [executive, ↵plati... (4695, [0, 1, 6, 7, 11... [0.05063554352934...					
10134 Delta Air Lines 2013-12-10 Economy 8.0 [msp, jfk, ↵mfp, r... (4695, [0, 1, 3, 10, 1... [0.01494708959842...					
10912 United Airlines 2014-04-07 Economy 3.0 [worst, ↵airline, ... (4695, [0, 1, 7, 8, 13... [0.04421751181232...					
10089 Delta Air Lines 2014-02-18 Economy 2.0 [dl, mia, ↵lax, im... (4695, [2, 4, 5, 7, 8, ... [0.02158861273876...					
10385 US Airways 2013-10-21 Economy 10.0 [flew, gla, ↵phl, ... (4695, [0, 1, 3, 5, 14... [0.03343845991816...					
10249 US Airways 2014-06-17 Economy 1.0 [friend, ↵book, fl... (4695, [0, 2, 3, 4, 5, ... [0.02362432562165...					
10289 US Airways 2014-04-12 Economy 10.0 [flew, air, ↵rome, ... (4695, [0, 1, 5, 8, 13... [0.01664012816210...					
10654 Southwest Airlines 2012-07-10 Economy 8.0 [lhr, jfk, ↵think, ... (4695, [0, 4, 5, 6, 8, ... [0.01526072330297...					
10754 American Airlines 2014-05-04 Economy 10.0 [san, diego, ↵moli... (4695, [0, 2, 8, 15, 2... [0.03571177612496...					
10646 Southwest Airlines 2012-08-17 Economy 7.0 [toledo, co, ↵stop... (4695, [0, 2, 3, 4, 7, ... [0.02394775146271...					
10097 Delta Air Lines 2014-02-03 First Class 10.0 [honolulu, ↵la, fi... (4695, [0, 4, 6, 7, 13... [0.02008375619661...					
10132 Delta Air Lines 2013-12-16 Economy 7.0 [manchester, ↵uk, ... (4695, [0, 1, 2, 3, 5, ... [0.01463126146601...					
10560 Southwest Airlines 2013-09-20 Economy 9.0 [first, time, ↵sou... (4695, [0, 3, 7, 8, 9, ... [0.04934836409896...					
10579 Southwest Airlines 2013-07-25 Economy 0.0 [plane, land, ↵pm, ... (4695, [2, 3, 4, 5, 7, ... [0.06106959241722...					
10425 US Airways 2013-08-06 Economy 3.0 [airway, bad, ↵pro... (4695, [2, 3, 4, 7, 8, ... [0.01770471771322...					
10650 Southwest Airlines 2012-07-27 Economy 9.0 [flew, jfk, ↵lhr, ... (4695, [0, 1, 6, 13, 1... [0.02676226245086...					
10260 US Airways 2014-06-03 Economy 1.0 [february, ↵air, u... (4695, [0, 2, 4, 17, 2... [0.02887390875079...					
10202 Delta Air Lines 2013-09-14 Economy 10.0 [aug, lhr, ↵jfk, b... (4695, [1, 2, 4, 7, 10... [0.02377704988307...					

(다음 페이지에 계속)

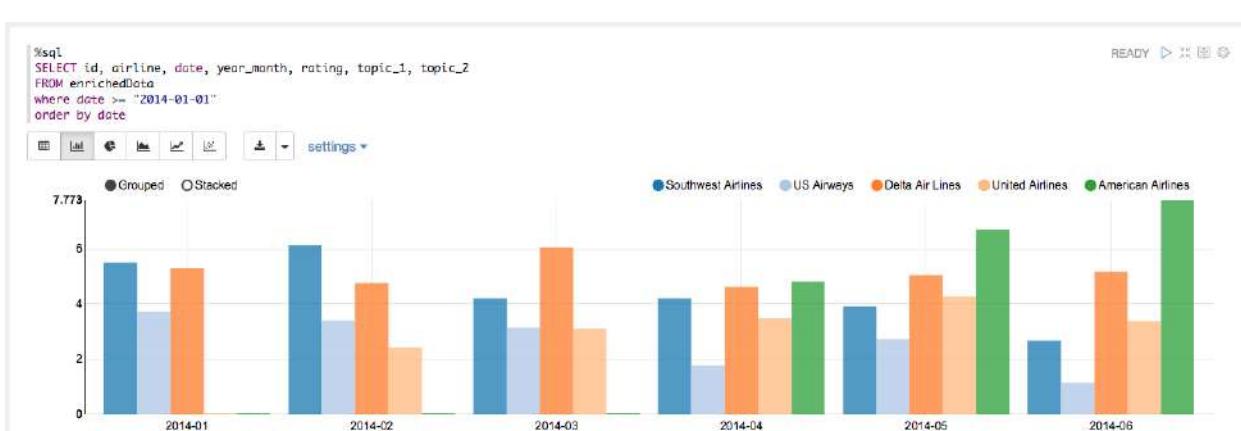
(이전 페이지에서 계속)

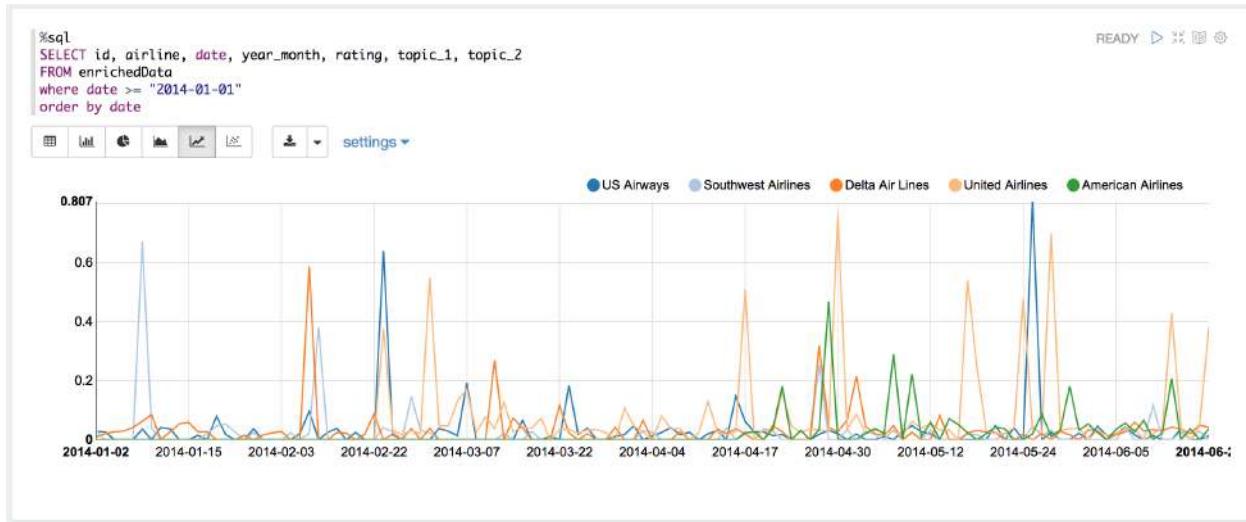
only showing top 20 rows

- 일별 항공사 평균 등급



- 월별 항공사 평균 등급





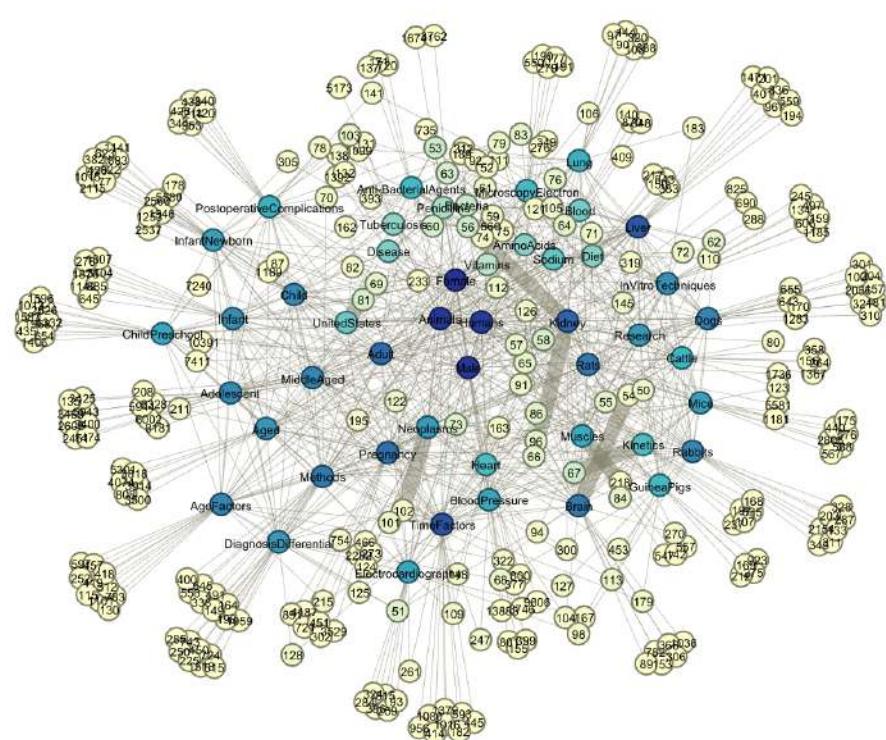
- 토픽 1과 관련된 리뷰(문서)

id	airline	date	review
10263	US Airways	2014-05-25	"Delays on all booked flights. Outward bound - Dublin to Philadelphia Philadelphia to Vegas. Vegas to LA. Baggage did not arrive some hours later. Cabin staff were unfriendly and quite rude. From Dublin We were sitting at the back of the plane and we were told ""thats what you get when you travel at the back"" . I opened my little tub of butter which had been heated in hot liquid and went all over my hand. I said to the stewardess who was passing by who could have brought me a napkin to sped by saying ""oh I know that happens"" . Only that I had a tray of food on my lap she would have had more to deal with! W baggage was to be stored in the overhead lockers or underneath the seat in front. This obviously does not apply to cabin staff back centre aisle seats of the plane it was not secured in any way. Their baggage was a danger to all the passengers in that what they preach and put staff baggage in the hold. Cabin crew were ungroomed in appearance. Homeward bound to Dublin delayed resulting in us overnighting in Orlando very grateful for the overnight accommodation provided at the Hyatt. Flew to Boarding for Dublin at Charlotte was very confused and inefficiently exercised by Gate staff - resulting in delay in take-off. Enjoy quality on those flights that had the facility. General impression overall. Very Disappointing."

소셜 네트워크 분석

중국 속담

무수한 방식으로 연결된 천의 감촉 - 중국의 옛 속담



15.1 도입

15.2 동시 출현 네트워크

동시 출현 네트워크는 일반적으로 서면 자료 내에 대표되는 사람, 조직, 개념 또는 다른 개체 간의 잠재적 관계의 그래픽 시각화를 제공하는 데 사용됩니다. 동시 출현 네트워크의 생성 및 시각화는 텍스트 마이닝에 적합한 전자 저장 텍스트의 출현으로 실용화되었습니다.

15.2.1 방법론

- 말뭉치 C 구축
- 말뭉치 C를 기반으로 문서-용어 행렬 D 구축
- 용어-문서 행렬 D^T 계산
- 인접 행렬 $A = D^T \cdot D$

알고리즘에서 네 가지 주요 구성 요소가 있습니다: 말뭉치 C, 문서-용어 행렬 D, 용어-문서 행렬 D^T 및 인접 행렬 A. 이 테모 파트에서 네 가지 주요 구성 요소를 구축하는 방법을 보여드리겠습니다.

만약 친구 세 그룹이 있다면, 다음과 같습니다.

+-----+		+-----+
words		
+-----+		+-----+
[[george] [jimmy] [john] [peter]]		
[[vincent] [george] [stefan] [james]]		
[[emma] [james] [olivia] [george]]		
+-----+		+-----+

1. 말뭉치 C

그 다음 주어진 그룹 데이터의 고유 요소를 기반으로 다음과 같은 말뭉치를 구축할 수 있습니다:

```
[u'george', u'james', u'jimmy', u'peter', u'stefan', u'veincent', u'  
→'olivia', u'john', u'emma']
```

해당 요소 빈도:



2. 말뭉치 C에 기반한 문서-용어 행렬 D(CountVectorizer)

```
from pyspark.ml.feature import CountVectorizer
count_vectorizer_wo = CountVectorizer(inputCol='term', outputCol='features')
# 총 고유 어휘
countVectorizer_mod_wo = count_vectorizer_wo.fit(df)
countVectorizer_twitter_wo = countVectorizer_mod_wo.transform(df)
# 생략된 고유 어휘 (99%)
count_vectorizer = CountVectorizer(vocabSize=48, inputCol='term',
                                   outputCol='features')
countVectorizer_mod = count_vectorizer.fit(df)
countVectorizer_twitter = countVectorizer_mod.transform(df)
```

```
+-----+
| features |
+-----+
|(9, [0,2,3,7], [1.0,1.0,1.0,1.0])|
|(9, [0,1,4,5], [1.0,1.0,1.0,1.0])|
|(9, [0,1,6,8], [1.0,1.0,1.0,1.0])|
+-----+
```

- 용어-문서 행렬 D^T

RDD:

```
[array([ 1.,  1.,  1.]), array([ 0.,  1.,  1.]), array([ 1.,  0.,  0.
˓→]),
 array([ 1.,  0.,  0.]), array([ 0.,  1.,  0.]), array([ 0.,  1.,  0.
˓→]),
 array([ 0.,  0.,  1.]), array([ 1.,  0.,  0.]), array([ 0.,  0.,  1.
˓→])]
```

행렬:

```
array([[ 1.,  1.,  1.],
       [ 0.,  1.,  1.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.],
       [ 1.,  0.,  0.],
       [ 0.,  0.,  1.]])
```

3. 인접 행렬 $A = D^T \cdot D$

RDD:

```
[array([ 1.,  1.,  1.]), array([ 0.,  1.,  1.]), array([ 1.,  0.,  0.
˓→]),
 array([ 1.,  0.,  0.]), array([ 0.,  1.,  0.]), array([ 0.,  1.,  0.
˓→]),
 array([ 0.,  0.,  1.]), array([ 1.,  0.,  0.]), array([ 0.,  0.,  1.
˓→])]
```

행렬:

```
array([[ 3.,  2.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
       [ 2.,  2.,  0.,  0.,  1.,  1.,  1.,  0.,  1.],
       [ 1.,  0.,  1.,  1.,  0.,  0.,  0.,  1.,  0.],
       [ 1.,  0.,  1.,  1.,  0.,  0.,  0.,  1.,  0.],
       [ 1.,  1.,  0.,  0.,  1.,  1.,  0.,  0.,  0.],
       [ 1.,  1.,  0.,  0.,  1.,  1.,  0.,  0.,  0.],
       [ 1.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  1.],
       [ 1.,  0.,  1.,  1.,  0.,  0.,  0.,  1.,  0.],
       [ 1.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  1.]])
```

15.2.2 인터뷰에 나온 코딩 퍼즐

- 문제

첨부된 인코딩된 utf-8 텍스트 파일에는 다음과 같이 형식이 지정된 온라인 생체 의학 과학 기사와 관련된 태그가 포함되어 있습니다(크기: 100000). 각 과학 기사는 캐리지 리턴으로(carriage return) 구분된 파일에서 한 줄에 해당합니다.

```
+-----+
|          words |
+-----+
| [ACTH Syndrome, E...|
| [Antibody Formati...|
| [Adaptation, Phys...|
| [Aerosol Propella...|
+-----+
only showing top 4 rows
```

이 파일을 입력으로 사용하여 최소 50개의 다른 과학 기사에서 순서와 위치에 따라 함께 나타나는 태그 쌍의 목록을 생성하는 프로그램을 작성합니다. 예를 들어 위 샘플에서 [여성]과 [인간]은 두 번 함께 나타나지만 모든 다른 쌍은 한 번만 나타납니다. 프로그램은 입력(예: 태그 1, 태그 2n)과 동일한 형식으로 stdout에 쌍 목록을 출력해야 합니다.

- 나의 답안

해당 단어 빈도:

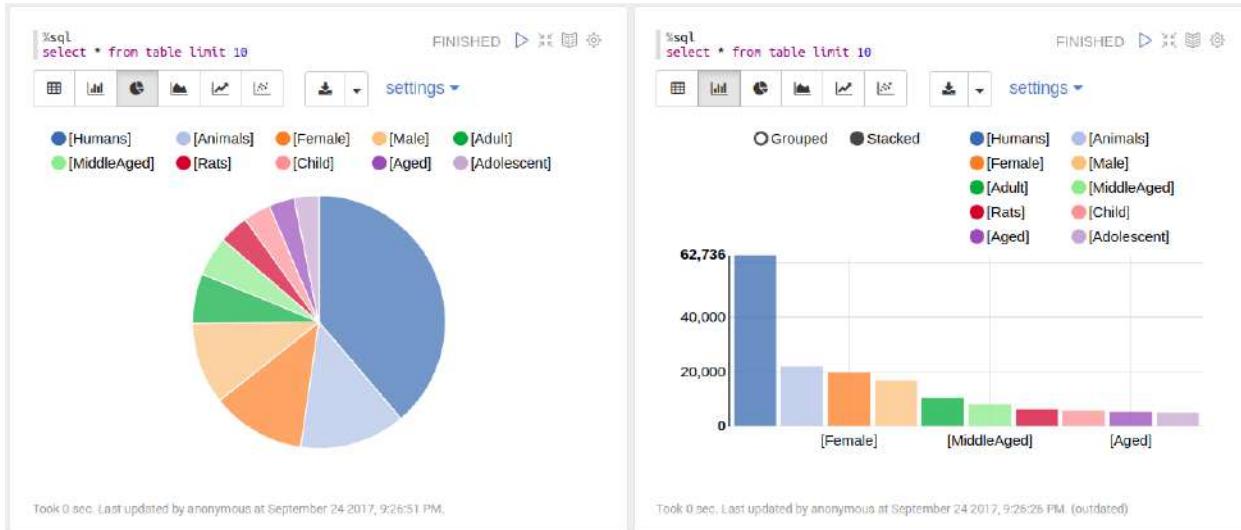


그림. 1: 단어 빈도

출력:

```
+-----+-----+-----+
| term.x|term.y| freq |
+-----+-----+-----+
| Female|Humans|16741.0|
| Male|Humans|13883.0|
| Adult|Humans|10391.0|
| Male|Female| 9806.0|
| MiddleAged|Humans| 8181.0|
| Adult|Female| 7411.0|
| Adult| Male| 7240.0|
| MiddleAged| Male| 6328.0|
| MiddleAged|Female| 6002.0|
| MiddleAged| Adult| 5944.0|
+-----+-----+-----+
only showing top 10 rows
```

해당 동시 출현 네트워크:

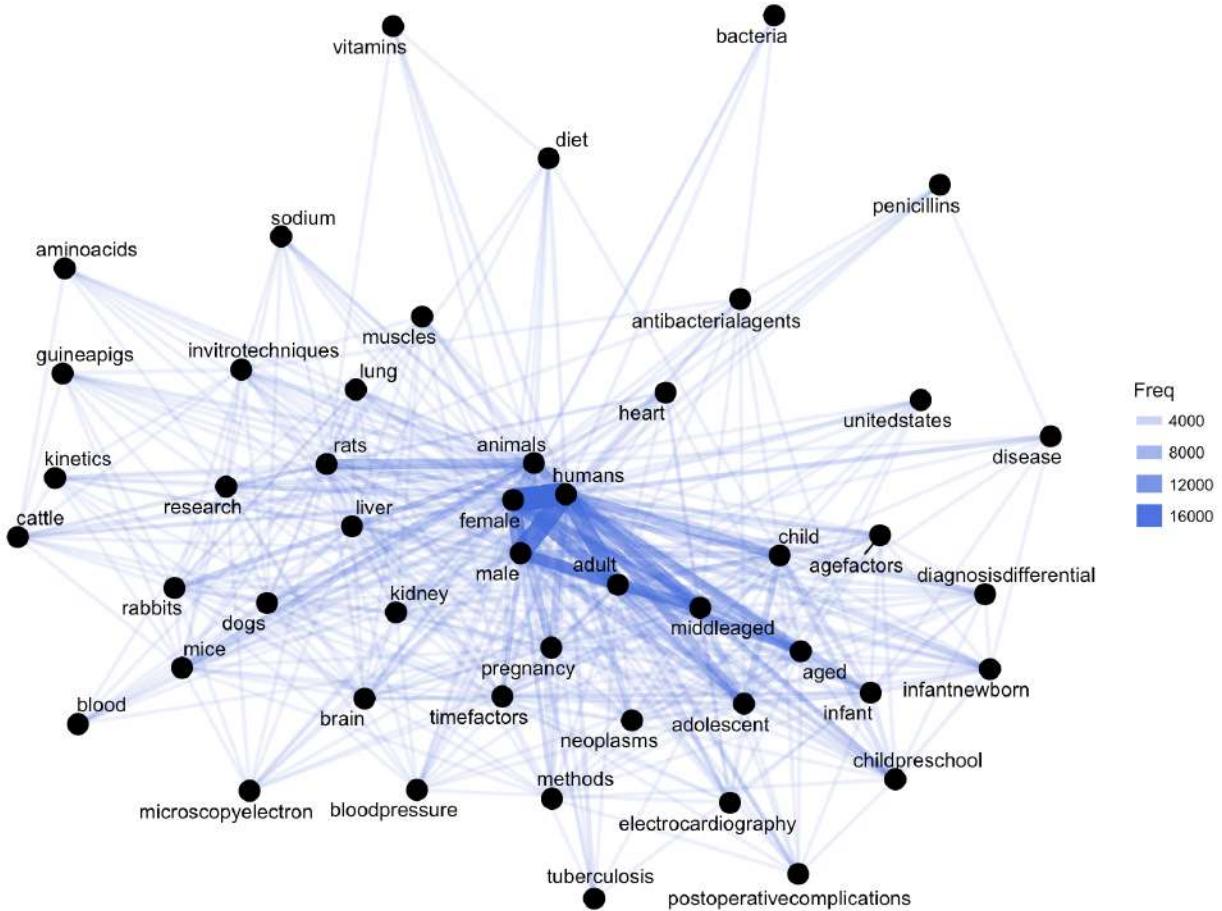


그림. 2: 동시 출현 네트워크

여러분은 그림 동시 출현 네트워크를 알게 될 것입니다.

15.3 부록: PySpark에서 행렬 곱셈

1. 테스트 행렬 로드하기

```
df = spark.read.csv("matrix1.txt", sep=",", inferSchema=True)
df.show()
```

```

+---+---+---+---+
|_c0|_c1|_c2|_c3|
+---+---+---+---+
|1.2|3.4|2.3|1.1|
|2.3|1.1|1.5|2.2|
|3.3|1.8|4.5|3.3|

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

5.3 2.2 4.5 4.4
9.3 8.1 0.3 5.5
4.5 4.3 2.1 6.6
+---+---+---+---+

2. PySpark에서 행렬 곱셈을 위한 주요 함수

```
from pyspark.sql import functions as F
from functools import reduce
# 참조: https://stackoverflow.com/questions/44348527/matrix-
→multiplication-at-a-in-pyspark
# 원하는 곱셈의 합을 수행하고 각 열에 대해 하나의 데이터 프레임을 얻습니다

colDFs = []
for c2 in df.columns:
    colDFs.append( df.select( [ F.sum(df[c1]*df[c2]).alias("op_{0}" .
→format(i)) for i,c1 in enumerate(df.columns) ] ) )
# 이제 "행렬"을 생성하기 위해 분리된 데이터 프레임을 통합합니다
mtxDF = reduce(lambda a,b: a.select(a.columns).union(b.select(a.columns)), ↵
→colDFs )
mtxDF.show()
```

op_0	op_1	op_2	op_3
152.45 118.88999999999999	57.15 121.44000000000001		
118.88999999999999 104.94999999999999	38.93	94.71	
57.15	38.93 52.54000000000006	55.99	
121.44000000000001	94.71	55.99 110.10999999999999	

3. 파이썬 버전과 결과가 동일한지 확인하기

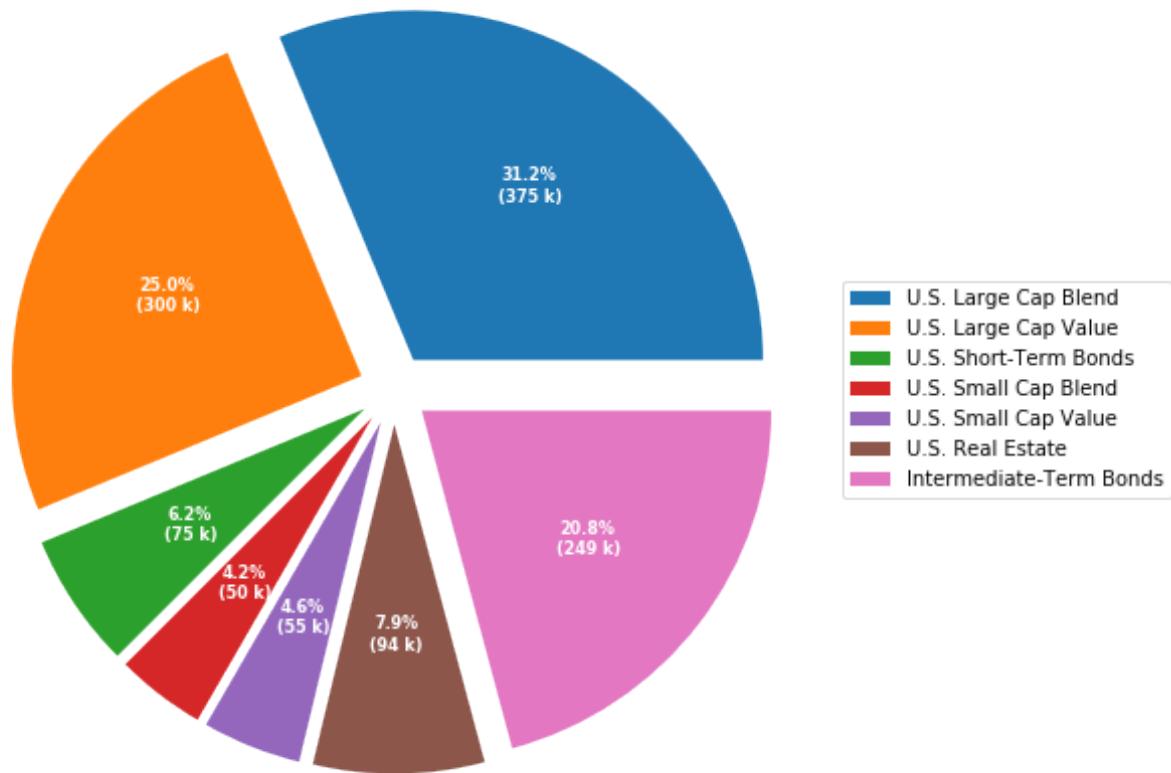
```
import numpy as np
a = np.genfromtxt("matrix1.txt", delimiter=",")
np.dot(a.T, a)
```

```
array([[152.45, 118.89,  57.15, 121.44],
       [118.89, 104.95,  38.93,  94.71],
       [ 57.15,  38.93,  52.54,  55.99],
       [121.44,  94.71,  55.99, 110.11]])
```


ALS: 주식 포트폴리오 추천

중국 속담

모든 것을 한 바구니에 넣지 말라.



위의 그림에 대한 코드:

```
import numpy as np
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 8), subplot_kw=dict(aspect="equal"))

recipe = ["375 k U.S. Large Cap Blend",
          "300 k U.S. Large Cap Value",
          "75 k U.S. Short-Term Bonds",
          "50 k U.S. Small Cap Blend",
          "55 k U.S. Small Cap Value",
          "95 k U.S. Real Estate",
          "250 k Intermediate-Term Bonds"]

data = [float(x.split()[0]) for x in recipe]
ingredients = [' '.join(x.split()[2:]) for x in recipe]

print(data)
print(ingredients)
def func(pct, allvals):
    absolute = int(pct/100.*np.sum(allvals))
    return "{:.1f}%\n({:d} k)".format(pct, absolute)

explode = np.empty(len(data)) # 0.1, 0.1, 0.1, 0.1, 0.1, 0.1) # explode 1st slice
explode.fill(0.1)

wedges, texts, autotexts = ax.pie(data, explode=explode, autopct=lambda pct: func(pct, data),
                                    textprops=dict(color="w"))
ax.legend(wedges, ingredients,
          title="Stock portfolio",
          loc="center left",
          bbox_to_anchor=(1, 0, 0.5, 1))

plt.setp(autotexts, size=8, weight="bold")

#ax.set_title("Stock portfolio")

plt.show()
```

16.1 추천 시스템

추천 시스템은(Recommender systems 혹은 recommendation systems)(때로는 "시스템"을 플랫폼 또는 엔진과 같은 동의어로 대체함) 사용자가 아이템에 부여할 "등급" 또는 "선호도"를 예측하려 하는 정보 필터링 시스템의 하위 클래스입니다.

주요 아이디어는 행렬, 사용자 R 아이템 등급 값을 만들고 이를 인수분해하여 다른 사용자가 등급을 매긴 주요 제품을 추천하는 것입니다. 이에 대한 일반적인 접근법은 교대 최소 제곱(ALS)입니다.

16.2 교대 최소 제곱(Alternating Least Squares)

Apache Spark ML은 매우 일반적인 권장 알고리즘인 협업 필터링을 위해 ALS를 구현합니다.

ALS 추천자는 교대 제곱에 가중치 람다-규제화(Alternating Lamda-Regularization)를 사용하는 행렬 인수분해 알고리즘입니다. 사용자 대 아이템 행렬 A를 사용자 대 특징 행렬 U와 아이템 대 특징 행렬 M으로 요인화합니다: ALS 알고리즘을 병렬 방식으로 실행합니다. ALS 알고리즘은 관측된 사용자를 아이템 등급으로 설명하는 잠재 요인을 밝혀내고 예측 등급과 실제 등급 사이의 최소 제곱을 최소화하기 위해 최적의 요인 가중치를 찾으려 합니다.

<https://www.elenacuoco.com/2016/12/22/alternating-least-squares-als-spark-ml/>

16.3 예모

- 주피터 노트북은 추천 시스템에서 다운로드할 수 있습니다.
- 데이터는 German Credit에서 다운로드할 수 있습니다.

16.3.1 데이터 읽기 및 정리

1. 스파크 컨텍스트 및 스파크세션 설정

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark RFM example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. 데이터 로드하기

```
df_raw = spark.read.format('com.databricks.spark.csv') \
    .options(header='true', \
    inferschema='true') \
    .load("Online Retail.csv", header=True);
```

데이터셋 확인하기

```
df_raw.show(5)
df_raw.printSchema()
```

여러분은 다음과 같은 결과를 얻을 수 있습니다.

InvoiceNo	StockCode	Description	Quantity	Country
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

(다음 페이지에서 계속)

(이전 페이지에서 계속)

<hr/>						
↳	536365	85123A	WHITE HANGING HEA...		6 12/1/10 8:26	2.55 ↳
↳	17850	United Kingdom				
↳	536365	71053	WHITE METAL LANTERN		6 12/1/10 8:26	3.39 ↳
↳	17850	United Kingdom				
↳	536365	84406B	CREAM CUPID HEART...		8 12/1/10 8:26	2.75 ↳
↳	17850	United Kingdom				
↳	536365	84029G	KNITTED UNION FLA...		6 12/1/10 8:26	3.39 ↳
↳	17850	United Kingdom				
↳	536365	84029E	RED WOOLLY HOTTIE...		6 12/1/10 8:26	3.39 ↳
↳	17850	United Kingdom				
<hr/>						
only showing top 5 rows						
<hr/>						
root						
-- InvoiceNo:	string	(nullable = true)				
-- StockCode:	string	(nullable = true)				
-- Description:	string	(nullable = true)				
-- Quantity:	integer	(nullable = true)				
-- InvoiceDate:	string	(nullable = true)				
-- UnitPrice:	double	(nullable = true)				
-- CustomerID:	integer	(nullable = true)				
-- Country:	string	(nullable = true)				

3. 데이터 정리 및 처리

- null값들을 확인하고 제거하기

```
from pyspark.sql.functions import count

def my_count(df_in):
    df_in.agg( *[ count(c).alias(c) for c in df_in.columns ] ).show()
```

```
import pyspark.sql.functions as F
from pyspark.sql.functions import round
df_raw = df_raw.withColumn('Asset', round(F.col('Quantity') * F.col('UnitPrice')
    ↵'), 2))
df = df_raw.withColumnRenamed('StockCode', 'Cusip') \
    .select('CustomerID', 'Cusip', 'Quantity', 'UnitPrice', 'Asset')
```

```
my_count(df)
```

<hr/>				
CustomerID	Cusip	Quantity	UnitPrice	Asset
<hr/>				
406829	541909 541909 541909 541909			
<hr/>				

카운트(count) 결과가 동일하지 않으므로 CustomerID 열에 null 값이 있습니다.

데이터 세트에서 해당 레코드를 삭제할 수 있습니다

```
df = df.filter(F.col('Asset')>=0)
df = df.dropna(how='any')
my_count(df)
```

CustomerID	Cusip	Quantity	UnitPrice	Asset
397924	397924	397924	397924	397924

```
df.show(3)
```

CustomerID	Cusip	Quantity	UnitPrice	Asset
17850	85123A	6	2.55	15.3
17850	71053	6	3.39	20.34
17850	84406B	8	2.75	22.0

only showing top 3 rows

- Cusip 열의 속성값을 일관된 형식으로 변환하기

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType

def toUpper(s):
    return s.upper()

upper_udf = udf(lambda x: toUpper(x), StringType())
```

- 상위 n개 주식 찾기

```
pop = df.groupBy('Cusip') \
    .agg(F.count('CustomerID').alias('Customers'), F.round(F.sum('Asset'), 2) \
    .alias('TotalAsset')) \
    .sort([F.col('Customers'), F.col('TotalAsset')], ascending=[0, 0])

pop.show(5)
```

Cusip	Customers	TotalAsset
85123A	2035	100603.5
22423	1724	142592.95
85099B	1618	85220.78
84879	1408	56580.34
47566	1397	68844.33

only showing top 5 rows

16.3.2 특징 행렬 생성하기

- 상위 n cusip 목록 가져오기

```
top = 10
cusip_lst = pd.DataFrame(pop.select('Cusip').head(top)).astype('str').iloc[:, -1]
cusip_lst.insert(0, 'CustomerID')
```

- 각 고객에 대한 포트폴리오 테이블 생성하기

```
pivot_tab = df.groupby('CustomerID').pivot('Cusip').sum('Asset')
pivot_tab = pivot_tab.fillna(0)
```

- 각 고객에 대한 n 개의 주식 포트폴리오 테이블 가져오기

```
selected_tab = pivot_tab.select(cusip_lst)
selected_tab.show(4)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| CustomerID | 85123A | 22423 | 85099B | 84879 | 47566 | 20725 | 22720 | 20727 | POST | 23203 |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 16503 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 33.0 | 0.0 | 0.0 |
| 15727 | 123.9 | 25.5 | 0.0 | 0.0 | 0.0 | 33.0 | 99.0 | 0.0 | 0.0 | 0.0 |
| 14570 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 14450 | 0.0 | 0.0 | 8.32 | 0.0 | 0.0 | 0.0 | 49.5 | 0.0 | 0.0 | 0.0 |
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 4 rows
```

- 등급 행렬 생성하기

```
def elemwiseDiv(df_in):
    num = len(df_in.columns)
    temp = df_in.rdd.map(lambda x: list(flatten([x[0], [x[i]/float(sum(x[1:])) if sum(x[1:])>0 else 0 for i in range(1, num)]])))
    return spark.createDataFrame(temp, df_in.columns)

ratings = elemwiseDiv(selected_tab)
```

```
ratings.show(4)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| CustomerID | 85123A | 22423 | 85099B | 84879 | 47566 | 20725 | 22720 | 20727 | POST | 23203 |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 16503 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| 15727 | 0.44 | 0.09 | 0.0 | 0.0 | 0.0 | 0.12 | 0.35 | 0.0 | 0.0 | 0.0 |
```

(다음 페이지에 계속)

(○] 전 페이지에서 계속)

14570 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
14450 0.0 0.0 0.14 0.0 0.0 0.0 0.86 0.0 0.0 0.0 0.0 0.0

- 등급 행렬을 long 테이블로 변환하기

```
from pyspark.sql.functions import array, col, explode, struct, lit

def to_long(df, by):
    """
    참조: https://stackoverflow.com/questions/37864222transpose-column-to-
    row-with-spark
    """

    # dtypes을 필터링하고 열 이름과 유형 설명으로 구분합니다
    cols, dtypes = zip(*((c, t) for (c, t) in df.dtypes if c not in by))
    # Spark SQL은 동종 열만 지원합니다
    assert len(set(dtypes)) == 1, "모든 열은 동일한 타입이여야 합니다"

    # (column_name, column_value) 구조 배열을 생성하고 여러 행으로 전개합니다
    kvs = explode(array([
        struct(lit(c).alias("Cusip"), col(c).alias("rating")) for c in
        cols])).alias("kvs")
```

```
df_all = to_long(ratings, ['CustomerID'])
df_all.show(5)
```

CustomerID	Cusip	rating
16503	85123A	0.0
16503	22423	0.0
16503	85099B	0.0
16503	84879	0.0
16503	47566	0.0

only showing top 5 rows

- 문자열 Cusip 을 숫자 인덱스로 변환합니다

```
from pyspark.ml.feature import StringIndexer
# 라벨들에 색인을 달고, 메타데이터를 라벨 칼럼에 추가합니다
labelIndexer = StringIndexer(inputCol='Cusip',
                             outputCol='indexedCusip').fit(df_all)
df_all = labelIndexer.transform(df_all)

df_all.show(5, True)
df_all.printSchema()
```

CustomerID	indexedCusip
16503	0.0
16503	0.0
16503	0.0
16503	0.0
16503	0.0

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
|CustomerID| Cusip|rating|indexedCusip|
+-----+-----+-----+
| 16503|85123A| 0.0| 6.0|
| 16503| 22423| 0.0| 9.0|
| 16503|85099B| 0.0| 5.0|
| 16503| 84879| 0.0| 1.0|
| 16503| 47566| 0.0| 0.0|
+-----+-----+-----+
only showing top 5 rows

root
|-- CustomerID: long (nullable = true)
|-- Cusip: string (nullable = false)
|-- rating: double (nullable = true)
|-- indexedCusip: double (nullable = true)
```

16.3.3 모델 훈련

- 훈련과 검증 셋 생성하기

```
train, test = df_all.randomSplit([0.8, 0.2])

train.show(5)
test.show(5)
```

```
+-----+-----+-----+-----+
|CustomerID|Cusip|indexedCusip| rating|
+-----+-----+-----+-----+
| 12940|20725| 2.0| 0.0|
| 12940|20727| 4.0| 0.0|
| 12940|22423| 9.0| 0.49990198000392083|
| 12940|22720| 3.0| 0.0|
| 12940|23203| 7.0| 0.0|
+-----+-----+-----+-----+
only showing top 5 rows

+-----+-----+-----+-----+
|CustomerID|Cusip|indexedCusip| rating|
+-----+-----+-----+-----+
| 12940|84879| 1.0| 0.1325230346990786|
| 13285|20725| 2.0| 0.2054154995331466|
| 13285|20727| 4.0| 0.2054154995331466|
| 13285|47566| 0.0| 0.0|
| 13623|23203| 7.0| 0.0|
+-----+-----+-----+-----+
only showing top 5 rows
```

- 모델 훈련하기

```

import itertools
from math import sqrt
from operator import add
import sys
from pyspark.ml.recommendation import ALS

from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                 predictionCol="prediction")
def computeRmse(model, data):
    """
    RMSE (Root mean Squared Error) 계산하기
    """
    predictions = model.transform(data)
    rmse = evaluator.evaluate(predictions)
    print("Root-mean-square error = " + str(rmse))
    return rmse

# 모델을 훈련하고 검증 셋으로 모델을 평가합니다

ranks = [4, 5]
lambdas = [0.05]
numIters = [30]
bestModel = None
bestValidationRmse = float("inf")
bestRank = 0
bestLambda = -1.0
bestNumIter = -1

val = test.na.drop()
for rank, lmbda, numIter in itertools.product(ranks, lambdas, numIters):
    als = ALS(rank=rank, maxIter=numIter, regParam=lmbda, numUserBlocks=10,
              numItemBlocks=10, implicitPrefs=False,
              alpha=1.0,
              userCol="CustomerID", itemCol="indexedCusip", seed=1, ratingCol=
              "rating", nonnegative=True)
    model=als.fit(train)

    validationRmse = computeRmse(model, val)
    print("RMSE (validation) = %f for the model trained with " %
          validationRmse +
          "rank = %d, lambda = %.1f, and numIter = %d." % (rank, lmbda,
          numIter))
    if (validationRmse, bestValidationRmse):
        bestModel = model
        bestValidationRmse = validationRmse
        bestRank = rank
        bestLambda = lmbda
        bestNumIter = numIter

model = bestModel

```

16.3.4 예측값 만들기

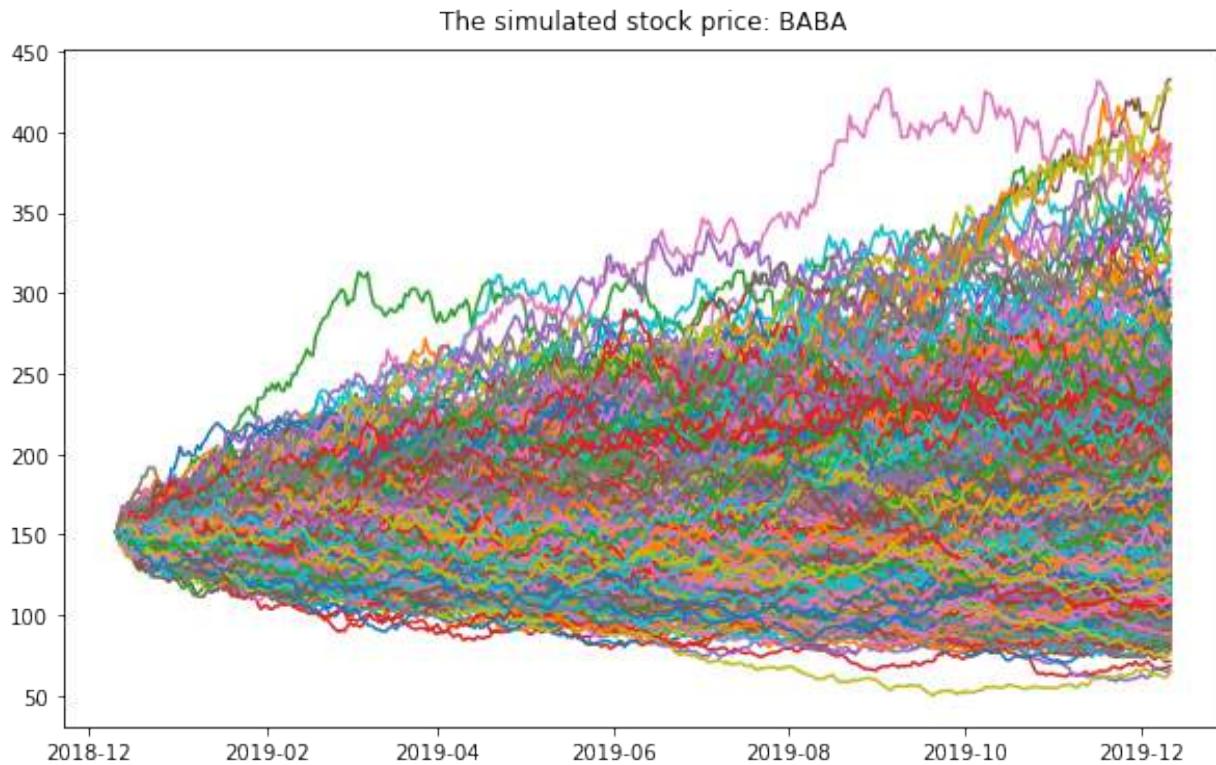
- 예측값 만들기

```
topredict=test[test['rating']==0]

predictions=model.transform(topredict)
predictions.filter(predictions.prediction>0) \
    .sort([F.col('CustomerID'),F.col('Cusip')],ascending=[0,0]).show(5)
```

```
+-----+-----+-----+-----+
|CustomerID| Cusip|indexedCusip|rating| prediction|
+-----+-----+-----+-----+
|     18282| 47566|          0.0|   0.0|  0.01625076|
|     18282| 85123A|          6.0|   0.0|  0.057172246|
|     18282| 84879|          1.0|   0.0|  0.059531752|
|     18282| 23203|          7.0|   0.0|  0.010502596|
|     18282| 22720|          3.0|   0.0|  0.053893942|
+-----+-----+-----+-----+
only showing top 5 rows
```

몬테 카를로 시뮬레이션



몬테카를로 시뮬레이션은 난수를 반복적으로 생성하여 고정된 매개 변수를 추정하는 방법입니다. 더 자세한 내용은 [마르코프 체인 몬테카를로 방법에 대한 Zero 수학 도입](#)에서 확인할 수 있습니다.

몬테카를로 시뮬레이션은 재정, 프로젝트 관리, 비용 및 기타 예측 모델에서 위험과 불확실성의 영향을 이해하는 데 사용되는 기술입니다. 몬테카를로 시뮬레이터는 결정의 위험에 대해 더 나은 생각을 가질 수 있도록 잠재적인 결과의 대부분 또는 모든 것을 시각화하는 데 도움이 됩니다. 더 자세한 내용은 [항상 이기는 도박](#)에서 확인할 수 있습니다.

17.1 카지노에서 승리 시뮬레이션

우리는 존 선수가 게임에서 이길 가능성이 49%이고 베팅은 게임당 5달러라고 가정합니다.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from functools import reduce

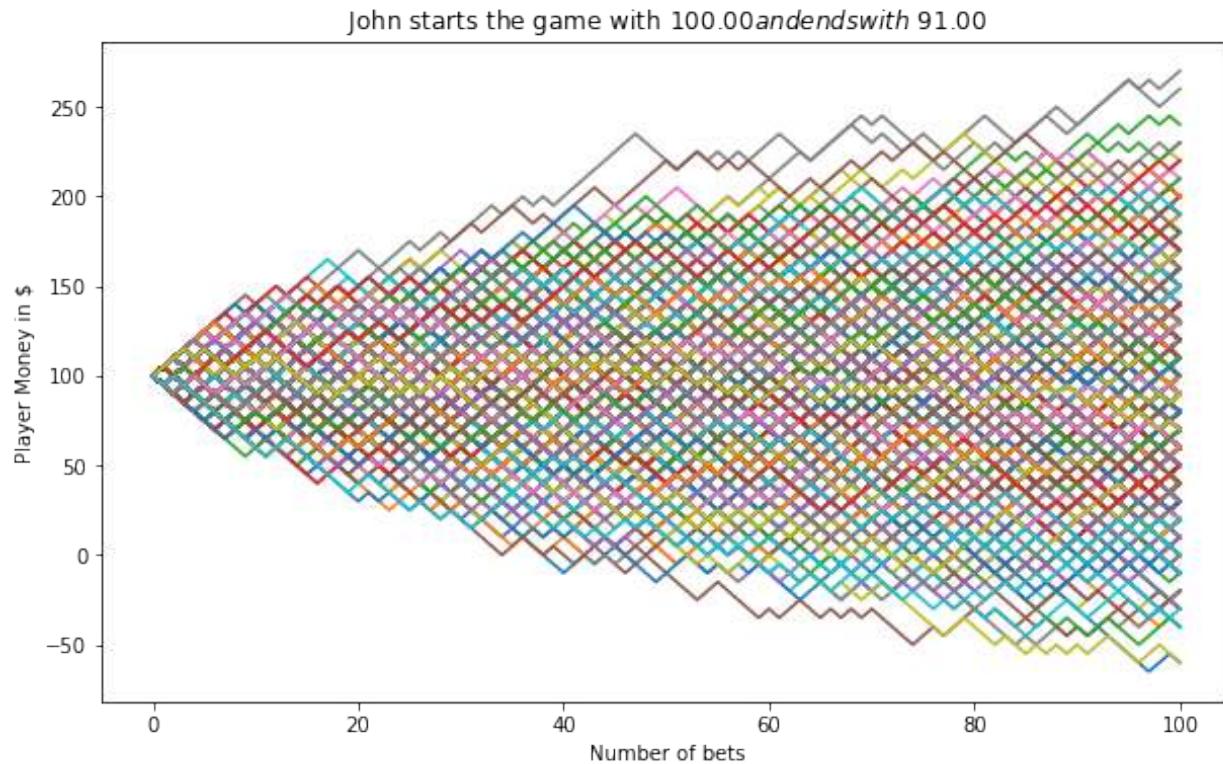
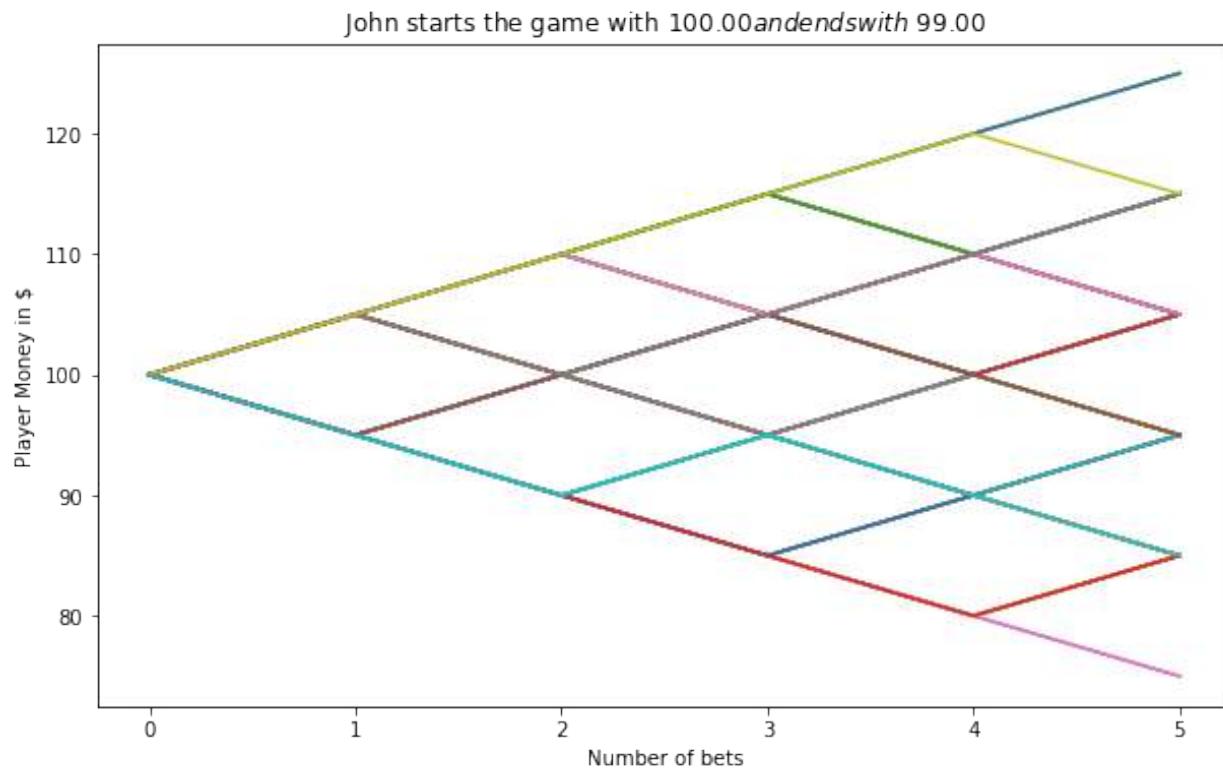
start_m = 100
wager = 5
bets = 100
trials = 1000

trans = np.vectorize(lambda t: -wager if t <=0.51 else wager)

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(1,1,1)

end_m = []
for i in range(trials):

    money = reduce(lambda c, x: c + [c[-1] + x], trans(np.random.
    ↪random(bets)), [start_m])
    end_m.append(money[-1])
    plt.plot(money)
plt.ylabel('Player Money in $')
plt.xlabel('Number of bets')
plt.title(("John starts the game with $ %.2f and ends with $ %.2f")%(start_m,
    ↪sum(end_m)/len(end_m)))
plt.show()
```



17.2 랜덤워크 시뮬레이션

17.2.1 과거 주가 불러오기

- finance.yahoo.com에서 회사명을 검색하고 historical data에서 time period를 지정해 해당 회사의 과거 주가 데이터를 다운로드할 수 있습니다. 상세 코드가 필요하시면 저에게 연락주시길 바랍니다.

```
stock.tail(4)
```

Date	Open	High	Low	Close	Adj Close	Volume
2018-12-07	155.399994	158.050003	151.729996	153.059998	153.059998	17447900
2018-12-10	150.389999	152.809998	147.479996	151.429993	151.429993	15525500
2018-12-11	155.259995	156.240005	150.899994	151.830002	151.830002	13651900
2018-12-12	155.240005	156.169998	151.429993	151.5	151.5	16597900

- str 타입 데이터를 date 타입으로 변환합니다

```
stock['Date'] = pd.to_datetime(stock['Date'])
```

- 데이터를 시각화합니다

```
# 매우 강력한 matplotlib 패키지를 활용하여 모든 것을 구성합니다
width = 10
height = 6
data = stock
fig = plt.figure(figsize=(width, height))
ax = fig.add_subplot(1,1,1)
ax.plot(data.Date, data.Close, label='Close')
ax.plot(data.Date, data.High, label='High')
# ax.plot(data.Date, data.Low, label='Low')
ax.set_xlabel('Date')
ax.set_ylabel('price ($)')
ax.legend()
ax.set_title('Stock price: ', y=1.01)
#plt.xticks(rotation=70)
plt.show()

# 매우 강력한 matplotlib 패키지를 활용하여 모든 것을 구성합니다
fig = plt.figure(figsize=(width, height))
ax = fig.add_subplot(1,1,1)
ax.plot(data.Date, data.Volume, label='Volume')
#ax.plot(data.Date, data.High, label='High')
# ax.plot(data.Date, data.Low, label='Low')
ax.set_xlabel('Date')
ax.set_ylabel('Volume')
ax.legend()
ax.set_title('Stock volume: ' + ticker, y=1.01)
#plt.xticks(rotation=70)
plt.show()
```

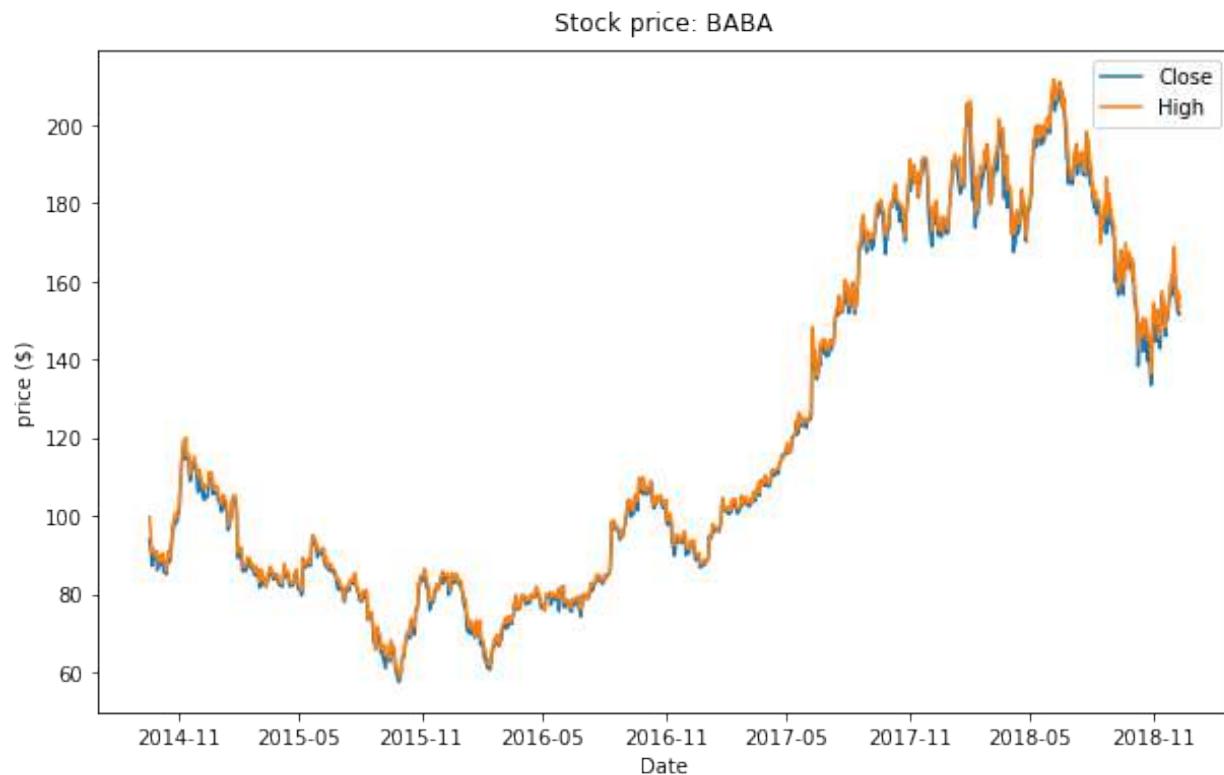


그림. 1: 과거 주가

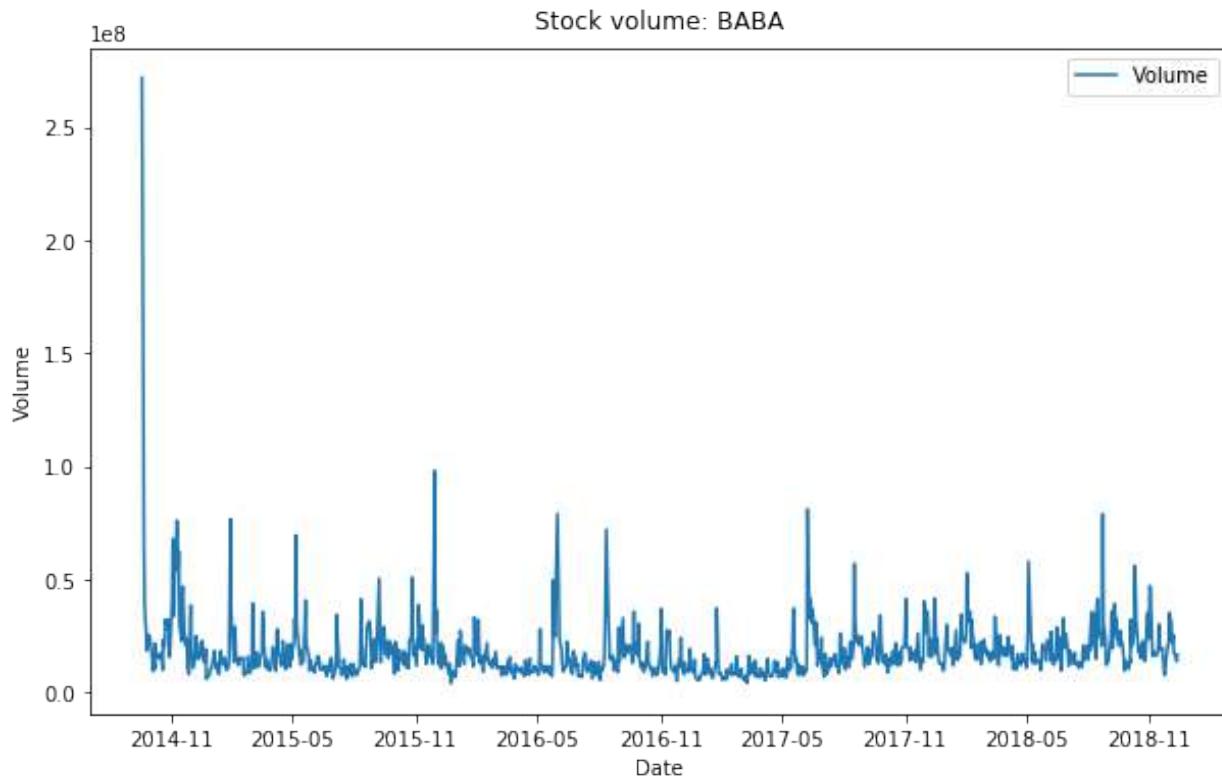


그림. 2: 과거 주식거래량

17.2.2 연평균 복합 성장률(Compound Annual Growth Rate) 계산

CAGR(Compound Annual Growth Rate) 공식은 투자 분석에 매우 유용합니다. 이 공식은 방정식의 대수적 형태에 따라 연간화된 수익률 또는 연간 백분율 수익률 또는 유효 연율로도 지칭될 수 있습니다. 주식과 같은 많은 투자는 급격하게 변동할 수 있는 수익률을 가지고 있습니다. CAGR 공식을 사용하면 다른 투자와 비교하는 데 사용할 수 있는 "평활한" 수익률을 계산할 수 있습니다. 이 공식은 다음과 같이 정의됩니다(자세한 내용은 CAGR 계산기 및 공식에서 확인할 수 있습니다).

$$CAGR = \left(\frac{\text{마지막 연도 값}}{\text{최초 연도 값}} \right)^{\frac{365}{\text{Days}}} - 1$$

```
days = (stock.Date.iloc[-1] - stock.Date.iloc[0]).days
cagr = (((stock['Adj Close'].iloc[-1]) / stock['Adj Close'].iloc[0])) ** (365.0/days) - 1
print ('CAGR =', '{0:.4f}'.format(cagr*100)+"%")
mu = cagr
```

17.2.3 연간 변동성 계산

주식의 변동성은 일정 기간 동안의 가격 변동입니다. 예를 들어, 한 주식은 훨씬 더 높고 낮게 변동하는 경향이 있는 반면, 다른 주식은 훨씬 더 안정적이고 덜 요동치는 방식으로 움직일 수 있습니다. 두 주식은 결국 같은 가격에 도달할 수 있지만, 그 시점까지의 경로는 매우 다를 수 있습니다. 먼저 일련의 백분율 수익률을 생성하고 연간 수익률 변동성을 계산합니다. 이 변동성을 연간 단위로 표시하려면 일별 표준 편차에 252 제곱근을 곱하면 됩니다. 이는 특정 연도에 252 거래일이 있다고 가정합니다. 더 자세한 내용은 연간 변동성 계산 방법을 참조하십시오.

```
stock['Returns'] = stock['Adj Close'].pct_change()
vol = stock['Returns'].std()*np.sqrt(252)
```

17.2.4 일별 수익 행렬 작성

- 랜덤 정규 분포를 사용하여 일일 수익 행렬 생성합니다. 균등 분포 $U(0.0, 1.0)$ 에서 i.i.d. 샘플로 구성된 RDD 행렬을 생성합니다.

```
import math
S = stock['Adj Close'].iloc[-1] #시작 주가(즉, 마지막 이용 가능한 실제 주가)
T = 5 #거래일수
mu = cagr #반환
vol = vol #변동성
trials = 10000
mat = RandomRDDs.normalVectorRDD(sc, trials, T, seed=1)
```

- 생성된 RDD의 분포를 $U(0.0, 1.0)$ 에서 $U(a, b)$ 로 변환하고, RandomRDDs.uniformRDD(sc, n, p, seed) .map(lambda v: a + (b - a) * v)를 사용합니다.

```
a = mu/T
b = vol/math.sqrt(T)
v = mat.map(lambda x: a + (b - a) * x)
```

- RDD행렬을 데이터 프레임으로 변환합니다.

```
df = v.map(lambda x: [round(i, 6)+1 for i in x]).toDF()
df.show(5)
```

_1	_2	_3	_4	_5
0.935234	1.162894	1.07972	1.238257	1.066136
0.878456	1.045922	0.990071	1.045552	0.854516
1.186472	0.944777	0.742247	0.940023	1.220934
0.872928	1.030882	1.248644	1.114262	1.063762
1.09742	1.188537	1.137283	1.162548	1.024612

only showing top 5 rows

```
from pyspark.sql.functions import lit
S = stock['Adj Close'].iloc[-1]
price = df.withColumn('init_price', lit(S))
```

```
price.show(5)

+-----+-----+-----+-----+-----+
|      _1|      _2|      _3|      _4|      _5|init_price|
+-----+-----+-----+-----+-----+
| 0.935234| 1.162894| 1.07972| 1.238257| 1.066136|      151.5|
| 0.878456| 1.045922| 0.990071| 1.045552| 0.854516|      151.5|
| 1.186472| 0.944777| 0.742247| 0.940023| 1.220934|      151.5|
| 0.872928| 1.030882| 1.248644| 1.114262| 1.063762|      151.5|
| 1.09742| 1.188537| 1.137283| 1.162548| 1.024612|      151.5|
+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
price = price.withColumn('day_0', col('init_price'))
price.show(5)
```

```
+-----+-----+-----+-----+-----+-----+
|      _1|      _2|      _3|      _4|      _5|init_price|day_0|
+-----+-----+-----+-----+-----+-----+
| 0.935234| 1.162894| 1.07972| 1.238257| 1.066136|      151.5| 151.5|
| 0.878456| 1.045922| 0.990071| 1.045552| 0.854516|      151.5| 151.5|
| 1.186472| 0.944777| 0.742247| 0.940023| 1.220934|      151.5| 151.5|
| 0.872928| 1.030882| 1.248644| 1.114262| 1.063762|      151.5| 151.5|
| 1.09742| 1.188537| 1.137283| 1.162548| 1.024612|      151.5| 151.5|
+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

17.2.5 몬테 카를로 시뮬레이션

```
from pyspark.sql.functions import round
for name in price.columns[:-2]:
    price = price.withColumn('day'+name, round(col(name)*col('init_price'), 2))
    price = price.withColumn('init_price', col('day'+name))
```

```
price.show(5)

+-----+-----+-----+-----+-----+-----+-----+
|      _1|      _2|      _3|      _4|      _5|init_price|day_0| day_1| day_2| day_3| day_4| day_5|
+-----+-----+-----+-----+-----+-----+-----+
| 0.935234| 1.162894| 1.07972| 1.238257| 1.066136|      234.87| 151.5| 141.69| 164.77| 177.91| 220.3| 234.87|
| 0.878456| 1.045922| 0.990071| 1.045552| 0.854516|      123.14| 151.5| 133.09| 139.2| 137.82| 144.1| 123.14|
+-----+-----+-----+-----+-----+-----+-----+
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
| 1.186472|0.944777|0.742247|0.940023|1.220934 | 144.67|151.5|179.75|169.
| 82|126.05|118.49|144.67|
| 0.872928|1.030882|1.248644|1.114262|1.063762 | 201.77|151.5|132.25|136.
| 33|170.23|189.68|201.77|
| 1.09742|1.188537|1.137283|1.162548|1.024612 | 267.7|151.5|166.26|197.
| 61|224.74|261.27| 267.7|
+-----+-----+-----+-----+-----+-----+-----+-----+
|-----+-----+-----+
only showing top 5 rows
```

17.2.6 요약

```
selected_col = [name for name in price.columns if 'day_' in name]

simulated = price.select(selected_col)
simulated.describe().show()
```

```
+-----+-----+-----+-----+-----+
|-----+-----+-----+-----+
| summary|2018-12-12| 2018-12-13| 2018-12-14| 2018-12-17|_
| 2018-12-18| 2018-12-19|_
+-----+-----+-----+-----+-----+
|-----+-----+
| count| 10000.0| 10000.0| 10000.0| 10000.0|_
| 10000.0| 10000.0| 10000.0| 10000.0| 10000.0|_
| mean| 151.5| 155.11643700000002| 158.489058| 162.23713200000003|_
| 166.049375| 170.006525| 18.313783237787845| 26.460919262517276| 33.
| std| 0.0| 18.313783237787845| 26.460919262517276| 33.
| 37780495150803| 39.369101074463416| 45.148120695490846| 65.87|_
| min| 151.5| 88.2| 74.54| 65.87|_
| 68.21| 58.25| 58.25| 65.87|_
| 25%| 151.5| 142.485| 140.15| 138.72|_
| 138.365| 137.33| 137.33| 138.72|_
| 50%| 151.5| 154.97| 157.175| 159.82|_
| 162.59| 165.04500000000002| 157.175| 159.82|_
| 75%| 151.5| 167.445| 175.48499999999999| 182.8625|_
| 189.725| 196.975| 196.975| 182.8625|_
| max| 151.5| 227.48| 275.94| 319.17|_
| 353.59| 403.68| 403.68| 319.17|_
+-----+-----+-----+-----+-----+
|-----+-----+-----+
```

```
data_plt = simulated.toPandas()
days = pd.date_range(stock['Date'].iloc[-1], periods= T+1, freq='B').date

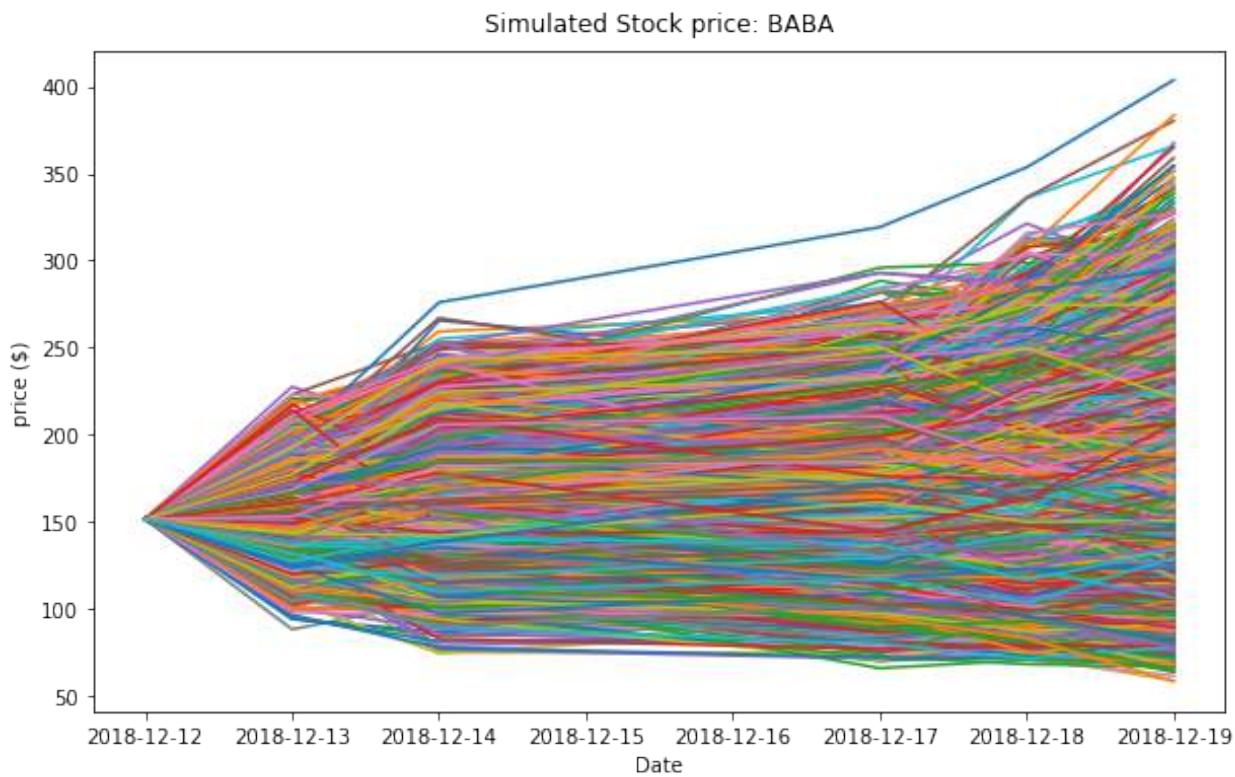
width = 10
height = 6
fig = plt.figure(figsize=(width, height))
ax = fig.add_subplot(1,1,1)
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
days = pd.date_range(stock['Date'].iloc[-1], periods= T+1, freq='B').date

for i in range(trials):
    plt.plot(days, data_plt.iloc[i])
ax.set_xlabel('Date')
ax.set_ylabel('price ($)')
ax.set_title('Simulated Stock price: ' + ticker, y=1.01)
plt.show()
```



17.2.7 1년 주식 가격 시뮬레이션

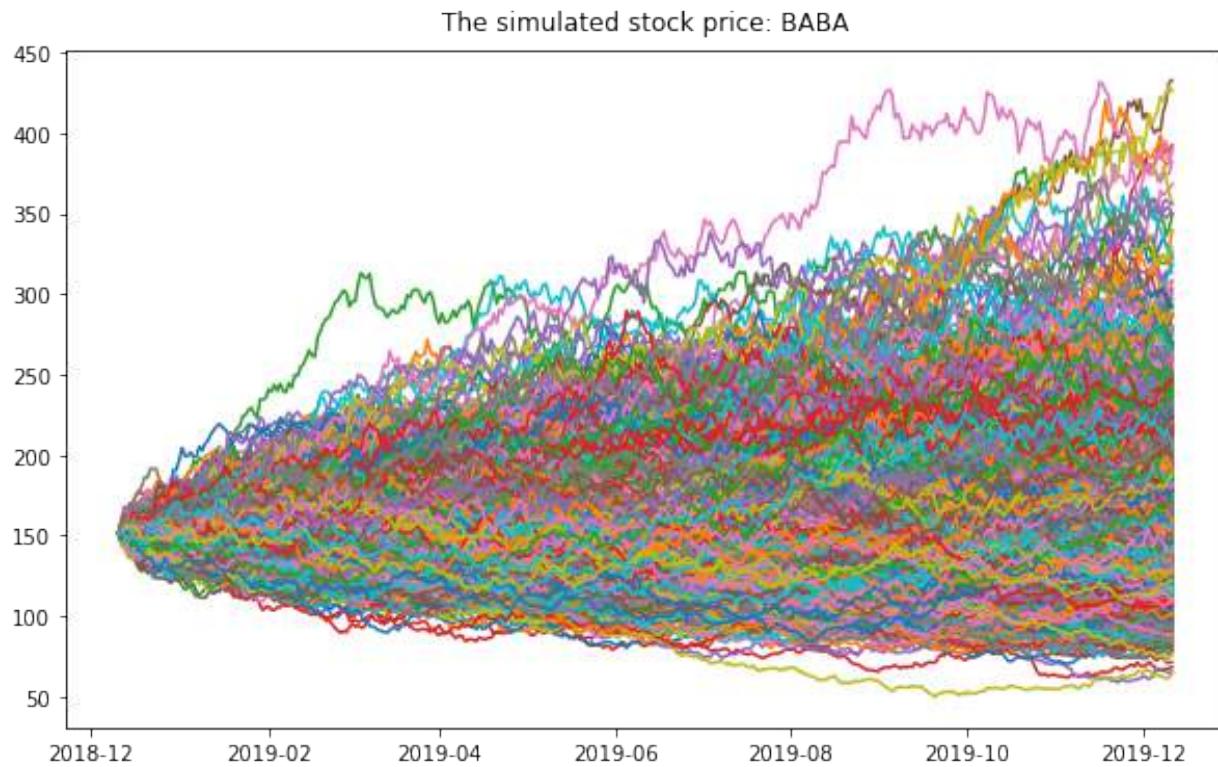


그림. 3: 모의 주가

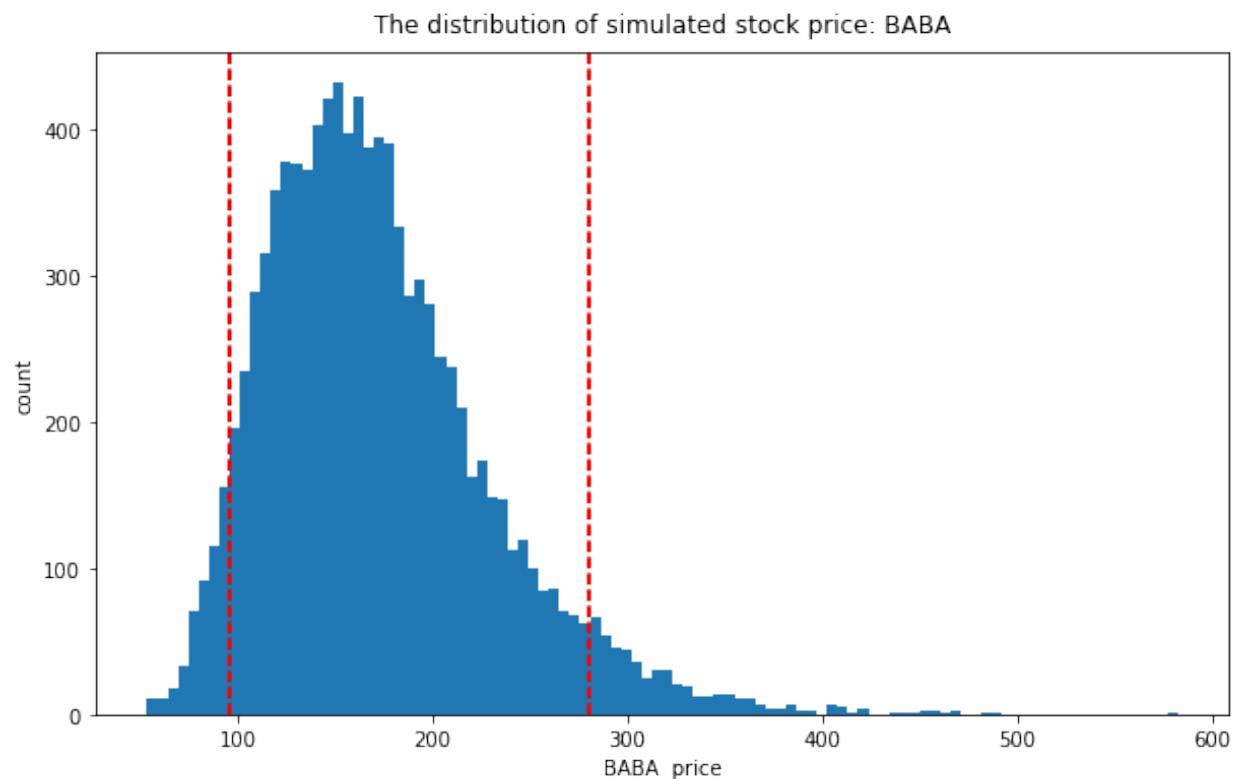
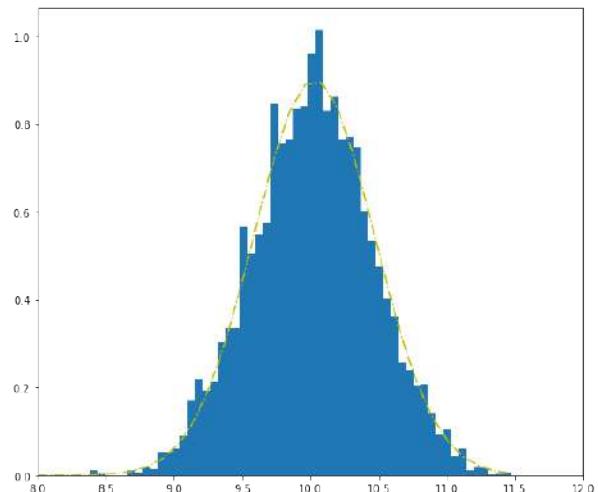
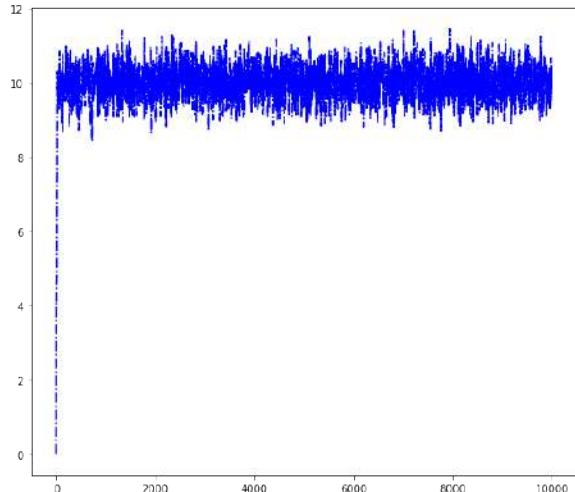


그림. 4: 모의 주가 분포

마르코프 체인 몬테 카를로

중국 속담

필요한 시기에 책은 알려져 있습니다



몬테카를로 시뮬레이션은 난수를 반복적으로 생성하여 고정된 매개변수를 추정하는 방법입니다. 보다 자세한 내용은 마르코프 체인 몬테카를로 방법에 대한 Zero 수학 도입에서 확인할 수 있습니다.

마코프 체인 몬테카를로(MCMC) 방법은 확률적 공간에서 무작위 샘플링에 의해 관심 파라미터의 사후 분포를 근사화하는 데 사용됩니다. 보다 자세한 내용은 마코프 체인 몬테카를로 방법에 대한 Zero 수학 도입에서 확인할 수 있습니다.

다음 이론과 예모는 레베카 C. 스토츠 박사의(Dr. Rebecca C. Steorts) 마코프 체인 몬테카를로의 도입입니다. 자세한 내용은 레베카 C. 스토츠 박사의 STA 360/601: 듀크 대학의 베이지안 방법과 현대 통계 수업에서 확인할 수 있습니다.

18.1 메트로폴리스 알고리즘

메트로폴리스 알고리즘은 크게 세 가지 단계를 거칩니다:

1. 샘플링합니다 $\theta^* \sim J(\theta|\theta^{(s)})$
2. 수용률을 계산합니다 (r)

$$r = \frac{p(\theta^*|y)}{p(\theta^{(s)}|y)} = \frac{p(y|\theta^*)p(\theta^*)}{p(y|\theta^{(s)})p(\theta^{(s)})}$$

3. 다음과 같이 정의합니다

$$\theta^{(s+1)} = \begin{cases} \theta^* & \text{확률 } \min(r, 1) \text{인 경우} \\ \theta^{(s)} & \text{그렇지 않은 경우} \end{cases} \quad (18.1)$$

노트: 실제로 단계 3의 (18.1)은 $u \sim \text{Uniform}(0, 1)$ 을 샘플링하고 $u < r$ 일 경우 $\theta^{(s+1)} = \theta^*$ 을 설정하고 그렇지 않으면 $\theta^{(s+1)} = \theta^{(s)}$ 을 설정하는 것으로 대체할 수 있습니다.

18.2 메트로폴리스의 토이 예제

다음 예제에서는 분산을 알 수 있을 때 켈레 정규-정규 모형에 대한 메트로폴리스 알고리즘을 테스트하려고 합니다.

18.2.1 켈레 정규-정규 모델

$$X_1, \dots, X_n \quad \theta \stackrel{iid}{\sim} \text{Normal}(\theta, \sigma^2) \\ \theta \sim \text{Normal}(\mu, \tau^2)$$

θ 의 사후 확률 분포가 $\text{Normal}(\mu_n, \tau_n^2)$ 임을 상기하고, 여기서

$$\mu_n = \bar{x} \frac{n/\sigma^2}{n/\sigma^2 + 1/\tau^2} + \mu \frac{1/\tau^2}{n/\sigma^2 + 1/\tau^2}$$

그리고

$$\tau_n^2 = \frac{1}{n/\sigma^2 + 1/\tau^2}$$

18.2.2 예제 설정

나머지 모수는 $\sigma^2 = 1$, $\tau^2 = 10$, $\mu = 5$, $n = 5$ 입니다. 그리고

$$y = [9.37, 10.18, 9.16, 11.60, 10.33]$$

이 설정의 경우 $\mu_n = 10.02745$ 및 $\tau_n^2 = 0.1960784$ 를 얻을 수 있습니다.

18.2.3 필수적인 수리 유도

메트로폴리스 알고리즘에서는 허용 비율 r 를 계산해야 합니다. 즉,

$$\begin{aligned} r &= \frac{p(\theta^*|x)}{p(\theta^{(s)}|x)} \\ &= \frac{p(x|\theta^*)p(\theta^*)}{p(x|\theta^{(s)})p(\theta^{(s)})} \\ &= \left(\frac{\prod_i \text{dnorm}(x_i, \theta^*, \sigma)}{\prod_i \text{dnorm}(x_i, \theta^{(s)}, \sigma)} \right) \left(\frac{\text{dnorm}(\theta^*, \mu, \tau)}{\text{dnorm}(\theta^{(s)}, \mu, \tau)} \right) \end{aligned}$$

많은 경우에, 비율 r 를 직접 계산하는 것은 수치적으로 불안정할 수 있지만, 이것은 $logr$ 를 취함으로써 다음과 같이 변형될 수 있습니다. 즉,

$$\begin{aligned} logr &= \sum_i \left(\log[\text{dnorm}(x_i, \theta^*, \sigma)] - \log[\text{dnorm}(x_i, \theta^{(s)}, \sigma)] \right) \\ &\quad + \sum_i \left(\log[\text{dnorm}(\theta^*, \mu, \tau)] - \log[\text{dnorm}(\theta^{(s)}, \mu, \tau)] \right) \end{aligned}$$

그러면 수용 기준은 다음과 같습니다: 만약 $logu < logr$, 여기서 u 는 Uniform(0, 1)의 샘플입니다.

18.3 데모

이제, 우리는 $\theta^{(0)} = 0$ 에서 시작하여 정규 제안 분포를 사용하여 메트로폴리스 알고리즘의 반복 S 을 생성합니다. 여기서,

$$\theta^{(s+1)} \sim \text{Normal}(\theta^{(s)}, 2).$$

18.3.1 R에서의 결과

```
# 값을 설정합니다
set.seed(1)
s2<-1
t2<-10
mu<-5; n<-5

# rnorm 을 소수 2자리로 반올림합니다
y<-round(rnorm(n,10,1),2)
# 정규 사후 확률 분포의 평균
mu.n<-( mean(y)*n/s2 + mu/t2 ) / ( n/s2+1/t2 )
# 정규 사후 확률 분포의 분산
t2.n<-1/(n/s2+1/t2)
# 데이터를 정의합니다
y<-c(9.37, 10.18, 9.16, 11.60, 10.33)

##### 메트로폴리스 알고리즘 부분#####
## S = 총 시뮬레이션 횟수
theta<-0 ; delta<-2 ; S<-10000 ; THETA<-NULL ; set.seed(1)
for(s in 1:S){
  ## 시뮬레이션
  #theta의 새 값
  #print(theta)
  theta.star<-rnorm(1,theta,sqrt(delta))
  ## 비율 r에 log를 취합니다
  log.r<-sum(dnorm(y,theta.star,sqrt(s2),log=TRUE))+dnorm(theta.star,mu,sqrt(t2),log=TRUE))-sum(dnorm(y,theta,sqrt(s2),log=TRUE))+dnorm(theta,mu,sqrt(t2),log=TRUE))
  #print(log.r)
  if(log(runif(1))<log.r) { theta<-theta.star }
  ## THETA를 업데이트합니다
  #print(log(runif(1)))
  THETA<-c(THETA,theta)
}

## 2개의 플롯: theta의 트레이스(trace)과 실제 사후 확률에 시뮬레이션된 값을
##의 경험적 분포 비교
par(mar=c(3,3,1,1),mgp=c(1.75,.75,0))
par(mfrow=c(1,2))
# 시퀀스를 생성합니다
skeep<-seq(10,S,by=10)
# 트레이스를 그릴 플롯을 생성합니다
plot(skeep,THETA[skeep],type="l",
      xlab="iteration",ylab=expression(theta))
# 히스토그램을 생성합니다
hist(THETA[-(1:50)],prob=TRUE,main="",
      xlab=expression(theta),ylab="density")
th<-seq(min(THETA),max(THETA),length=100)
lines(th,dnorm(th,mu.n,sqrt(t2.n)) )
```

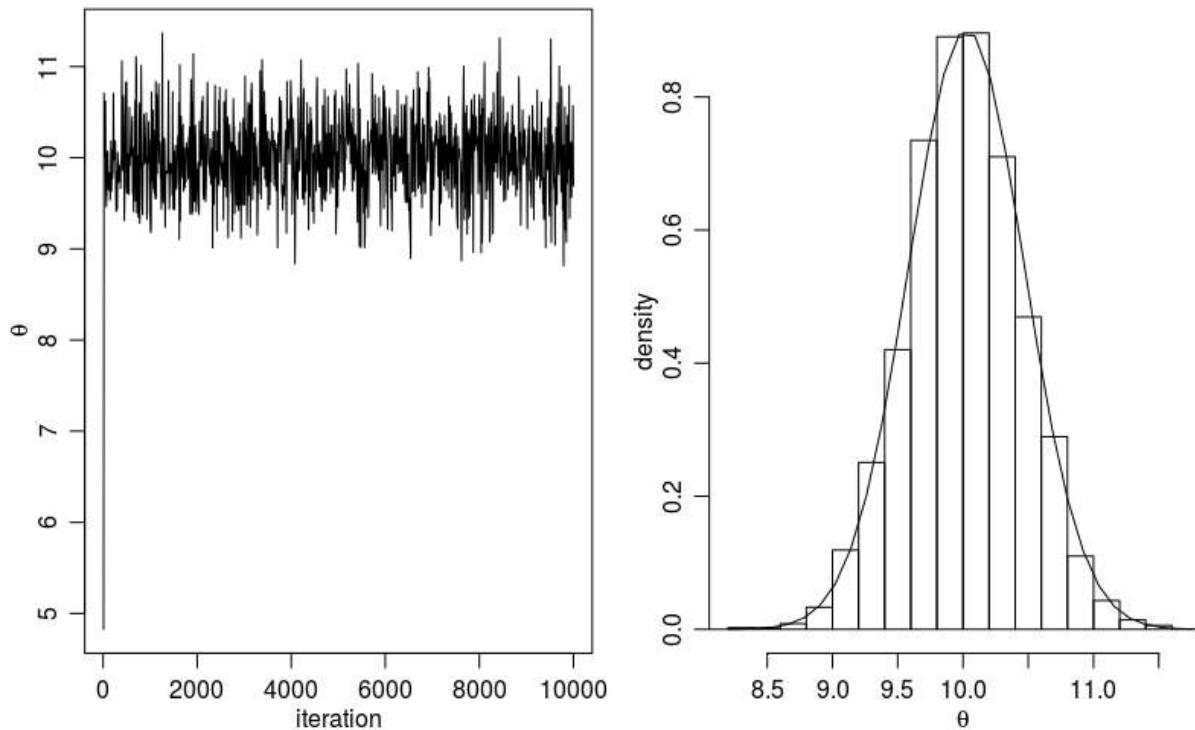


그림. 1: 비율 r 을 사용한 메트로폴리스 알고리즘 히스토그램

그림. 비율 r 을 사용한 메트로폴리스 알고리즘 히스토그램은 실행에 대한 트레이스 플롯과 메트로폴리스 알고리즘의 히스토그램을 실제 정규 밀도에서 추출한 것과 비교하여 보여줍니다.

18.3.2 Python에서의 결과

```
# coding: utf-8

# In[1]:


import numpy as np


# In[2]:


from scipy.stats import norm


def rnorm(n, mean, sd):
    """
    r에서 rnorm과 같은 함수
    r: rnorm(n, mean=0, sd=1)
    py: rvs(loc=0, scale=1, size=1, random_state=None)
    """
    return norm.rvs(loc=mean, scale=sd, size=n)
```

(다음 페이지에 계속)

(continued from previous page)

```
def dnorm(x,mean,sd, log=False):
    """
    r에서 dnorm과 같은 함수
    dnorm(x, mean=0, sd=1, log=False)
    pdf(x, loc=0, scale=1)
    """
    if log:
        return np.log(norm.pdf(x=x,loc=mean,scale=sd))
    else:
        return norm.pdf(x=x,loc=mean,scale=sd)

def runif(n,min=0, max=1):
    """
    r: runif(n, min = 0, max = 1)
    py: random.uniform(low=0.0, high=1.0, size=None)
    """
    return np.random.uniform(min,max,size=n)

# In[3]:
s2 = 1
t2 = 10
mu = 5
n = 5

# In[4]:
y = rnorm(n,10,1)
y

# In[5]:
# 정규 사후 확률 분포의 평균
mu_n = (np.mean(y)*n/s2 + mu/float(t2)) / (n/float(s2)+1/float(t2))
mu_n

# In[6]:
# 정규 사후 확률 분포의 분산
# t2.n<-1/(n/s2+1/t2)

t2_n = 1.0 / (n/float(s2)+1.0/t2)
t2_n

# In[7]:
```

(다음 페이지에서 계속)

(이전 페이지에서 계속)

```
# 데이터를 정의합니다
# y<-c(9.37, 10.18, 9.16, 11.60, 10.33)

y = [9.37, 10.18, 9.16, 11.60, 10.33]

# In[8]():

mu_n = (np.mean(y)*n/s2 + mu/float(t2))/(n/float(s2)+1/float(t2))
mu_n

# In[9]():

####메트로폴리스 알고리즘 부분#####
##S = total num of simulations
# theta<-0 ; delta<-2 ; S<-10000 ; THETA<-NULL ; set.seed(1)

theta = 0
delta = 2

S = 10000

theta_v = []

# In[ ]():

for s in range(S):
    theta_star = norm.rvs(theta,np.sqrt(delta),1)
    logr = (sum(dnorm(y,theta_star,np.sqrt(s2),log=True)) +
             sum(dnorm(theta_star,mu,np.sqrt(t2),log=True))-
             (sum(dnorm(y,theta,np.sqrt(s2),log=True)) +
              sum(dnorm([theta],mu,np.sqrt(t2),log=True))))
    #print(logr)
    if np.log(runif(1))<logr:
        theta = theta_star
    #print(theta)
    theta_v.append(theta)

# In[ ]():

import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 8))

plt.subplot(1, 2, 1)
plt.plot(theta_v,'b-.' )
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
plt.subplot(1, 2, 2)
#bins = np.arange(0, S, 10)
plt.hist(theta_v, density=True, bins='auto')
x = np.linspace(min(theta_v), max(theta_v), 100)
y = norm.pdf(x, mu_n, np.sqrt(t2_n))
plt.plot(x, y, 'y-.')
plt.xlim(right=12) # 왼쪽은 바꾸지 않은 채 오른쪽만 조정합니다
plt.xlim(left=8) # 오른쪽은 바꾸지 않은 채 왼쪽만 조정합니다
plt.show()
```

In[]:

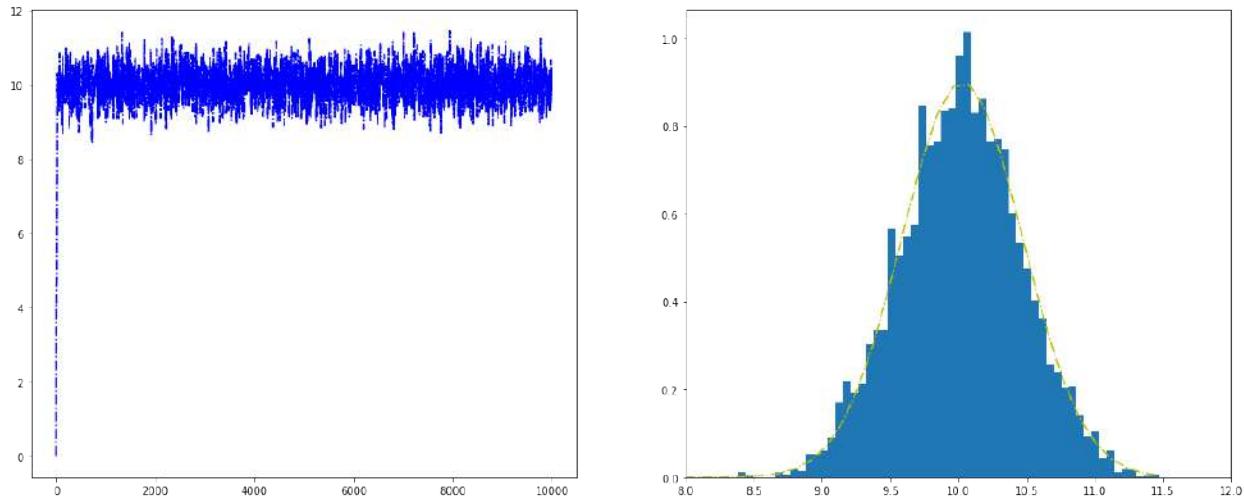


그림. 2: 파이썬으로 그린 메트로폴리스 알고리즘 히스토그램

그림. 파이썬으로 그린 메트로폴리스 알고리즘 히스토그램은 실행에 대한 추적 그림과 메트로 폴리스 알고리즘에 대한 히스토그램을 실제 정규 밀도에서 추출한 것과 비교하여 보여줍니다.

18.3.3 PySpark에서의 결과

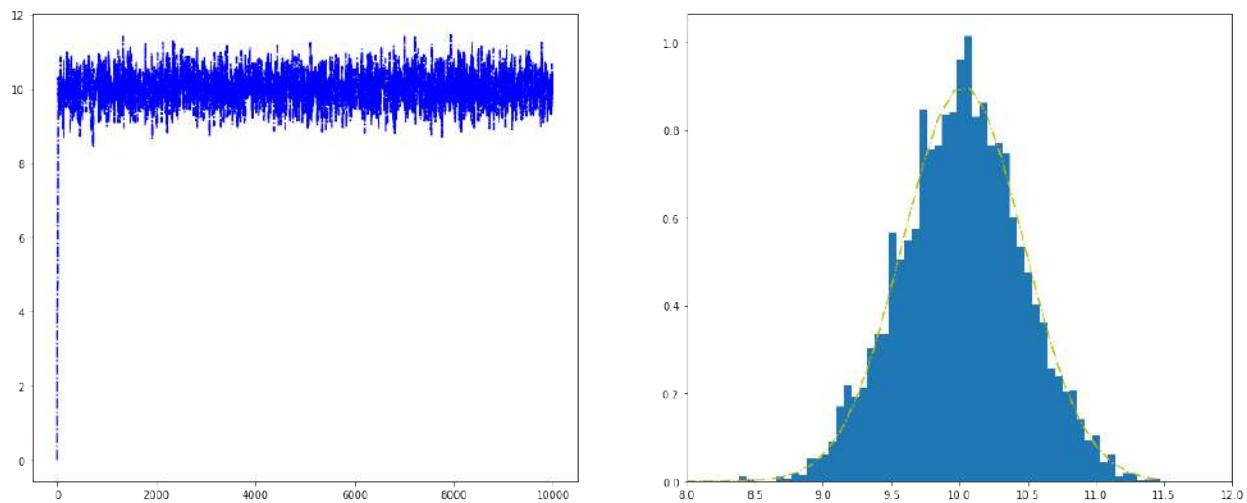


그림. 3: PySpark로 그린 메트로폴리스 알고리즘 히스토그램

그림. [PySpark로 그린 메트로폴리스 알고리즘 히스토그램](#)은 실행에 대한 추적 그림과 메트로 폴리스 알고리즘에 대한 히스토그램을 실제 정규 밀도에서 추출한 것과 비교하여 보여줍니다.

뉴럴 네트워크

중국 속담

칼을 더 오래 가는 것은 장작을 쉽게 부술 수 있습니다 – 중국 옛 속담

19.1 순방향 신경망

19.1.1 도입

순방향 뉴럴 네트워크는 유닛들 사이의 연결이 순환을 이루지 않는 인공 뉴럴 네트워크로서, 순환 뉴럴 네트워크와는 다릅니다.

순방향 신경망은 고안된 최초의 가장 간단한 인공 신경망 유형입니다. 이 네트워크에서 정보는 입력 노드에서 히든 노드(있는 경우)를 거쳐 출력 노드로 전진하는 (그림. [다층 뉴럴 네트워크](#)) 한 방향으로만 이동합니다. 네트워크에는 사이클이나 루프가 없습니다.

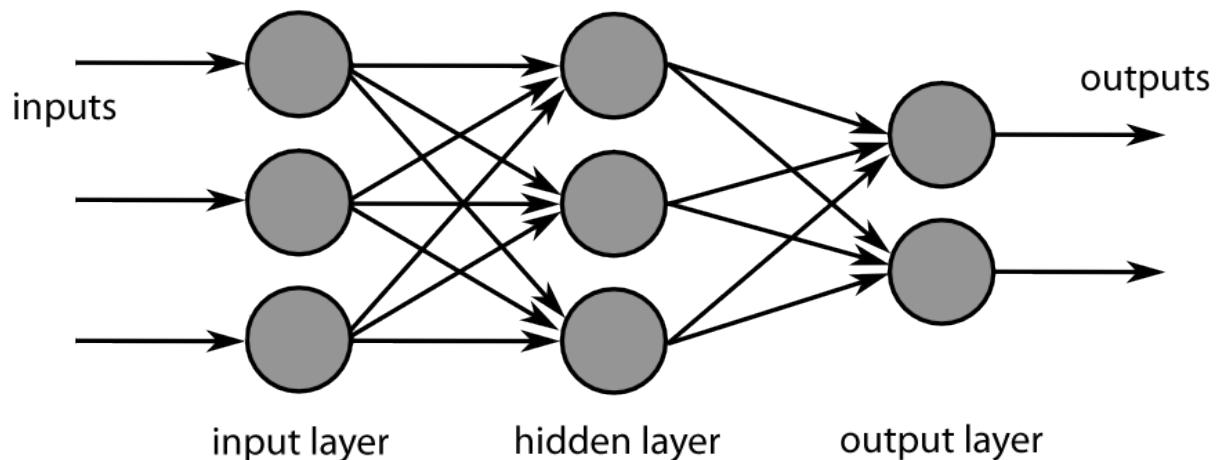


그림. 1: [다층 뉴럴 네트워크](#)

19.1.2 테모

1. 스파크 컨텍스트 및 스파크세션 설정

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark Feedforward neural network example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. 데이터셋 로드하기

```
+---+-----+-----+-----+-----+-----+-----+-----+-----+
|fixed|volatile|citric|sugar|chlorides|free|total|density| 
|pH|sulphates|alcohol|quality|
+---+-----+-----+-----+-----+-----+-----+-----+
| 7.4| 0.7| 0.0| 1.9| 0.076|11.0| 34.0| 0.9978|3.51| 0.56| 
| 9.4| 5| | | | | | | |
| 7.8| 0.88| 0.0| 2.6| 0.098|25.0| 67.0| 0.9968| 3.2| 0.68| 
| 9.8| 5| | | | | | | |
| 7.8| 0.76| 0.04| 2.3| 0.092|15.0| 54.0| 0.997|3.26| 0.65| 
| 9.8| 5| | | | | | | |
| 11.2| 0.28| 0.56| 1.9| 0.075|17.0| 60.0| 0.998|3.16| 0.58| 
| 9.8| 6| | | | | | | |
| 7.4| 0.7| 0.0| 1.9| 0.076|11.0| 34.0| 0.9978|3.51| 0.56| 
| 9.4| 5| | | | | | | |
+---+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

3. 수치형 변수를 속성값의 범위에 따라 범주형 변수로 변환하기

```
# float형식을 string형식으로 변환하기
def string_to_float(x):
    return float(x)

def condition(r):

    if (0 <= r <= 4):
        label = "low"
    elif(4 < r <= 6):
        label = "medium"
    else:
        label = "high"
    return label
```

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, DoubleType
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
string_to_float_udf = udf(string_to_float, DoubleType())
quality_udf = udf(lambda x: condition(x), StringType())
df= df.withColumn("quality", quality_udf("quality"))
```

4. 데이터를 고밀도 벡터로 변환하기

```
# 데이터를 고밀도 벡터로 변환합니다
def transData(data):
    return data.rdd.map(lambda r: [r[-1], Vectors.dense(r[:-1])]).\
        toDF(['label', 'features'])

from pyspark.sql import Row
from pyspark.ml.linalg import Vectors

data= transData(df)
data.show()
```

5. 데이터를 훈련 및 테스트 셋으로 분할하기(40%는 테스트 셋).

```
# 데이터를 훈련 및 테스트 셋으로 분할합니다
(trainingData, testData) = data.randomSplit([0.6, 0.4])
```

6. 뉴럴 네트워크 훈련하기

```
# 신경망에 대한 레이어를 지정합니다:
# 크기 11(features)의 입력층, 크기 5와 4의 중간 2개 은닉층 그리고 크기 7(classes)의
# 출력
layers = [11, 5, 4, 4, 3, 7]

# 학습자를 만들고 파라미터들을 설정합니다
FNN = MultilayerPerceptronClassifier(labelCol="indexedLabel", \
                                       featuresCol="indexedFeatures", \
                                       maxIter=100, layers=layers, \
                                       blockSize=128, seed=1234)

# 인덱싱된 라벨들을 본래 라벨들로 변환합니다
labelConverter = IndexToString(inputCol="prediction", outputCol=
                                "predictedLabel",
                                labels=labelIndexer.labels)

# 파이프라인에 인덱서들과 포레스트(forest)를 묶습니다
from pyspark.ml import Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, FNN, \
                           labelConverter])

# 모형을 훈련합니다
# 모형을 훈련합니다. 인덱서들도 같이 실행됩니다.
model = pipeline.fit(trainingData)
```

7. 예측값 만들기

```
# 예측값 생성합니다
predictions = model.transform(testData)
# 표시할 예제 행을 선택합니다
predictions.select("features", "label", "predictedLabel").show(5)
```

8. 평가하기

```
# (예측값, 실제 값) 을 선택하고 검증 오차 계산하기
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy"
    ↵")
accuracy = evaluator.evaluate(predictions)
print("Predictions accuracy = %g, Test Error = %g" % (accuracy, (1.0 - ↵
    ↵accuracy)))
```

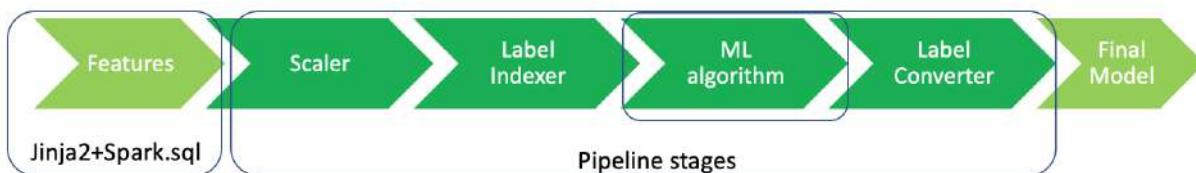
CLOUDERA DISTRIBUTION HADOOP 자동화

CDH(Cloudera Distribution Hadoop)은 Apache Hadoop에서 가장 완벽하고, 테스트되었으며, 널리 배포된 배포판입니다. 많은 중소기업들이 CDH를 사용하고 있습니다. Cloudera는 CDH에서 IPython 또는 Jupyter 노트북을 지원하지 않습니다. Cloudera Data Science Workbench는 고가여서 많은 회사들이 CDH+zeppelin 또는 CDH+jupyterhub 인프라를 사용하고 있습니다. 이 인프라는 꽤 잘 작동하지만 데이터 엔지니어나 데이터 사이언티스트가 생산 과정에서 자동화를 수행하기엔 불편합니다. 이 장에서는 Cloudera Distribution Hadoop용 자동화를 구현하기 위해 [Jinja2](#), [spark sql](#) 및 [ML Pipelines](#)을 사용하는 방법에 대해 설명합니다.

20.1 자동화 파이프라인

자동화 파이프라인은 주로 두 가지 부분을 포함하고 있습니다:

1. [Jinja2 + spark sql](#)로 데이터 정제(clean) 및 조작 자동화 가능
2. 기계 학습 자동화를 위한 [ML Pipelines](#)



20.2 데이터 정제 및 조작 자동화

20.2.1 Jinja 2

Jinja는 장고의 템플릿을 본떠 만든 파이썬의 현대적이고 디자이너 친화적인 템플릿 언어입니다. SQL 쿼리를 생성하려면 Jinja2를 사용해야 합니다:

1. 템플릿 가져오기

```
temp = """
    SELECT project, timesheet, hours
    FROM timesheet
    WHERE user_id = {{ user_id }}
    {%- if project_id %}
    AND project_id = {{ project_id }}
    {%- endif %}
"""
"""
```

2. 템플릿을 렌더링하기

```
args = {"user_id": u"runawayhorse",
        "project_id": 123}

query= Template(temp).render(args)

print(query)
```

그러면 다음과 같은 SQL 쿼리를 얻을 수 있습니다:

```
SELECT project, timesheet, hours
FROM timesheet
WHERE user_id = runawayhorse

AND project_id = 123
```

노트

Jinja 는 생각보다 똑똑합니다. 다음을 시도해보면

```
args = {"user_id": u"runawayhorse"}

query= Template(temp).render(args)

print(query)
```

다음과 같은 SQL 쿼리가 나타납니다:

```
SELECT project, timesheet, hours
FROM timesheet
WHERE user_id = runawayhorse
```

긴 쿼리가 있는 경우 Jinja get_template 를 사용하여 template를 읽을 수 있습니다:

```
import os
from jinja2 import Template
from jinja2 import Environment, FileSystemLoader

path = os.path.abspath(os.path.join(sys.path[0]))
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

try:
    os.mkdir(path)
except OSError:
    pass
os.chdir(path)
print(path)

jinja_env = Environment(loader=FileSystemLoader(path))
template = jinja_env.get_template('test.sql')
query = template.render(states=states)
print(query)

```

test.sql 파일을 사용하면 다음과 같습니다:

```

select id
{%
  for var in states %
, (CASE WHEN (off_st = '{{var}}') THEN 1 ELSE 0 END) AS off_st_{{var}}
{%
  endfor %
} FROM table1

```

여러분은 다음과 같은 쿼리 결과를 확인하실 수 있습니다:

```

select id
, (CASE WHEN (off_st = 'MO') THEN 1 ELSE 0 END) AS off_st_MO
, (CASE WHEN (off_st = 'KS') THEN 1 ELSE 0 END) AS off_st_KS
, (CASE WHEN (off_st = 'KY') THEN 1 ELSE 0 END) AS off_st_KY
, (CASE WHEN (off_st = 'OH') THEN 1 ELSE 0 END) AS off_st_OH
FROM table1

```

20.2.2 Spark SQL

여기서 Spark SQL을 호출하여 기존 웨어하우스에 Jinja2에서 생성한 SQL 혹은 HiveQL 쿼리를 실행합니다.

```

# 출력 없이
spark.sql(query)

# 출력과 함께
df = spark.sql(query)

```

20.3 ML 파이프라인 자동화

ML 파이프라인에 대한 자세한 내용은 여기서 다루지 않겠습니다. 관심 있는 독자들은 [ML 파이프라인](#)을 참조하시길 바랍니다. ML 파이프라인 절차를 정의하는 주요 단계는 다음과 같습니다:

```
scaler = 'Standard'

from pyspark.ml.feature import Normalizer, StandardScaler, MinMaxScaler
if scaler=='Normal':
    scaler = Normalizer(inputCol="features", outputCol="scaledFeatures", p=1.
    ↪0)
elif scaler=='Standard':
    scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures",
                           withStd=True, withMean=False)
else:
    scaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")

from pyspark.ml.feature import StringIndexer
# 라벨들에 색인을 달고, 메타데이터를 라벨 칼럼에 추가합니다
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='label').fit(transformed)

from pyspark.ml.feature import IndexToString
# 인덱싱된 라벨들을 본래 라벨들로 변환합니다
labelConverter = IndexToString(inputCol="prediction", outputCol=
    ↪"predictedLabel",
                                labels=labelIndexer.labels)

from pyspark.ml.classification import LogisticRegression
ml = LogisticRegression(featuresCol='scaledFeatures', labelCol='label',_
    ↪maxIter=100, regParam=0.01, elasticNetParam=0.6)

# 파이프라인에 인덱서와 트리를 묶습니다
pipeline_model = Pipeline(stages=[scaler,labelIndexer,ml,labelConverter])

# 모형을 훈련합니다. 인덱서들도 같이 실행됩니다.
model = pipeline_model.fit(trainingData)

# 예측값을 생성합니다
predictions = model.transform(testData)
```

20.4 파이프라인 모델 저장 및 로드

```
# 파이프라인 모델을 저장합니다  
model.write().overwrite().save(out_path)  
  
# 파이프라인 모델을 로드합니다  
from pyspark.ml import PipelineModel  
  
model = PipelineModel.load(out_path)
```

20.5 결과를 다시 Hadoop에 제출

```
df.createOrReplaceTempView("temp_table")  
  
query = '''  
create table database_name.prediction_{dt} AS  
SELECT *  
FROM temp_table  
'''  
output = Template(query).render(dt=dt)  
spark.sql(output)
```


PYSPARK 패키지화

Python으로 본인만의 패키지를 만드는 것은 매우 간편합니다. 일상적인 작업에서 자주 사용하던 기능들을 담아두었습니다. [My PySpark Package](#)에서 다운로드하여 설치할 수 있습니다. 이 패키지의 계층구조와 디렉토리 구조는 다음과 같습니다.

21.1 패키지화

21.1.1 계층구조

```
|-- build
|   |-- bdist.linux-x86_64
|   |-- lib.linux-x86_64-2.7
|       |-- PySparkTools
|           |-- __init__.py
|           |-- Manipulation
|               |-- DataManipulation.py
|               |-- __init__.py
|           |-- Visualization
|               |-- __init__.py
|               |-- PyPlots.py
|-- dist
|   |-- PySparkTools-1.0-py2.7.egg
|-- __init__.py
|-- PySparkTools
|   |-- __init__.py
|   |-- Manipulation
|       |-- DataManipulation.py
|       |-- __init__.py
|   |-- Visualization
|       |-- __init__.py
|       |-- PyPlots.py
|       |-- PyPlots.pyc
|-- PySparkTools.egg-info
|   |-- dependency_links.txt
|   |-- PKG-INFO
|   |-- requires.txt
|   |-- SOURCES.txt
```

(다음 페이지에 계속)

(○] 전 페이지에서 계속)

```
|   |-- top_level.txt  
|-- README.md  
|-- requirements.txt  
|-- setup.py  
|-- test  
    |-- spark-warehouse  
    |-- test1.py  
    |-- test2.py
```

위의 계층 구조로부터, 여러분은 각 디렉토리에 `__init__.py` 가 있어야 한다는 것을 알게 될 것입니다. 아래 예를 들어 `__init__.py` 파일을 설명하겠습니다:

21.1.2 설정

```
from setuptools import setup, find_packages  
  
try:  
    with open("README.md") as f:  
        long_description = f.read()  
except IOError:  
    long_description = ""  
  
try:  
    with open("requirements.txt") as f:  
        requirements = [x.strip() for x in f.read().splitlines() if x.strip()]  
except IOError:  
    requirements = []  
  
setup(name='PySparkTools',  
      install_requires=requirements,  
      version='1.0',  
      description='Python Spark Tools',  
      author='Wenqiang Feng',  
      author_email='von198@gmail.com',  
      url='https://github.com/runawayhorse001/PySparkTools',  
      packages=find_packages(),  
      long_description=long_description  
)
```

21.1.3 ReadMe

```
# PySparkTools  
  
This is my PySpark Tools. If you want to clone and install it, you can use  
- clone  
``` {bash}
```

(다음 페이지에 계속)

(이전 페이지에 계속)

```
git clone git@github.com:runawayhorse001/PySparkTools.git
```
- install

``` {bash}
cd PySparkTools
pip install -r requirements.txt
python setup.py install
```

- test

``` {bash}
cd PySparkTools/test
python test1.py
```

```

21.2 PyPI에서 패키지 퍼블리싱

21.2.1 twine 설치

```
pip install twine
```

21.2.2 패키지 빌드

```
python setup.py sdist bdist_wheel
```

여러분은 새 폴더 dist를 얻을 수 있습니다:

```
.
├── PySparkAudit-1.0.0-py2.7.egg
├── PySparkAudit-1.0.0-py2-none-any.whl
└── PySparkAudit-1.0.0.tar.gz
```

21.2.3 패키지 업로드

```
twine upload dist/*
```

업로드를 처리하는 동안 PyPI 계정의 사용자 이름과 암호를 제공해야 합니다:

```
Enter your username: runawayhorse001
Enter your password: *****
```

21.2.4 PyPI에서 패키지

[PyPI](<https://pypi.org/project/PySparkAudit>)에 있는 PySparkAudit 패키지입니다.
PySparkAudit을 사용해 설치할 수 있습니다:

```
pip install PySparkAudit
```

PYSPARK 데이터 감사 라이브러리

PySparkAudit: PySpark 데이터 감사 라이브러리. PDF 버전은 [여기서 다운로드 가능합니다.](#)
python 버전 **PyAudit:** Python 데이터 감사 라이브러리 API는 [PyAudit](#)에서 확인 가능합니다.

22.1 pip로 설치하기

[PyPI](<https://pypi.org/project/PySparkAudit>)에서 PySparkAudit을 설치할 수 있습니다:

```
pip install PySparkAudit
```

22.2 Repo로부터 설치하기

22.2.1 Repository 복제하기

```
git clone https://github.com/runawayhorse001/PySparkAudit.git
```

22.2.2 설치하기

```
cd PySparkAudit  
pip install -r requirements.txt  
python setup.py install
```

22.3 삭제하기

```
pip uninstall PySparkAudit
```

22.4 테스트

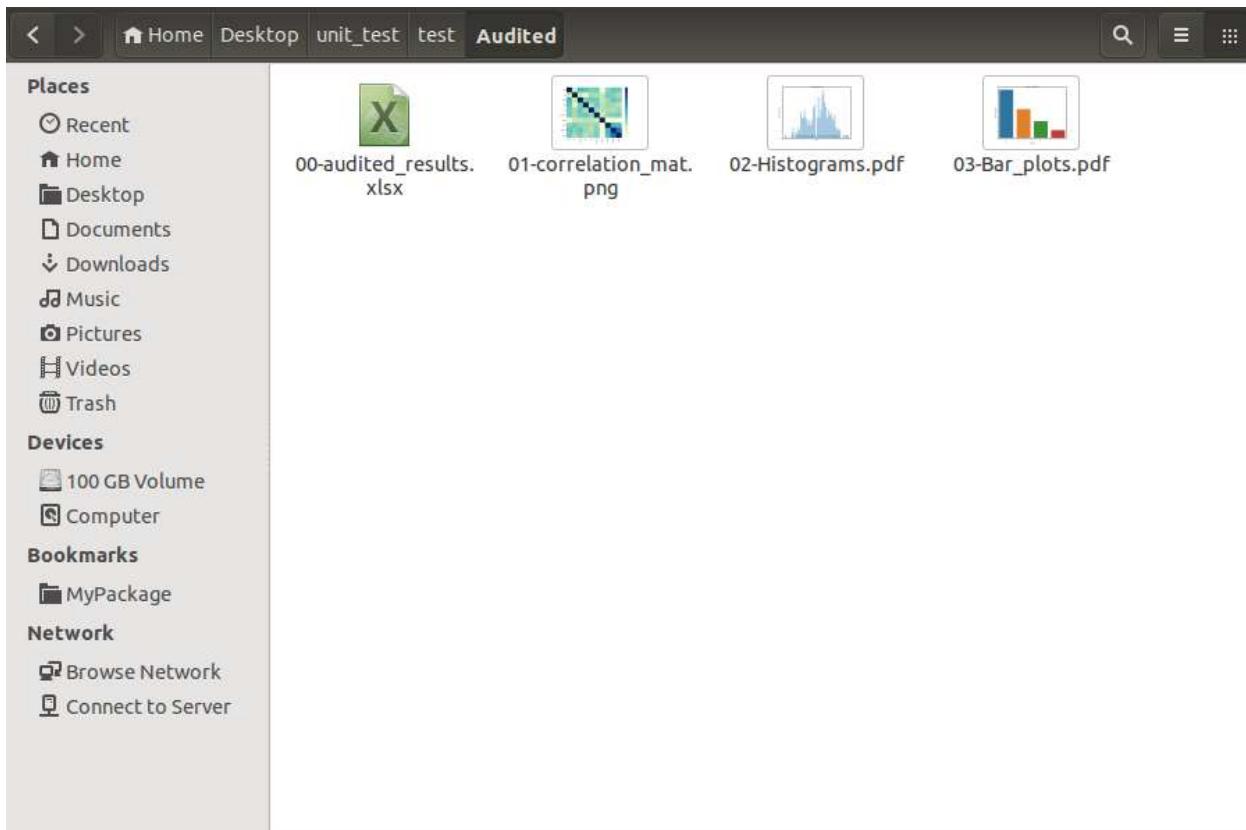
22.4.1 테스트 코드 실행하기

```
cd PySparkAudit/test  
python test.py
```

test.py

```
from pyspark.sql import SparkSession  
  
spark = SparkSession \  
.builder \  
.appName("Python Spark regression example") \  
.config("spark.some.config.option", "some-value") \  
.getOrCreate()  
  
# from PySparkAudit import dtypes_class, hist_plot, bar_plot, freq_items,  
# feature_len  
# from PySparkAudit import dataset_summary, rates, trend_plot  
  
# path = '/home/feng/Desktop'  
  
# PySpark 감사 함수를 불러옵니다  
from PySparkAudit import auditing  
  
# 데이터셋을 로드합니다  
data = spark.read.csv(path='Heart.csv',  
                      sep=',', encoding='UTF-8', comment=None, header=True,   
                      inferSchema=True)  
  
# 한 함수로 auditing  
print(auditing(data, display=True))
```

22.4.2 감사된 결과들



1-audited_results.xlsx에서 파일들:

2-Dataset_summary(데이터셋 요약)

The screenshot shows a spreadsheet application with a toolbar at the top containing various icons for file operations, font selection, and other functions. The main area displays a table titled 'Dataset_summary' with two columns: 'summary' and 'value'. The table contains 26 rows of data, starting with 'sample_size' (value 303) and ending with 'string' (value 4). Below the table, there are tabs for 'Dataset_summary', 'Numeric_summary', 'Category_summary', and 'Correlation_matrix'. At the bottom, there is a search bar labeled 'Find' and a status bar indicating 'Sheet 1 / 4' and 'PageStyle_Dataset_summary'.

| | A | B | C | D | E | F | G | H | I | J |
|----|-----------------------|---------|---|---|---|---|---|---|---|---|
| 1 | summary | value | | | | | | | | |
| 2 | sample_size | 303 | | | | | | | | |
| 3 | feature_size | 14 | | | | | | | | |
| 4 | single_unique_feature | 0 | | | | | | | | |
| 5 | row_w_null | 0 | | | | | | | | |
| 6 | row_w_empty | 0 | | | | | | | | |
| 7 | row_w_zero | 299 | | | | | | | | |
| 8 | row_avg_null_count | 0 | | | | | | | | |
| 9 | row_avg_empty_count | 0 | | | | | | | | |
| 10 | row_avg_zero_count | 3.25083 | | | | | | | | |
| 11 | numerical_fields | 10 | | | | | | | | |
| 12 | categorical_fields | 4 | | | | | | | | |
| 13 | date_fields | 0 | | | | | | | | |
| 14 | unsupported_fields | 0 | | | | | | | | |
| 15 | double | 1 | | | | | | | | |
| 16 | int | 9 | | | | | | | | |
| 17 | string | 4 | | | | | | | | |
| 18 | | | | | | | | | | |
| 19 | | | | | | | | | | |
| 20 | | | | | | | | | | |
| 21 | | | | | | | | | | |
| 22 | | | | | | | | | | |
| 23 | | | | | | | | | | |
| 24 | | | | | | | | | | |
| 25 | | | | | | | | | | |
| 26 | | | | | | | | | | |

2. Numeric_summary(수치형 변수 요약)

The screenshot shows the Apache Calc spreadsheet application. The main window displays a table titled "feature" with 11 rows and 13 columns. The columns are labeled A through K, with A being the row index. The table includes descriptive statistics for each feature: dypes, count, null count, distinct count, mean, standard deviation, minimum, maximum, Q1, and Med. The "dypes" column shows data types like int and double. The "count" column consistently shows 303. The "null count" column shows values ranging from 0 to 29. The "distinct count" column shows values ranging from 50 to 152. The "mean" column shows values ranging from 20.148514 to 41.544389. The "stddev" column shows values ranging from 0.356190 to 17.599749. The "min" column shows values ranging from 1 to 77. The "max" column shows values ranging from 0 to 564. The "Q1" column shows values ranging from 0 to 120. The "Med" column shows values ranging from 0 to 130. Below the table, there are several blank rows (12 through 25) and a footer section with tabs for "Dataset_summary", "Numeric_summary", "Category_summary", and "Correlation_matrix". The "Numeric_summary" tab is currently selected. At the bottom, there are search and filter options, and a status bar indicating "Sheet 2 / 4" and "PageStyle_Numeric_summary".

| A | feature | dtypes | count | null count | distinct count | mean | stddev | min | max | Q1 | Med |
|----|---------|--------|-------|------------|----------------|-----------|-----------|-----|-----|-----|-----|
| 1 | Age | int | 303 | 303 | 41 | 54.438949 | 9.038662 | 29 | 77 | 48 | 56 |
| 2 | Sex | int | 303 | 303 | 2 | 0.679867 | 0.467298 | 0 | 1 | 0 | 1 |
| 3 | RestBP | int | 303 | 303 | 50 | 131.6897 | 17.599749 | 4 | 200 | 120 | 130 |
| 4 | Chol | int | 303 | 303 | 152 | 246.6930 | 51.776912 | 6 | 564 | 211 | 242 |
| 5 | Fbs | int | 303 | 303 | 2 | 0.148514 | 0.356190 | 0 | 1 | 0 | 0 |
| 6 | RestECG | int | 303 | 303 | 3 | 0.990099 | 0.994971 | 0 | 2 | 0 | 1 |
| 7 | MaxHR | int | 303 | 303 | 91 | 149.6072 | 22.875007 | 1 | 202 | 134 | 153 |
| 8 | ExAng | int | 303 | 303 | 2 | 0.326732 | 0.469794 | 0 | 1 | 0 | 0 |
| 9 | Oldpeak | double | 303 | 303 | 40 | 1.039603 | 1.161075 | 0.0 | 6.2 | 0 | 0.8 |
| 10 | Slope | int | 303 | 303 | 3 | 1.600660 | 0.616226 | 1 | 3 | 1 | 2 |
| 11 | | | | | | | | | | | |
| 12 | | | | | | | | | | | |
| 13 | | | | | | | | | | | |
| 14 | | | | | | | | | | | |
| 15 | | | | | | | | | | | |
| 16 | | | | | | | | | | | |
| 17 | | | | | | | | | | | |
| 18 | | | | | | | | | | | |
| 19 | | | | | | | | | | | |
| 20 | | | | | | | | | | | |
| 21 | | | | | | | | | | | |
| 22 | | | | | | | | | | | |
| 23 | | | | | | | | | | | |
| 24 | | | | | | | | | | | |
| 25 | | | | | | | | | | | |
| 26 | | | | | | | | | | | |

3. Category_summary(범주형 변수 요약)

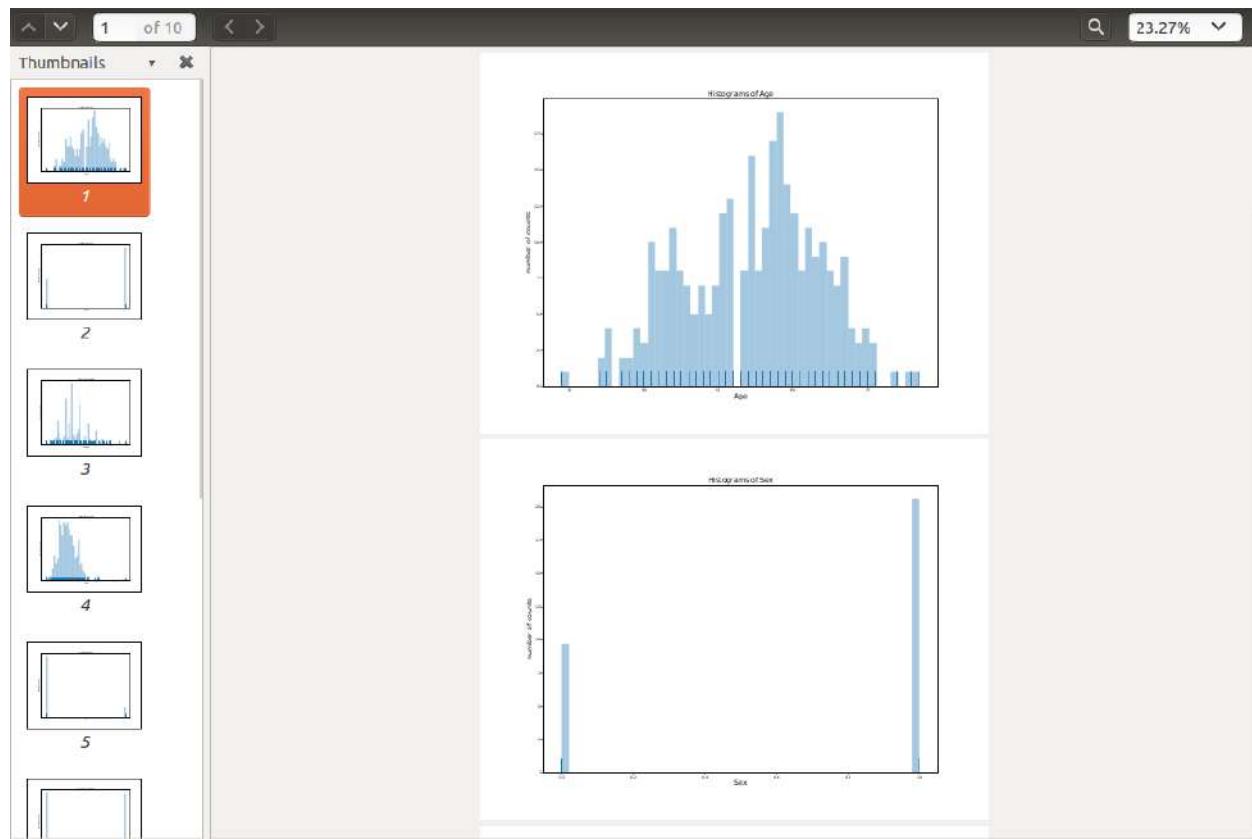
4. Correlation_matrix (상관 행렬)

The screenshot shows a correlation matrix in a spreadsheet application. The columns are labeled A through K, and the rows are numbered 1 through 26. The matrix includes the following columns:

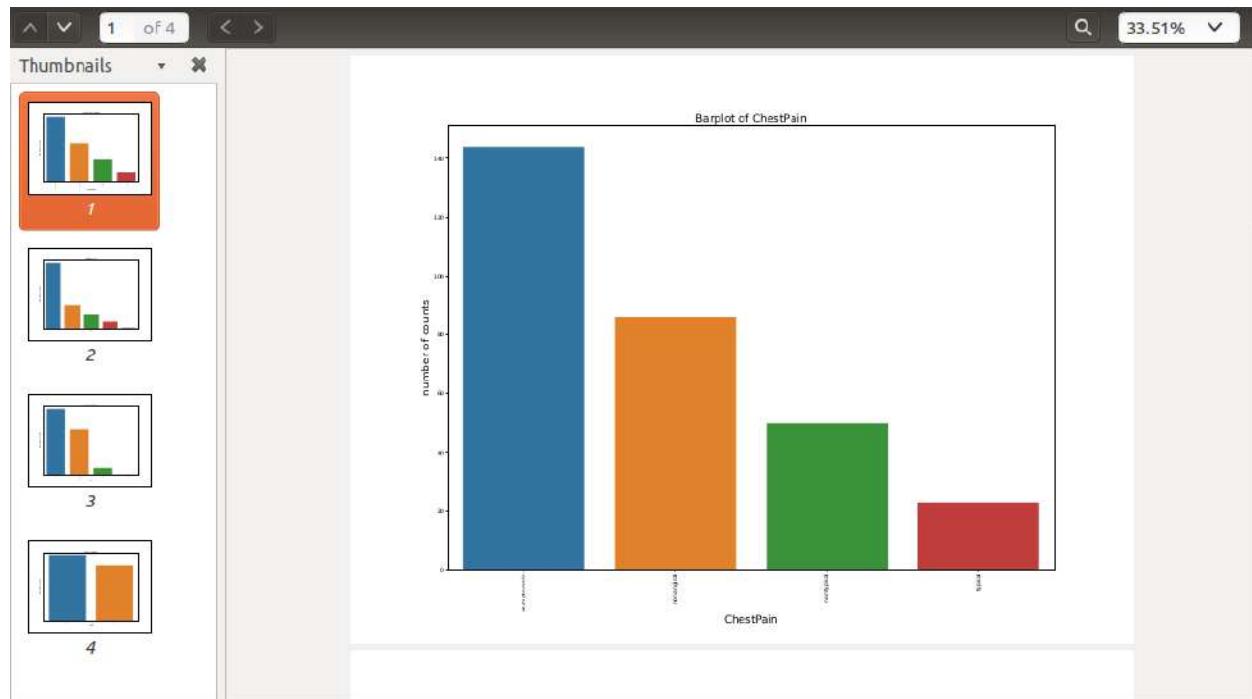
| | A | B | C | D | E | F | G | H | I | J | K |
|----|------------|------------|---------------|-------------|------------|----------------|--------------|--------------|----------------|--------------|---|
| 1 | Age | Sex | RestBP | Chol | Fbs | RestECG | MaxHR | ExAng | Oldpeak | Slope | |
| 2 | 1 | -0.09754 | 0.28495 | 0.20895 | 0.11853 | 0.14887 | -0.39381 | 0.09166 | 0.20381 | 0.16177 | |
| 3 | -0.09754 | 1 | -0.06446 | -0.19991 | 0.04786 | 0.02165 | -0.04866 | 0.1462 | 0.10217 | 0.03753 | |
| 4 | 0.28495 | -0.06446 | 1 | 0.13012 | 0.17534 | 0.14656 | -0.04535 | 0.06476 | 0.18917 | 0.11738 | |
| 5 | 0.20895 | -0.19991 | 0.13012 | 1 | 0.00984 | 0.17104 | -0.00343 | 0.06131 | 0.04656 | -0.00406 | |
| 6 | 0.11853 | 0.04786 | 0.17534 | 0.00984 | 1 | 0.06956 | -0.00785 | 0.02567 | 0.00575 | 0.05989 | |
| 7 | 0.14887 | 0.02165 | 0.14656 | 0.17104 | 0.06956 | 1 | -0.08339 | 0.08487 | 0.11413 | 0.13395 | |
| 8 | -0.39381 | -0.04866 | -0.04535 | -0.00343 | -0.00785 | -0.08339 | 1 | -0.3781 | -0.34309 | -0.3856 | |
| 9 | 0.09166 | 0.1462 | 0.06476 | 0.06131 | 0.02567 | 0.08487 | -0.3781 | 1 | 0.28822 | 0.25775 | |
| 10 | 0.20381 | 0.10217 | 0.18917 | 0.04656 | 0.00575 | 0.11413 | -0.34309 | 0.28822 | 1 | 0.57754 | |
| 11 | 0.16177 | 0.03753 | 0.11738 | -0.00406 | 0.05989 | 0.13395 | -0.3856 | 0.25775 | 0.57754 | 1 | |
| 12 | | | | | | | | | | | |
| 13 | | | | | | | | | | | |
| 14 | | | | | | | | | | | |
| 15 | | | | | | | | | | | |
| 16 | | | | | | | | | | | |
| 17 | | | | | | | | | | | |
| 18 | | | | | | | | | | | |
| 19 | | | | | | | | | | | |
| 20 | | | | | | | | | | | |
| 21 | | | | | | | | | | | |
| 22 | | | | | | | | | | | |
| 23 | | | | | | | | | | | |
| 24 | | | | | | | | | | | |
| 25 | | | | | | | | | | | |
| 26 | | | | | | | | | | | |

The status bar at the bottom shows the path: Dataset_summary \ Numeric_summary \ Category_summary \ Correlation_matrix, and the page style is PageStyle_Correlation_matrix.

5. Histograms.pdf에서 히스토그램들



6. Bar_plots.pdf에서 막대형 그래프들



22.5 빅 데이터 셋 감사

본 절에서는 빅데이터 세트에 대한 감사 실적 및 감사 결과를 보여드리겠습니다. 데이터 세트는 스페인 고속철도 승차권 가격입니다: <https://www.kaggle.com/thegurus/spanish-high-speed-rail-system-ticket-pricing>. 이 데이터 세트에는 2579771 샘플과 10가지 특징이 있습니다.

다음 CPU 시간으로부터 히스토그램을 그리는 데 소요된 대부분의 시간을 확인할 수 있습니다. 시간과 메모리가 제한되어 있다면 `sample_size`를 사용하여 히스토그램을 그리기 위한 데이터 셋의 부분 집합을 생성하는 것을 제안합니다.

예를 들어:

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark regression example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

# from PySparkAudit import dtypes_class, hist_plot, bar_plot, freq_items,
# feature_len
# from PySparkAudit import dataset_summary, rates, trend_plot

# 감사 결과 출력 경로
out_path = '/home/feng/Desktop'

# PySpark 감사 함수를 불러옵니다
from PySparkAudit import auditing

# 데이터를 로드합니다
# 스페인 고속철도 렌페 승차권 가격 - (~2.58M)
# https://www.kaggle.com/thegurus/spanish-high-speed-rail-system-ticket-
# pricing

data = spark.read.csv(path='/home/feng/Downloads/renfe.csv',
                      sep=',', encoding='UTF-8', comment=None, header=True,
                      inferSchema=True)

# 한 함수로 auditing
auditing(data, output_dir=out_path, tracking=True)
```

결과:

22.5.1 bash에서 출력

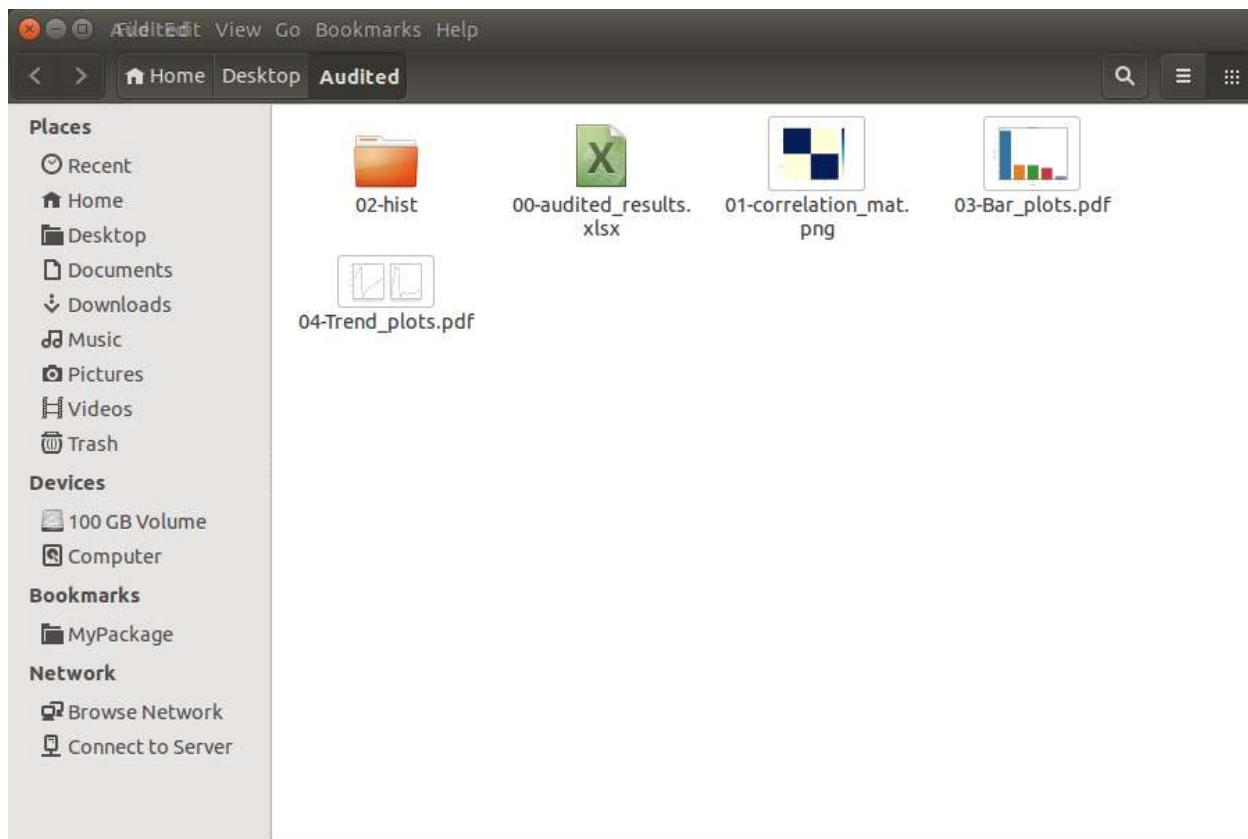
```
=====
The audited results summary audited_results.xlsx was located at:
/home/feng/Desktop/Audited
Generate data set summary took = 60.535009145736694 s
=====
Collecting data types.... Please be patient!
Generate counts took = 0.0016515254974365234 s
=====
Collecting features' counts.... Please be patient!
Generate counts took = 6.502962350845337 s
=====
Collecting data frame description.... Please be patient!
Generate data frame description took = 1.5562639236450195 s
=====
Calculating percentiles.... Please be patient!
Generate percentiles took = 19.76785445213318 s
=====
Calculating features' length.... Please be patient!
Generate features' length took = 4.953453540802002 s
=====
Calculating top 5 frequent items.... Please be patient!
Generate rates took: 4.761325359344482 s
=====
Calculating rates.... Please be patient!
Generate rates took: 17.201056718826294 s
Auditing numerical data took = 54.77840781211853 s
=====
Collecting data types.... Please be patient!
Generate counts took = 0.001623392105102539 s
=====
Collecting features' counts.... Please be patient!
Generate counts took = 12.59226107597351 s
=====
Calculating features' length.... Please be patient!
Generate features' length took = 5.332952976226807 s
=====
Calculating top 5 frequent items.... Please be patient!
Generate rates took: 6.832213878631592 s
=====
Calculating rates.... Please be patient!
Generate rates took: 23.704302072525024 s
Auditing categorical data took = 48.484763622283936 s
=====
The correlation matrix plot Corr.png was located at:
/home/feng/Desktop/Audited
Calculating correlation matrix... Please be patient!
Generate correlation matrix took = 19.61273431777954 s
=====
The Histograms plots *.png were located at:
/home/feng/Desktop/Audited/02-hist
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
Plotting histograms of _c0.... Please be patient!
Plotting histograms of price.... Please be patient!
Histograms plots are DONE!!!
Generate histograms plots took = 160.3421311378479 s
=====
The Bar plot Bar_plots.pdf was located at:
/home/feng/Desktop/Audited
Plotting barplot of origin.... Please be patient!
Plotting barplot of destination.... Please be patient!
Plotting barplot of train_type.... Please be patient!
Plotting barplot of train_class.... Please be patient!
Plotting barplot of fare.... Please be patient!
Plotting barplot of insert_date.... Please be patient!
Plotting barplot of start_date.... Please be patient!
Plotting barplot of end_date.... Please be patient!
Bar plots are DONE!!!
Generate bar plots took = 24.17994236946106 s
=====
The Trend plot Trend_plots.pdf was located at:
/home/feng/Desktop/Audited
Plotting trend plot of _c0.... Please be patient!
Plotting trend plot of price.... Please be patient!
Trend plots are DONE!!!
Generate trend plots took = 11.697550296783447 s
Generate all the figures took = 196.25823402404785 s
Generate all audited results took = 379.73954820632935 s
=====
The auditing processes are DONE!!!
```

22.5.2 감사된 결과 풀더



JUPYTER NOTEBOOK에서 ZEPPELIN

Zeppelin 사용자는 저처럼 Zeppelin .json 노트북을 열거나 읽기 어렵다는 문제를 가지고 있을 수 있습니다. **ze2nb**: Zeppelin .json 노트북을 .ipynb Jupyter 노트북, .py 및 .html 파일로 변환하기 위한 코드 조각. 이 라이브러리는 Ryan Blue의 Jupyter/Zeppelin 변환을 기반으로 합니다: [jupyter-zeppelin]. API 북은 [ze2nb API](#) 또는 [\[zeppelin2nb\]](#)에서 찾을 수 있습니다. 여러분은 이를 다운로드하여 배포할 수 있습니다. 그러나 노트에는 오타뿐만 아니라 부정확하거나 잘못된 설명이 포함되어 있습니다.

23.1 설치 방법

23.1.1 pip로 설치하기

ze2nb 는 [PyPI](<https://pypi.org/project/ze2nb>)에서 설치할 수 있습니다:

```
pip install ze2nb
```

23.1.2 Repo에서 설치하기

1. Repository 복제하기

```
git clone https://github.com/runawayhorse001/ze2nb.git
```

2. 설치하기

```
cd zeppelin2nb
pip install -r requirements.txt
python setup.py install
```

23.1.3 삭제하기

```
pip uninstall ze2nb
```

23.2 데모 변환

다음 데모는 zeppelin2nb 를 사용하여 .json 을 .ipynb , .py 및 .html로 변환하는 방법을 보여주도록 설계되었습니다.

23.2.1 한 함수로 변환하기

예를 들어:

```
# python 라이브러리 불러오기
import os, sys

# zeppelin2nb 모듈 불러오기
from ze2nb import ze2nb

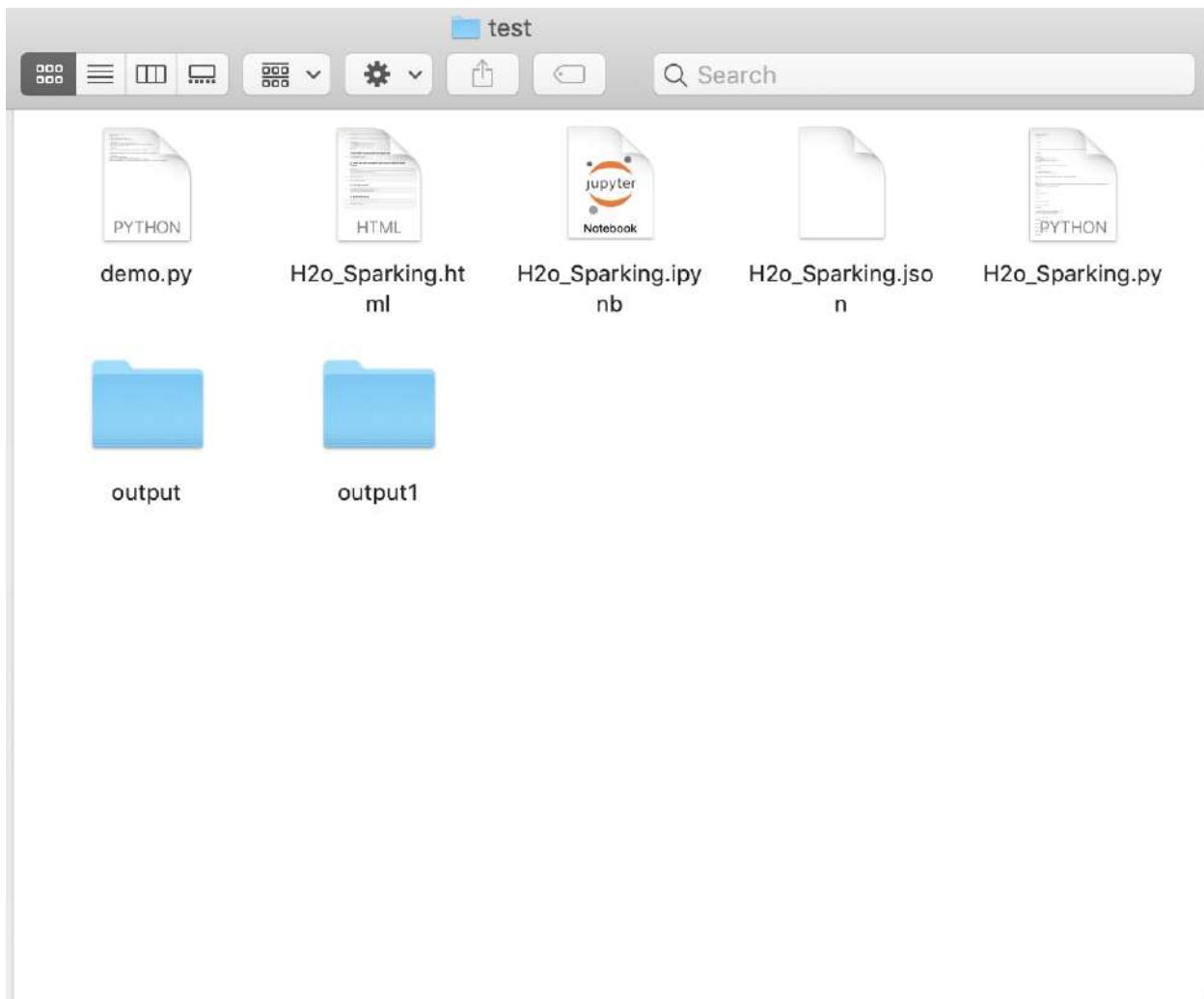
# 시나리오 1
# 현재 디렉토리에서 파일 및 출력물
# 출력 경로, 기본 출력 경로는 현재 디렉토리가 됩니다
ze2nb('H2o_Sparking.json')

# 시나리오 2
output = os.path.abspath(os.path.join(sys.path[0])) + '/output'
ze2nb('H2o_Sparking.json', out_path=output, to_html=True, to_py=True)

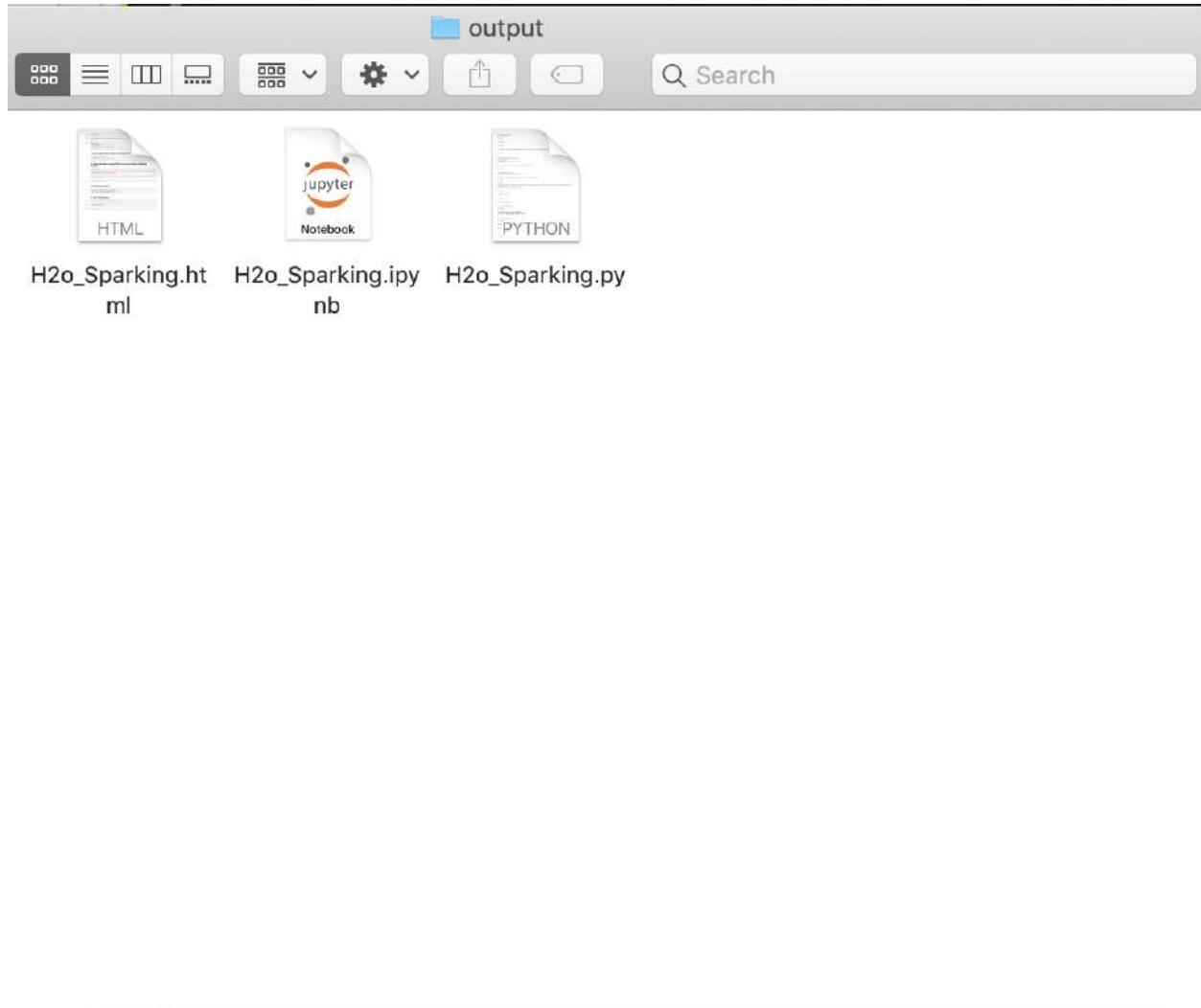
# 시나리오 3
# 로드 및 출력 경로를 지정하여
load_path = '/Users/dt216661/Documents/MyJson/'
output = os.path.abspath(os.path.join(sys.path[0])) + '/output1'
ze2nb('H2o_GBM.json', load_path=load_path, out_path=output, to_html=True, to_
    py=True)
```

23.2.2 변환된 결과

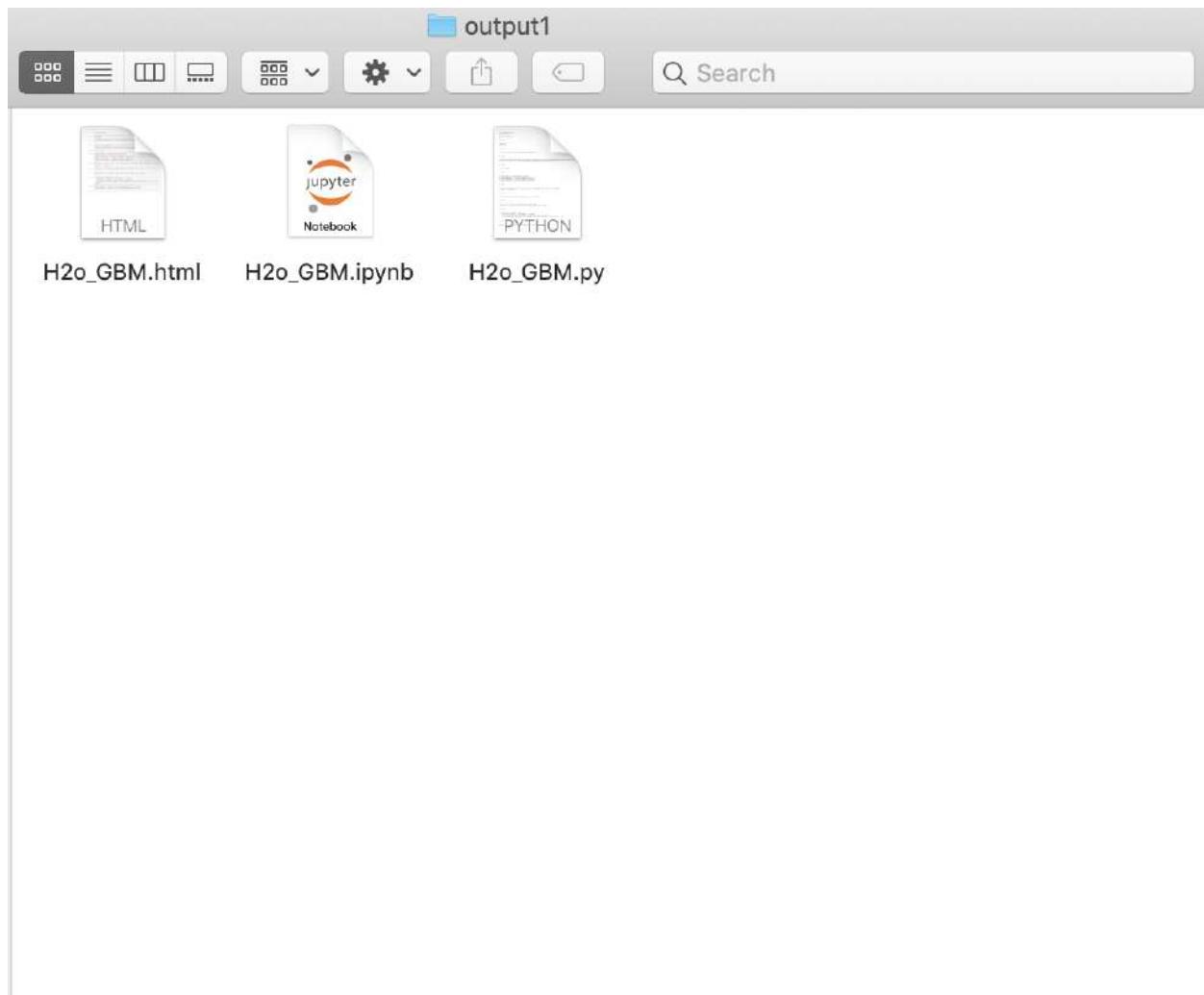
결과:



output 폴더의 결과:



output1 폴더의 결과:



커닝 페이퍼

PDF 버전: PySpark Cheat Sheet 및 pdDataFrame vs rddDataFrame을 다운로드할 수 있습니다.

PySpark 커닝 페이퍼

Wenqiang Feng
E-mail: von198@gmail.com, Web: http://web.utk.edu/~wfeng; https://runawayhorse001.github.io/LearningApacheSpark

Spark 환경 설정

```
from pyspark.sql import SparkSession
spark = SparkSession.builder
    .appName('Python Spark regression example')
    .config("spark.option", "value").getOrCreate()
```

데이터 토막

```
# parallelize() 사용 예제
df = spark.parallelize(parallelize([('1', 'Joe', '70000', '1'),
                                      ('2', 'Henry', '80000', None)])
dfDF[['Id', 'Name', 'Salary', 'DepartmentId']]
# createDataFrame() 사용 예제
df = spark.createDataFrame([(1, 'Joe', '70000', '1'),
                           ('2', 'Henry', '80000', None),
                           ('3', 'Mike', '90000', '2')])
df.show(3)
+---+-----+-----+-----+
| Id|Name |Salary|DepartmentId|
+---+-----+-----+-----+
1	Joe	70000	1
2	Henry	80000	null
3	Mike	90000	2
+---+-----+-----+-----+
```

Data 소스들로 부터

- From CSV
- From JSON
- From HDFS

데이터 감사

스키마 확인

```
df.printSchema()
root
 |-- Id: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Salary: double (nullable = true)
 |-- DepartmentId: double (nullable = true)
```

결측값 확인

```
from pyspark.sql.functions import count
def my_count(df):
    df.agg(*[count(c).alias(c) for c in df.columns]).show()
my_count(df_raw)
+-----+-----+
| count| count|
+-----+-----+
| 3000 | 3000 |
+-----+-----+
```

통계 결과 확인

```
df_raw.describe().show()
+-----+-----+-----+
| summary| count| max | min |
+-----+-----+-----+
count	3000	3000	3000
mean	84.94333333333334	5642000000000000	36.81333333333333
std	22.12020202020202	21.779260202020202	11.779260202020202
min	34.81	34.81	34.81
max	234.41	49.41	114.41
+-----+-----+-----+
```

데이터 조작 (다음 페이지에 자세히)

결측값 수정

| Function | Description |
|--------------|-----------------|
| df.na.fill() | 결측값 대체하기 |
| df.na.drop() | 결측값을 갖는 행을 제거하기 |

데이터 Join

| Description | Function |
|-------------|--|
| #Data join | left.join(right, key, how='*') = left,right,inner,full |

UDF를 통한 데이터 정제

```
from pyspark.sql import functions as F
from pyspark.sql.types import DoubleType
# 사용자 정의 함수
def complexFun(x):
    return results
Fn = F.udf(lambda x: complexFun(x), DoubleType())
df.withColumn('Zcol', Fn(df.col))
```

특정 줄이기

```
df.select(featureNameList)
```

모델 특징과 라벨 데이터 다루기

```
# 범주형 특징과 라벨 데이터 다룬다
from pyspark.ml.feature import VectorIndexer
featureIndexer = VectorIndexer(inputCol="features",
                                outputCol="indexedFeatures",
                                maxCategories=4).fit(data)
featureIndexer.transform(data).show(2, True)
+-----+-----+
| featuresLabel| indexedFeatures |
+-----+-----+
| (2,1,1,1,1,1)| 0 |
| (2,1,1,1,1,1)| 0 |
+-----+-----+
```

범주형 라벨 데이터 다룬다
labelIndexer=StringIndexer(inputCol='label',
 outputCol='indexedLabel').fit(data)
labelIndexer.transform(data).show(2, True)
+-----+-----+
| featuresLabel| indexedLabel |
+-----+-----+
| (2,1,1,1,1,1)| 0 |
+-----+-----+

데이터 훈련 및 검증 셋으로 분할

```
(trainningData, testData) = data.randomSplit([0.6, 0.4])
```

모델 불러오기

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol='indexedFeatures',
                        labelCol='indexedLabel')
```

인덱스 처리된 라벨들을 문제 라벨로 변환하기

```
from pyspark.ml.feature import IndexToString
labelConverter = IndexToString(inputCol="prediction",
                               outputCol="predictedLabel",
                               labelMap=labelIndexer.labels)
```

Pipeline

```
pipeline = Pipeline(stages=[labelIndexer, featureIndexer,
                            lr, labelConverter])
```

모델 훈련 및 예측값 생성

```
model = pipeline.fit(trainningData)
predictions = model.transform(testData)
predictions.select("features", "label", "predictedLabel").show(2)
+-----+-----+-----+
| featuresLabel| predictedLabel |
+-----+-----+
| (2,1,1,1,1,1)| 0 |
| (2,1,1,1,1,1)| 0 |
+-----+-----+
```

모델 평가

```
from pyspark.ml.evaluation import *
evaluator = MulticlassClassificationEvaluator(
    metricName="accuracy",
    predictionCol="prediction", metricName="accuracy")
accu = evaluator.evaluate(predictions)
print("Test Error: %g, AUC: %g" % (1-accu,Summary.areasUnderROC))
Test Error: 0.098565, AUC: 0.868994289577
```

파이썬으로 아파치 스파크 학습하기

pd.DataFrame vs rdd.DataFrame

Wenqiang Feng
E-mail: von198@gmail.com, Web: <http://web.utk.edu/~wfeng1>; <https://runawayhorse001.github.io/LearningApacheSpark>

데이터 프레임 티피

- 데이터 프레임 생성**

```
pd.DataFrame(my_list,columns= col_name)
spark.createDataFrame(my_list, col_name).show()
```
- Dictionary로 부터**

```
pd.DataFrame(d)
spark.createDataFrame(np.array(list(d.values()))).toList(),
list(d.keys())).show()
```

Data 소스로 부터

- Database로 부터**

```
con = psycopg2.connect(host=host, database=d_name,
user=user, password=password)
cur = con.cursor()
sql = """select * from table_name
        """.format(table_name=table_name)
df = pd.read_sql(sql, con)

url='jdbc:postgresql://'+host+'/' +d_name+'?user=' +user
        +'&password=' +pw
p='(driver)'org.postgresql.Driver','password':pw,'user':user
dfspark.read.jdbc(url=url, table=table_name, properties=p)
```
- csv로 부터**

```
df = pd.read_csv('Advertising.csv')
ds = spark.read.csv(path='Advertising.csv',
header=True,inferSchema=True)
```
- json로 부터**

```
df = pd.read_json('data/date.json')
ds = spark.read.json('data/date.json')
```

기초 조작

하나 혹은 그 이상의 변수명 변경

```
import 'pyspark' as p
df.rename(columns=naming).show(4)
new_names = [mapping.get(col,col) for col in df.columns]
df.toDF(*new_names).show(4)
```

하나 혹은 그 이상의 데이터 탑재 변경

```
a = df.select('col1').cast('string').rdd
df = df.withColumn('col1',a)
ds = ds.select(*list(set(ds.columns)-set(d.keys())))
+ (col(c0).alias(c1)).alias(c0) for c in d.items()])
```

랜덤 분할

```
from sklearn.model_selection import train_test_split
a, b = train_test_split(df, test_size=0.5)
```

Unistime 탑재의 날짜

```
df['date']=pd.to_datetime(df['ts'],unit='s').dttz.localize('UTC')
spark.conf.set('spark.sql.session.timeZone', "UTC")
ds.withColumn('date', F.from_unixtime('ts'))
```

신규 변수 생성

```
df['tv_norm'] = df.TV.sum().rdd
ds.withColumn('tv_norm', df.TV).groupby()
.agg(F.sum('TV')).collect()[0][0].show(4)
```

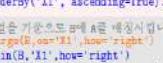
```
df['cond1'] = df.apply(lambda x: 1 if ((x.TV>10)&(x.Radio<60))
else 2 if (x.Radio>10
else 3,alias=1)
ds.withColumn('cond1',F.when((ds.TV>10)&(ds.Radio<60),1)
.when(ds.Sales>10, 2)
.otherwise(3)).show(4)
```

```
df['log_tv'] = np.log(df.TV)
ds.withColumn('log_tv', F.log(ds.TV)).show(4)
df['tv10'] = df.TV.apply(lambda x: x*10)
ds.withColumn('tv10', ds.TV*10).show(4)
```

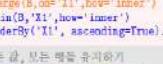
Join으로 데이터 변형

A  B

#X1을 기준으로 A에 B를 병합시킵니다
A.merge(B, on='X1', how='left')
A.join(B, 'X1', how='left')
.orderBy('X1', ascending=True).show()

A  B

#X2를 기준으로 B에 A를 병합시킵니다
B.merge(A, on='X2', how='right')
B.join(A, 'X2', how='right')
.orderBy('X2', ascending=True).show()

A  B

#모든 A는 B를 유지하기
A.merge(B, on='X1', how='inner')
A.join(B, 'X1', how='inner')
.orderBy('X1', ascending=True).show()

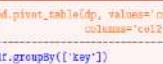
A  B

#모든 A는 B를 유지하기
A.merge(B, on='X1', how='full')
A.join(B, 'X1', how='full')
.orderBy('X1', ascending=True).show()

Data 그룹화

A  B

```
df.groupby(['A']).agg('B':min, 'C':avg).show()
```

A  B

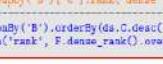
```
df.pivot_table(values='col1', index='key',
columns='col2', aggfunc=np.sum)
```

파티션

A  B

```
w = Window.partitionBy('B').orderBy(ds.C.desc())
ds = ds.withColumn('rank', F.dense_rank().over(w))
```

Windows

A  B

```
df['rank'] = df.groupby('B').rank().orderBy(ds.C.desc(), ascending=False)
```

데이터 요약

열 삭제하기

```
df.drop(name_list, axis=1)
ds.drop(*name_list).show()
```

크로스 테이블

```
pd.crosstab(df.col1, df.col2)
ds.crosstab('col1', 'col2')
```

값 대체

```
df.A.replace(['male', 'female'], [1, 0], inplace=True)
ds.na.replace(['male', 'female'], [1, 0]).show()
```

SQL

```
sql = """
    SELECT * FROM table_name
    """.format(table_name=table_name)
df = pc.read_sql(sql, conn)
# df.registerTempTable('ds')
spark.sql("SELECT * FROM ds").show()
```

379

JDBC 연동

이 장에서는 JDBC 소스를 사용하여 PySpark에서 데이터를 쓰고 읽을 수 있는 방법에 대해 배울 것입니다. 스파크에 대한 아이디어 데이터베이스는 HDFS입니다. 많지 않은 회사가 모든 데이터(PII 데이터)를 Databricks의 HDFS로 이동하지 않으려 합니다. 그렇다면 외부 데이터베이스에 연결하기 위해 JDBC를 사용해야 합니다. JDBC 읽기와 쓰기는 초보자에게는 항상 까다롭고 혼란스럽습니다.

25.1 JDBC 드라이버

성공적으로 연결하려면 지정된 데이터베이스에 해당하는 JDBC 드라이버가 필요합니다. 여기서는 Greenplum 데이터베이스를 예로 들어 올바른 .jar 파일을 얻는 방법과 .jar을 저장할 위치를 보여 드리겠습니다.

25.1.1 .jar 파일 가져오기

Greenplum은 PostgreSQL을 사용하고 있기 때문에 여러분은 ‘PostgreSQL JDBC Driver’로 검색할 수 있습니다. 여러분이 해당 페이지를 확인하게 될 것입니다: <https://jdbc.postgresql.org/download.html>. 그 다음 .jar 파일을 다운로드합니다.

25.1.2 jars 폴더에 .jar 넣기

이제 여러분이 해야 할 일은 .jar 파일을 여러분의 스파크 설치 폴더의 하위 jar 폴더에 넣는 것입니다. 여기에 제 jar 폴더가 있습니다: /opt/spark/jars

```
feng@feng-ThinkPad: /opt/spark/jars
oro-2.0.8.jar
osgi-resource-locator-1.0.1.jar
paranamer-2.3.jar
parquet-column-1.8.1.jar
parquet-common-1.8.1.jar
parquet-encoding-1.8.1.jar
parquet-format-2.3.0-incubating.jar
parquet-hadoop-1.8.1.jar
parquet-hadoop-bundle-1.6.0.jar
parquet-jackson-1.8.1.jar
pmml-model-1.2.15.jar
pmml-schema-1.2.15.jar
postgresql-42.1.4.jre6.jar
protobuf-java-2.5.0.jar
py4j-0.10.4.jar
pyrolite-4.13.jar
RoaringBitmap-0.5.11.jar
scala-compiler-2.11.8.jar
scala-library-2.11.8.jar
scalap-2.11.8.jar
scala-parser-combinators_2.11-1.0.4.jar
scala-reflect-2.11.8.jar
scala-xml_2.11-1.0.2.jar
shapeless_2.11-2.0.0.jar
```

그림. 1: JDBC 연동 jars 풀더

25.2 JDBC read

JDBC lower-upper Bound 코드를 확인하시길 바랍니다.

```
stride = (upper_bound/partitions_number) - (lower_bound/partitions_number)
partition_nr = 0
while (partition_nr < partitions_number)
    generate WHERE clause:
        partition_column IS NULL OR partition_column < stride
        if:
            partition_nr == 0 AND partition_nr < partitions_number
        or generate WHERE clause:
            partition_column >= stride AND partition_column < next_stride
            if:
                partition_nr < 0 AND partition_nr < partitions_number
            or generate WHERE clause
                partition_column >= stride
                if:
                    partition_nr > 0 AND partition_nr == partitions_number
    where next_stride is calculated after computing the left sideo
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
of the WHERE clause by next_stride += stride

(stride = (20/5) - (0/5) = 4
SELECT * FROM my_table WHERE partition_column IS NULL OR partition_column < 4
SELECT * FROM my_table WHERE partition_column >= 4 AND partition_column < 8
SELECT * FROM my_table WHERE partition_column >= 8 AND partition_column < 12
SELECT * FROM my_table WHERE partition_column >= 12 AND partition_column < 16
SELECT * FROM my_table WHERE partition_column >= 16
```

보시다시피, 위 쿼리는 5개의 데이터 파티션을 생성하며, 각 파티션에는 다음의 값이 포함됩니다: (0-3), (4-7), (8-11), (12-15) 및 (16 그리고 그 이상).

DATABRICKS 팁

이번 장에서는 Databricks를 사용할 때 유용한 몇 가지 팁을 공유하겠습니다.

26.1 샘플 표시

pyspark에서는 `show()`를 사용하여 주어진 샘플 크기를 표시할 수 있습니다. databricks 환경에서는 샘플 레코드를 표시하는 `display()` 함수도 있습니다. 일반적으로 빅데이터 테이블/세트의 CPU 시간은 다음과 같습니다.

```
display(df) < df.limit(n).show() < df.show(n)
# n 은 주어진 숫자
```

26.2 자동 파일 다운로드

Databricks는 가장 강력한 빅 데이터 분석 및 기계 학습 플랫폼이지만 완벽하진 않습니다. 파일 관리 시스템은 jupyter Notebook/Lab만큼 좋지 않습니다. 여기서는 dbfs:/FileStore 의 하위 파일을 다운로드하는 한 가지 방법을 제공하겠습니다(이 방법은 dbfs:/FileStore 의 하위 파일에만 작동합니다).

일반적으로 다운로드를 위한 파일 링크는 다음과 같습니다.:

```
f" {cluster_address}/files/{file_path}?o={cluster_no}"
```

여기서 자동 클릭 다운로드 함수를 제공합니다:

```
import os
import IPython
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
from jinja2 import Template
from pathlib import Path
from subprocess import Popen, PIPE

def get_hdfs_files(hdfs_path, relative_path=True):
    """
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

지정된 HDFS 경로에 대한 파일 이름과 파일 경로 또는 상대 경로를 가져옵니다.
노트: os.listdir 는 ``community.cloud.databricks``에서 작동하지 않습니다.

```
:param hdfs_path: 입력 HDFS 경로
:param relative_path: 반환 전체 경로의 플래그 또는
    ``dbfs:/FileStore`` 관련 경로
    (파일 다운로드를 위해 관련 경로가 필요합니다)

:return file_names: 지정된 경로의 하위 파일 이름
:return relative_p: 파일 경로, ``relative_path=False`` 면 전체 경로 아니면
    ``dbfs:/FileStore`` 와 관련된 경로
"""

# 파일 정보를 가져옵니다
xx = dbutils.fs.ls(hdfs_path)

# 폴더 이름과 hdfs 경로를 가져옵니다
file_names = [list(xx[i])[1] for i in range(len(xx))]
hdfs_paths = [list(xx[i])[0] for i in range(len(xx))]

if relative_path:
    try:
        relative_p = [os.path.relpath(hdfs_path, 'dbfs:/FileStore') for hdfs_
    ↪path in hdfs_paths]
    except:
        print("Only support the files under 'dbfs:/FileStore/'")
else:
    relative_p = hdfs_paths

return file_names, relative_p

def azdb_files_download(files_path, cluster_address="https://community.cloud.
    ↪databricks.com",
                        cluster_no='4622560542654492'):
"""

파일 다운로드 링크를 나열합니다

:param files_path: 지정된 파일 혹은 폴더 경로
:param cluster_address: 여러분의 databricks 클러스터 주소, 즉 ``?o`` 전 링크
:param cluster_no: 여러분의 databricks 클러스터 개수, 즉 ``?o=`` 다음 숫자
"""

if not os.path.isfile(files_path): # os.path.isdir(files_path):

    file_names, files_path = get_hdfs_files(files_path)

    if not isinstance(files_path, list):
        files_path = [files_path]

    urls = [f"{cluster_address}/files/{file_path}?o={cluster_no}" for file_path_
    ↪in files_path]
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
temp = """
    <h2>AZDB files download</h2>
    { % for i in range(len(urls)) %

        <a href="{{urls[i]}}" target='_blank'> Click me to download  {
        ↪{files_path[i].split('/')[-1]} }</a> <br>
    { % endfor %
"""
"""

html = Template(temp).render(files_path=files_path, urls=urls, len=len)

# dbutils 모듈을 가져옵니다
dbutils = IPython.get_ipython().user_ns["dbutils"]

dbutils.displayHTML(html)
```

노트: 상용 버전의 databricks에서는 여러분이 cluster_no.를 가져올 수 있습니다.

```
spark.conf.get("spark.databricks.clusterUsageTags.instanceWorkerEnvId")
```

하지만 커뮤니티 버전에서는 이를 사용할 수 없습니다.

위 코드를 사용하여 여러분은 dbfs:/FileStore관련 파일을 다운로드 할 수 있습니다.

dbfs:/FileStore/data의 하위 파일:

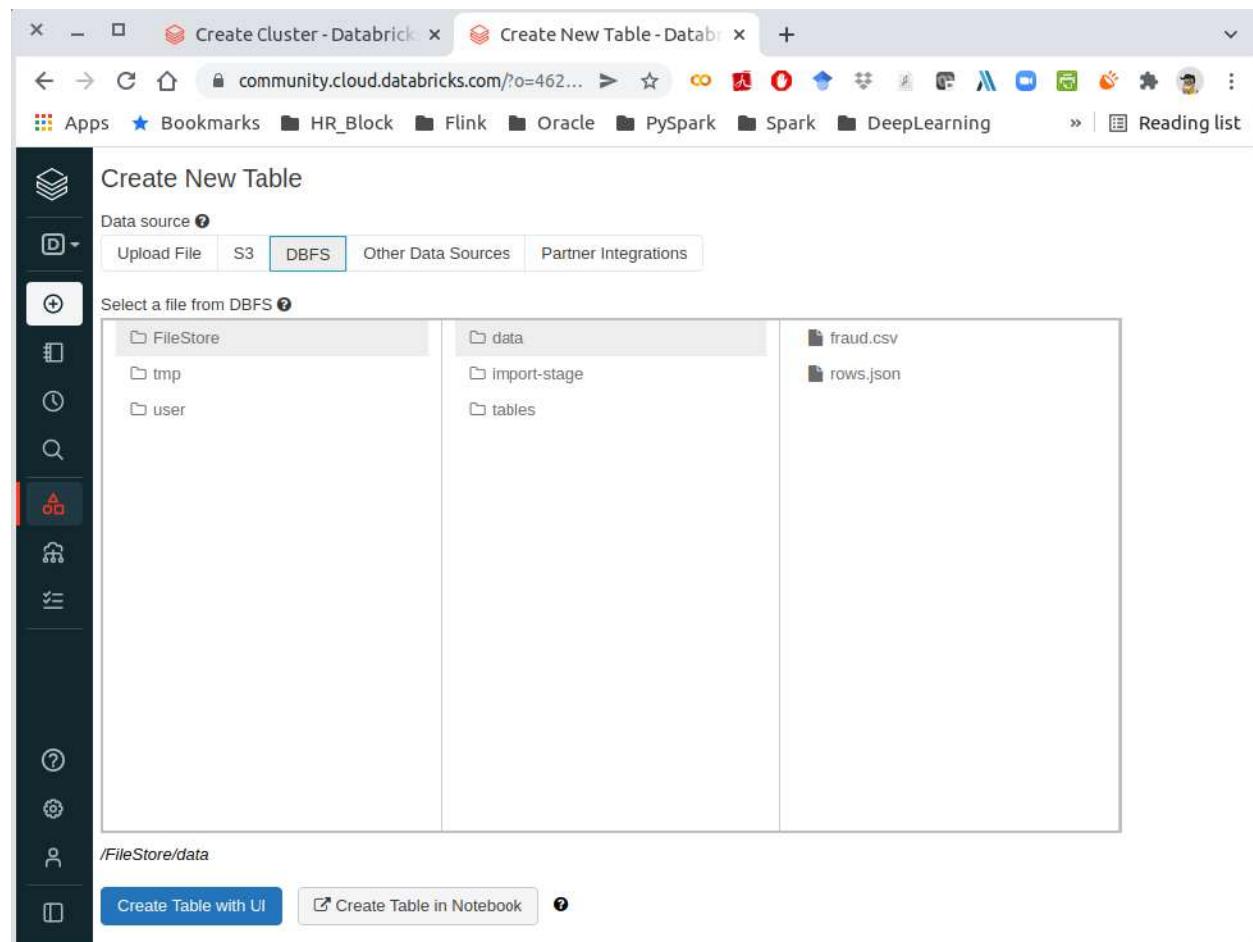


그림. 1: dbfs:/FileStore/data의 하위 파일

download demos를 클릭합니다:

The screenshot shows the Databricks interface with two command cells in a notebook titled "auto_download_demos".

Cmd 127:

```
1 hdfs_path = 'dbfs:/FileStore/data/fraud.csv'
2 azdb_files_download(hdfs_path)
```

AZDB files download

[Click me to download fraud.csv](#)

Command took 0.10 seconds -- by wfeng1@utk.edu at 11/19/2021, 1:27:54 PM on test

Cmd 128:

```
1 hdfs_path = 'dbfs:/FileStore/data'
2 azdb_files_download(hdfs_path)
```

AZDB files download

[Click me to download fraud.csv](#)

[Click me to download rows.json](#)

Command took 0.14 seconds -- by wfeng1@utk.edu at 11/19/2021, 1:28:09 PM on test

그림. 2: databricks에서 파일 다운로드

26.3 AWS S3에서 작업하기

많은 회사들이 민감한 데이터를 AWS S3에 저장하기로 선택했습니다. 따라서 Databricks에서 python으로 데이터를 다뤄야 할 수도 있지만 python은 Databricks에서 작동하기에 많은 문제가 있을 것입니다(Databricks에서 환경이 올바르게 설정되어 있으면 PySpark는 문제가 없을 것입니다): 예를 들어 os.system 커맨드와 같은 기능은 더 이상 사용하지 않습니다. 다른 유형의 파일을 업로드하거나 다운로드하는 통합된 방법이 없습니다. 여기서는 Databricks의 python을 사용하여 AWS S3에서 작업하는 방법을 소개하겠습니다:

26.3.1 자격 증명 (Credentials)

STS(Security Token Service)를 사용하여 AWS S3에 액세스하기 위한 자격 증명을 생성합니다. STS를 사용하면 IAM(Identity and Access Management) 사용자 또는 여러분이 인증한 사용자(federated 사용자)에 대한 임시 제한 privilege 자격 증명을 요청할 수 있습니다. 자세한 내용은 다음에서 확인할 수 있습니다: <https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sts.html>

```
import boto3

response = boto3.client('sts')\
    .assume_role(RoleArn='arn:aws:iam::123456789012:role/demo',
                 RoleSessionName='your_role_session_name')
credentials = response['Credentials']
```

26.3.2 S3에 파일 업로드

여기서의 주요 아이디어는 메모리나 임시 파일에 파일을 저장한 다음 `put_object`를 사용하여 S3에 파일을 넣는 것입니다. 해당 포맷된 파일을 메모리에 저장하는 것은 조금 까다롭습니다. 예제에서 몇 가지 일반적인 유형을 나열하겠습니다.

1. `s3_file_upload` 함수

```
def s3_file_upload(data, path):

    file_type = path.split('/')[-1].split('.')[1].lower()
    try:
        content_type = s3_content_type(file_type)
    except:
        print('Do not support the current input type!!!!')

    s3_path = path.replace('s3://', '').replace('s3a://', '')
    bucket = s3_path.split('/')[0]
    key = s3_path.split('/', 1)[1]

    try:
        s3_resource.Bucket(bucket)\n            .put_object(Key=key,
                    Body=data,
                    ContentType=content_type,
                    ACL='public-read')
        print(f'{key.split('/')[-1]} has been successfully saved in s3!')
    except Exception as err:
        print(err)
```

2. `s3_content_type` 함수

```
def s3_content_type(file_type):
    mapping = {png:image/png}

    return mapping[file_type]
```

전체 맵핑 목록은 다음과 같습니다:

```
3dm:x-world/x-3dmf
3dmf:x-world/x-3dmf
a:application/octet-stream
aab:application/x-authorware-bin
aam:application/x-authorware-map
aas:application/x-authorware-seg
abc:text/vnd.abc
acgi:text/html
afl:video/animaflex
ai:application/postscript
aif:audio/aiff
#aif:audio/x-aiff
aifc:audio/aiff
#aifc:audio/x-aiff
aiff:audio/aiff
#aiff:audio/x-aiff
aim:application/x-aim
aip:text/x-audiosoft-intra
ani:application/x-navi-animation
aos:application/x-nokia-9000-communicator-add-on-software
aps:application/mime
arc:application/octet-stream
arj:application/arj
art:image/x-jg
ASF:video/x-ms-asf
asm:text/x-asm
asp:text/asp
ass:application/x-mplayer2
#ass:video/x-ms-asf
#ass:video/x-ms-asf-plugin
au:audio/basic
#au:audio/x-au
#avi:video/avi
#avi:video/msvideo
avi:video/x-msvideo
avs:video/avs-video
bcpio:application/x-bcpio
#bin:application/mac-binary
#bin:application/macbinary
#bin:application/octet-stream
bin:application/x-binary
#bin:application/x-macbinary
bm:image/bmp
bmp:image/bmp
boo:application/book
book:application/book
boz:application/x-bzip2
bsh:application/x-bsh
bz:application/x-bzip
bz2:application/x-bzip2
c:text/plain
```

(다음 페이지에서 계속)

(이전 페이지에서 계속)

```
c++:text/plain  
cat:application/vnd.ms-pki.seccat  
cc:text/plain  
ccad:application/clariscad  
cco:application/x-cocoa  
cdf:application/cdf  
cer:application/pkix-cert  
cha:application/x-chat  
chat:application/x-chat  
class:application/java  
com:application/octet-stream  
conf:text/plain  
cpio:application/x-cpio  
cpp:text/x-c  
cpt:application/mac-compactpro  
crl:application/pkcs-crl  
crt:application/pkix-cert  
csh:application/x-csh  
css:text/css  
cxx:text/plain  
dcr:application/x-director  
deepv:application/x-deepv  
def:text/plain  
der:application/x-x509-ca-cert  
dif:video/x-dv  
dir:application/x-director  
dl:video/dl  
doc:application/msword  
dot:application/msword  
dp:application/commonground  
drw:application/drafting  
dump:application/octet-stream  
dv:video/x-dv  
dvi:application/x-dvi  
dwf:model/vnd.dwf  
dwg:application/acad  
dxr:application/x-director  
el:text/x-script.elisp  
elc:application/x-bytecode.elisp  
env:application/x-envoy  
eps:application/postscript  
es:application/x-esrehber  
etx:text/x-setext  
evy:application/envoy  
exe:application/octet-stream  
f:text/plain  
f77:text/x-fortran  
f90:text/plain  
fdf:application/vnd.fdf  
fif:application/fractals  
fli:video/fli
```

(다음 페이지에서 계속)

(○] 전 페이지에서 계속)

```
flo:image/florian
flx:text/vnd.fmi.flexstor
fmf:video/x-atomic3d-feature
for:text/plain
fpx:image/vnd.fpx
frl:application/freeloader
funk:audio/make
g:text/plain
g3:image/g3fax
gif:image/gif
gl:video/gl
gsd:audio/x-gsm
gsm:audio/x-gsm
gsp:application/x-gsp
gss:application/x-gss
gtar:application/x-gtar
gz:application/x-gzip
gzip:application/x-gzip
h:text/plain
hdf:application/x-hdf
help:application/x-helpfile
hgl:application/vnd.hp-hpcl
hh:text/plain
hlp:application/hlp
hpg:application/vnd.hp-hpgl
hpgl:application/vnd.hp-hpgl
hqx:application/binhex
hta:application/hta
htc:text/x-component
htm:text/html
html:text/html
htmls:text/html
htt:text/webviewhtml
htx:text/html
ice:x-conference/x-cooltalk
ico:image/x-icon
idc:text/plain
ief:image/ief
iefs:image/ief
iges:application/iges
igs:application/iges
ima:application/x-ima
imap:application/x-httpd-imap
inf:application/inf
ins:application/x-internett-signup
ip:application/x-ip2
isu:video/x-isvideo
it:audio/it
iv:application/x-inventor
ivr:i-world/i-vrml
ivy:application/x-livescreen
jam:audio/x-jam
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
java:text/plain
jcm:application/x-java-commerce
jfif:image/jpeg
jpeg:image/jpeg
jpg:image/jpeg
jps:image/x-jps
js:application/x-javascript
jut:image/jutvision
kar:audio/midi
ksh:text/x-script.ksh
la:audio/nspaudio
lam:audio/x-liveaudio
latex:application/x-latex
lha:application/octet-stream
lhx:application/octet-stream
list:text/plain
lma:audio/nspaudio
log:text/plain
lst:text/plain
lsx:text/x-la-asf
ltx:application/x-latex
lzh:application/octet-stream
lzx:application/octet-stream
m:text/plain
m1v:video/mpeg
m2a:audio/mpeg
m2v:video/mpeg
m3u:audio/x-mpeqlurl
m4v:video/x-m4v
man:application/x-troff-man
mht:message/rfc822
mhtml:message/rfc822
midi:audio/midi
mif:application/x-frame
mjf:audio/x-vnd.audioexplosion.mjuicemediafile
mjpg:video/x-motion-jpeg
mod:audio/mod
mov:video/quicktime
movie:video/x-sgi-movie
mp2:audio/mpeg
mp3:audio/mpeg
#mpa:audio/mpeg
mpa:video/mpeg
mpc:application/x-project
mpeg:video/mpeg
mpg:video/mpeg
mpga:audio/mpeg
ogg:video/ogg
ogv:video/ogg
p:text/x-pascal
p10:application/pkcs10
#p12:application/pkcs-12
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

p12:application/x-pkcs12
p7a:application/x-pkcs7-signature
p7c:application/x-pkcs7-mime
p7m:application/pkcs7-mime
p7r:application/x-pkcs7-certreqresp
p7s:application/pkcs7-signature
part:application/pro_eng
pas:text/pascal
pbm:image/x-portable-bitmap
pcl:application/x-pcl
pct:image/x-pict
pcx:image/x-pcx
pdb:chemical/x-pdb
pdf:application/pdf
pfunk:audio/make
pgm:image/x-portable-graymap
pic:image/pict
pict:image/pict
pkg:application/x-newton-compatible-pkg
pko:application/vnd.ms-pki.pko
pl:text/plain
plx:application/x-pixclscript
pm:image/x-xpixmap
pm4:application/x-pagemaker
pm5:application/x-pagemaker
png:image/png
pnm:image/x-portable-anymap
pot:application/mspowerpoint
ppa:application/vnd.ms-powerpoint
ppm:image/x-portable-pixmap
pps:application/mspowerpoint
ppt:application/mspowerpoint
#ppt:application/powerpoint
#ppt:application/vnd.ms-powerpoint
#ppt:application/x-mspowerpoint
ppz:application/mspowerpoint
pre:application/x-freelance
prt:application/pro_eng
ps:application/postscript
psd:application/octet-stream
pvu:paleovu/x-pv
pwz:application/vnd.ms-powerpoint
py:text/x-script.phyton
pyc:applicaiton/x-bytecode.python
qcp:audio/vnd.qcelp
qd3:x-world/x-3dmf
#qd3d:x-world/x-3dmf
qif:image/x-quicktime
qt:video/quicktime
qtc:video/x-qtc
qti:image/x-quicktime
qtif:image/x-quicktime

```

(다음 페이지에서 계속)

(이전 페이지에서 계속)

```
ra:audio/x-pn-realaudio
#ra:audio/x-pn-realaudio-plugin
#ra:audio/x-realaudio
ram:audio/x-pn-realaudio
ras:application/x-cmu-raster
#ras:image/cmu-raster
#ras:image/x-cmu-raster
#rast:image/cmu-raster
#rexx:text/x-script.rexx
#rf:image/vnd.rn-realflash
rgb:image/x-rgb
rm:application/vnd.rn-realmedia
#rm:audio/x-pn-realaudio
rmi:audio/mid
rmm:audio/x-pn-realaudio
rmp:audio/x-pn-realaudio
#rmp:audio/x-pn-realaudio-plugin
rng:application/ringing-tones
#rng:application/vnd.nokia.ringing-tone
rnx:application/vnd.rn-realplayer
roff:application/x-troff
rp:image/vnd.rn-realpix
rpm:audio/x-pn-realaudio-plugin
rt:text/richtext
#rt:text/vnd.rn-realtext
rtf:application/rtf
#rtf:application/x-rtf
#rtf:text/richtext
#rtx:application/rtf
#rtx:text/richtext
rv:video/vnd.rn-realvideo
s:text/x-asn
s3m:audio/s3m
#saveme:application/octet-stream
sbk:application/x-tbook
scm:application/x-lotusscreencam
#scm:text/x-script.guile
#scm:text/x-script.scheme
#scm:video/x-scm
sdml:text/plain
sdp:application/sdp
#sdp:application/x-sdp
sdr:application/sounder
sea:application/sea
#sea:application/x-sea
set:application/set
sgm:text/sgml
#sgm:text/x-sgml
sgml:text/sgml
#sgml:text/x-sgml
sh:application/x-bsh
#sh:application/x-sh
```

(다음 페이지에 계속)

(○] 전 페이지에서 계속)

```
#sh:application/x-shar
#sh:text/x-script.sh
shar:application/x-bsh
#shar:application/x-shar
shtml:text/html
#shtml:text/x-server-parsed-html
sid:audio/x-psid
#sit:application/x-sit
sit:application/x-stuffit
skd:application/x-koan
skm:application/x-koan
skp:application/x-koan
skt:application/x-koan
sl:application/x-seelogo
smi:application/smil
smil:application/smil
#snd:audio/basic
snd:audio/x-adpcm
sol:application/solids
#spc:application/x-pkcs7-certificates
spc:text/x-speech
spl:application/futuresplash
spr:application/x-sprite
sprite:application/x-sprite
src:application/x-wais-source
ssi:text/x-server-parsed-html
ssm:application/streamingmedia
sst:application/vnd.ms-pki.certstore
step:application/step
stl:application/sla
#stl:application/vnd.ms-pki.stl
#stl:application/x-navistyle
stp:application/step
sv4cpio:application/x-sv4cpio
sv4crc:application/x-sv4crc
svf:image/vnd.dwg
#svf:image/x-dwg
svr:application/x-world
#svr:x-world/x-svr
swf:application/x-shockwave-flash
t:application/x-troff
talk:text/x-speech
tar:application/x-tar
tbk:application/toolbook
#tbk:application/x-tbook
tcl:application/x-tcl
#tcl:text/x-script.tcl
tcsh:text/x-script.tcsh
tex:application/x-tex
texi:application/x-texinfo
texinfo:application/x-texinfo
#text:application/plain
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
text:text/plain
#tgz:application/gnutar
tgz:application/x-compressed
tif:image/tiff
#tif:image/x-tiff
tiff:image/tiff
#tiff:image/x-tiff
tr:application/x-troff
tsi:audio/tsp-audio
tsp:application/dsptype
#tsp:audio/tsplayer
tsv:text/tab-separated-values
turbot:image/florian
txt:text/plain
uil:text/x-uil
uni:text/uri-list
unis:text/uri-list
unv:application/i-deas
uri:text/uri-list
uris:text/uri-list
ustar:application/x-ustar
#ustar:multipart/x-ustar
uu:application/octet-stream
#uu:text/x-uuencode
uue:text/x-uuencode
vcd:application/x-cdlink
vcs:text/x-vcalendar
vda:application/vda
vdo:video/vdo
vew:application/groupwise
viv:video/vivo
#viv:video/vnd.vivo
vivo:video/vivo
#vivo:video/vnd.vivo
vmd:application/vocaltec-media-desc
vmf:application/vocaltec-media-file
voc:audio/voc
#voc:audio/x-voc
vos:video/vosaic
vox:audio/voxware
vqe:audio/x-twinvq-plugin
vqf:audio/x-twinvq
vql:audio/x-twinvq-plugin
vrml:application/x-vrml
#vrml:model/vrml
#vrml:x-world/x-vrml
vrt:x-world/x-vrt
vsd:application/x-visio
vst:application/x-visio
vsw:application/x-visio
w60:application/wordperfect6.0
w61:application/wordperfect6.1
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
w6w:application/msword
wav:audio/wav
#wav:audio/x-wav
wb1:application/x-qpro
wbmp:image/vnd.wap.wbmp
web:application/vnd.xara
wiz:application/msword
wk1:application/x-123
wmf:windows/metafile
wml:text/vnd.wap.wml
wmlc:application/vnd.wap.wmlc
wmls:text/vnd.wap.wmlscript
wmlsc:application/vnd.wap.wmlscriptc
word:application/msword
wp:application/wordperfect
wp5:application/wordperfect
#wp5:application/wordperfect6.0
wp6:application/wordperfect
wpd:application/wordperfect
#wpd:application/x-wpwin
wql:application/x-lotus
wri:application/mswrite
#wri:application/x-wri
#wrl:application/x-world
wrl:model/vrml
#wrl:x-world/x-vrml
#wrz:model/vrml
#wrz:x-world/x-vrml
#wsc:text/scriptlet
wsrcc:application/x-wais-source
wtk:application/x-wintalk
#xbm:image/x-xbitmap
#xbm:image/x-xbm
xbm:image/xbm
xdr:video/x-amt-demorun
xgz:xgl/drawing
xif:image/vnd.xiff
xl:application/excel
xla:application/excel
#xla:application/x-excel
#xla:application/x-msexcel
#xlb:application/excel
#xlb:application/vnd.ms-excel
xlb:application/x-excel
#xlc:application/excel
#xlc:application/vnd.ms-excel
#xlc:application/x-excel
xld:application/excel
#xld:application/x-excel
#xlk:application/excel
xlk:application/x-excel
#xll:application/excel
```

(다음 페이지에 계속)

(으)전 페이지에서 계속)

```
#xll:application/vnd.ms-excel  
xll:application/x-excel  
#xlm:application/excel  
#xlm:application/vnd.ms-excel  
xlm:application/x-excel  
#xls:application/excel  
#xls:application/vnd.ms-excel  
xls:application/x-msexcel  
#xlt:application/excel  
xlt:application/x-excel  
#xlv:application/excel  
xlv:application/x-excel  
#xlw:application/excel  
#xlw:application/vnd.ms-excel  
#xlw:application/x-excel  
xlw:application/x-msexcel  
xm:audio/xm  
#xml:application/xml  
xml:text/xml  
xmz:xgl/movie  
xpix:application/x-vnd.ls-xpix  
#xpm:image/x-xpixmap  
xpm:image/xpm  
x-png:image/png  
xsr:video/x-amt-showrun  
#xwd:image/x-xwd  
xwd:image/x-xwindowdump  
xyz:chemical/x-pdb  
#z:application/x-compress  
z:application/x-compressed  
#zip:application/x-compressed  
#zip:application/x-zip-compressed  
zip:application/zip  
#zip:multipart/x-zip  
zoo:application/octet-stream  
zsh:text/x-script.zsh
```

3. 예제

a. .csv 파일

메모리에 csv 파일을 저장합니다:

```
csv_io = io.StringIO()  
df.to_csv(csv_io, sep='\t', header=True, index=False)  
csv_io.seek(0)  
# csv데이터는 부호화가 필요합니다  
csv_data = io.BytesIO(csv_io.getvalue().encode())
```

노트:

`tempfile`을 사용하는 대안적 방법입니다:

```
with tempfile.TemporaryFile(mode='r+') as fp:
    df.to_csv(fp, sep='\t', header=True, index=False)
    fp.seek(0)
#
s3_file_upload(csv_data, file_path)
```

파일을 업로드합니다

```
>>> file_path = 'my_bucket/~/~/test/test.csv'
>>> s3_file_upload(csv_data, file_path)
test.csv has been successfully saved in S3!
```

b. .json 파일

```
>>> json_object = """ your json content"""
>>> json_data = json.dumps(json_object)
>>> file_path = 'my_bucket/~/~/test/test.json'
>>> s3_file_upload(json_data, file_path)
test.json has been successfully saved in S3!
```

노트:

`tempfile`을 사용하는 대안적 방법입니다:

```
with tempfile.TemporaryFile() as fp:
    joblib.dump(json_data, fp)
    fp.seek(0)
#
s3_file_upload(json_data, file_path)
```

c. .png, .jpeg 혹은 .pdf

메모리에 이미지를 저장합니다:

```
flights = sns.load_dataset("flights")
may_flights = flights.query("month == 'May'")
fig = plt.figure(figsize=(20,8))
sns.lineplot(data=may_flights, x="year", y="passengers")

img_data = io.BytesIO()
plt.savefig(img_data, format='png')
img_data.seek(0)
```

S3에 인메모리 이미지를 저장합니다:

```
path = 'my_bucket/my_key'
s3_file_save(img_data, path)
```

노트: 위 방법은 .jpeg 와 .pdf 형식에도 적용됩니다.

26.3.3 S3에서 파일 다운로드

주요 아이디어는 파일을 다운로드하고 임시 파일로 /temp에 저장하기 위해 S3 리소스 함수를 사용하고, 해당 형식 함수를 사용하여 파일을 읽는 것입니다.

1. s3_file_download 함수

```
def s3_file_download(path):  
  
    # 주어진 경로에서 버킷 및 키를 추출합니다  
    s3_path = path.replace('s3://', '').replace('s3a://', '')  
    bucket = s3_path.split('/')[0]  
    key = s3_path.split('/', 1)[1]  
  
    # 파일을 다운로드하여 임시 파일로 저장합니다  
    file_name = os.path.join('/tmp', path.split('/')[-1])  
    s3_resource.Bucket(bucket).download_file(key, file_name)  
  
    # 저장된 파일 경로를 반환합니다  
    return file_name
```

2. 예제

```
>>> file_path = 'my_bucket/***/***/test/test.json'  
>>> file_name = s3_file_download(file_path)  
'/temp/test.json'  
>>> joblib.load(filename)
```

26.3.4 S3에서 파일 관리

저는 S3에서 파일 관리를 돋기 위해 주로 s3_fs 를 사용합니다.

1. s3_fs 함수

s3_fs는 주로 s3fs 패키지를 기반으로 합니다. 최상위 클래스 s3fs 는 연결 정보를 보유하고 cp, mv, ls, walk, du, glob 등과 같은 일반적인 파일 시스템 스타일 작업을 허용합니다. 더 자세한 내용은 다음과 같습니다: <https://s3fs.readthedocs.io/en/latest/index.html>

```
import s3fs  
  
s3_fs = s3fs.S3FileSystem(anon=False,  
                           key=credentials['AccessKeyId'],  
                           secret=credentials['SecretAccessKey'],  
                           token=credentials['SessionToken'])
```

2. 예제

간단한 파일 찾기 및 읽기:

```
>>> s3_fs.ls('my-bucket')
['demo-file.csv']
>>> with fs.open('my-bucket/demo-file.csv', 'rb') as f:
...     print(f.read())
b'UserId\tdate\nuser_id1\t2019-05-02\nuser_id2\t2019-12-02\n'
```


PYSPARK API

API들은 PySpark 패키지에서 자동으로 생성되므로 모든 CopyRights는 Spark에 속합니다.

27.1 Stat API

```
class pyspark.ml.stat.ChiSquareTest
```

노트: 실험적

라벨에 대한 모든 특징에 대해 피어슨(Pearson)의 독립성 검정을 수행합니다. 각 특징에 대해 (특징, 라벨) 쌍은 카이 제곱 통계량이 계산되는 분할 행렬로 변환됩니다. 모든 라벨 및 특징 값은 범주형이어야 합니다.

귀무 가설은 결과의 발생이 통계적으로 독립적이라는 것입니다.

버전 2.2.0의 새 기능입니다.

static test (*dataset*, *featuresCol*, *labelCol*)

데이터 셋을 사용하여 Pearson의 독립성 검정을 수행합니다.

파라미터

- **dataset** – 범주형 라벨 및 범주형 특징의 데이터 프레임. 실제 값을 갖는 특징은 각 구별되는 값에 대해 범주형으로 취급됩니다.
- **featuresCol** – 데이터 셋의 특징 열 이름, 벡터 타입 *Vector* (*VectorUDT*)
- **labelCol** – 데이터 셋의 라벨 열의 이름으로, 숫자 타입

반환 결과 라벨에 대한 모든 특징에 대한 테스트 결과를 포함하는 데이터 프레임. 이 데이터 프레임에는 다음 필드가 있는 단일 행이 포함됩니다: - *pValues* : *Vector* - *degreesOfFreedom* : *Array[Int]* - *statistics* : *Vector* 각 필드에는 특징 당 하나의 값이 있습니다.

```
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.stat import ChiSquareTest
>>> dataset = [[0, Vectors.dense([0, 0, 1])],
...             [0, Vectors.dense([1, 0, 1])],
...             [1, Vectors.dense([2, 1, 1])],
...             [1, Vectors.dense([3, 1, 1])]]
>>> dataset = spark.createDataFrame(dataset, ["label", "features"])
>>> chiSqResult = ChiSquareTest.test(dataset, 'features', 'label')
>>> chiSqResult.select("degreesOfFreedom").collect()[0]
Row(degreesOfFreedom=3, 1, 0)
```

버전 2.2.0의 새 기능입니다.

class pyspark.ml.stat.Correlation

노트: 실험적

지정한 메서드를 사용하여 벡터의 입력 데이터 셋에 대한 상관 행렬을 계산합니다. 현재 지원되는 메서드는 다음과 같습니다: *pearson* (기본값), *spearman*.

노트: 순위 상관 관계인 Spearman의 경우, 각 열에 대해 RDD[Double]을 생성하고 정렬이 필요합니다. 순위를 검색한 다음, 열을 RDD[Vector]에 다시 결합하려면 상당히 비용이 많이 듭니다. 공통 계통을 다시 계산하지 않도록 하기 위해 *method='spearman'*으로 corr을 호출하기 전에 입력 데이터 셋을 캐시합니다.

버전 2.2.0의 새 기능입니다.

static corr (dataset, column, method='pearson')

데이터 셋을 사용하여 지정된 방법으로 상관 행렬을 계산합니다.

파라미터

- **dataset** – 데이터 셋 또는 데이터 프레임
- **column** – 상관 계수를 계산해야 하는 벡터 열의 이름입니다. 데이터 셋의 열이어야 하며 벡터 개체가 포함되어 있어야 합니다.
- **method** – 상관 관계 계산을 위해 사용할 방법을 지정하는 문자열입니다. 지원되는 항목: *pearson* (기본값), *spearman*.

반환 벡터 열의 상관 행렬이 들어 있는 데이터 프레임을 반환합니다..

이 데이터 프레임에는 '\$METHOD-NAME(\$COLUMN)' 이름의 단일 행과 단일 열이 포함되어 있습니다.

```
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.stat import Correlation
>>> dataset = [[Vectors.dense([1, 0, 0, -2])],
...             [Vectors.dense([4, 5, 0, 3])],
```

(다음 페이지에 계속)

(continued from previous page)

```

...
[Vectors.dense([6, 7, 0, 8])],
[... [Vectors.dense([9, 0, 0, 1])]]
>>> dataset = spark.createDataFrame(dataset, ['features'])
>>> pearsonCorr = Correlation.corr(dataset, 'features', 'pearson').
collect()[0][0]
>>> print(str(pearsonCorr).replace('nan', 'NaN'))
DenseMatrix([[ 1.          ,  0.0556...,      NaN,  0.4004...],
             [ 0.0556...,  1.          ,      NaN,  0.9135...],
             [      NaN,      NaN,  1.          ,      NaN],
             [ 0.4004...,  0.9135...,      NaN,  1.        ]])
>>> spearmanCorr = Correlation.corr(dataset, 'features', method=
'spearman').collect()[0][0]
>>> print(str(spearmanCorr).replace('nan', 'NaN'))
DenseMatrix([[ 1.          ,  0.1054...,      NaN,  0.4        ],
             [ 0.1054...,  1.          ,      NaN,  0.9486...],
             [      NaN,      NaN,  1.          ,      NaN],
             [ 0.4        ,  0.9486...,      NaN,  1.        ]])

```

버전 2.2.0의 새 기능입니다.

```
class pyspark.ml.stat.KolmogorovSmirnovTest
```

노트: 실험적

연속형 분포에서 표본 추출된 데이터에 대해 양측 Kolmogorov Smirnov (KS) 검정을 수행합니다.

표본 데이터의 경험적 누적 분포와 이론적 분포 사이의 가장 큰 차이를 비교함으로써 표본 데이터가 해당 이론적 분포에서 나온다는 귀무 가설에 대한 검정을 제공할 수 있습니다.

버전 2.4.0의 새 기능입니다.

```
static test (dataset, sampleCol, distName, *params)
```

확률 분포의 동일성을 위해 일원 표본, 양측 Kolmogorov-Smirnov 검정을 실시합니다. 현재 모수로 평균 및 표준 편차를 사용하여 정규 분포를 지원합니다.

파라미터

- **dataset** – 테스트할 데이터 샘플이 들어 있는 데이터 셋 또는 데이터 프레임
- **sampleCol** – 수치형 데이터 셋에서 샘플 열 이름입니다
- **distName** – 이론적 분포의 *string* 이름으로, 현재 "norm"만 지원합니다
- **params** – 이론적 분포에 사용할 모수를 지정하는 *Double* 값의 목록입니다. "norm" 분포의 경우 파라미터에는 평균과 분산이 포함됩니다.

반환 입력 샘플링된 데이터에 대한 Kolmogorov-Smirnov 테스트 결과를 포함하는 데이터 프레임. 이 데이터 프레임에는 다음 필드를 포함하는 단일 행이 포함됩니다:

- *pValue : Double - statistic : Double*

```
>>> from pyspark.ml.stat import KolmogorovSmirnovTest
>>> dataset = [[-1.0], [0.0], [1.0]]
>>> dataset = spark.createDataFrame(dataset, ['sample'])
>>> ksResult = KolmogorovSmirnovTest.test(dataset, 'sample', 'norm', -0.0, 1.0).first()
>>> round(ksResult.pValue, 3)
1.0
>>> round(ksResult.statistic, 3)
0.175
>>> dataset = [[2.0], [3.0], [4.0]]
>>> dataset = spark.createDataFrame(dataset, ['sample'])
>>> ksResult = KolmogorovSmirnovTest.test(dataset, 'sample', 'norm', -3.0, 1.0).first()
>>> round(ksResult.pValue, 3)
1.0
>>> round(ksResult.statistic, 3)
0.175
```

버전 2.4.0의 새 기능입니다.

class pyspark.ml.stat.Summarizer

노트: 실험적

MLlib Vectors에 대한 벡터화된 통계를 위한 도구. 이 패키지의 메소드는 데이터 프레임 내부에 포함된 벡터(Vector)에 대한 다양한 통계를 제공합니다. 이 클래스는 사용자가 주어진 열에 대해 추출하고자 하는 통계를 선택할 수 있도록 합니다.

```
>>> from pyspark.ml.stat import Summarizer
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> summarizer = Summarizer.metrics("mean", "count")
>>> df = sc.parallelize([Row(weight=1.0, features=Vectors.dense(1.0, 1.0, -1.0)),
...                     Row(weight=0.0, features=Vectors.dense(1.0, 2.0, -3.0))]).toDF()
>>> df.select(summarizer.summary(df.features, df.weight)).show(truncate=False)
+-----+
| aggregate_metrics(features, weight) |
+-----+
| [[1.0,1.0,1.0], 1]                   |
+-----+
>>> df.select(summarizer.summary(df.features)).show(truncate=False)
+-----+
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

| aggregate_metrics(features, 1.0) |
+-----+
|[ [1.0,1.5,2.0], 2]           |
+-----+


>>> df.select(Summarizer.mean(df.features, df.weight)) .
    ↵show(truncate=False)
+-----+
| mean(features) |
+-----+
|[ 1.0,1.0,1.0] |
+-----+


>>> df.select(Summarizer.mean(df.features)) . show(truncate=False)
+-----+
| mean(features) |
+-----+
|[ 1.0,1.5,2.0] |
+-----+

```

버전 2.4.0의 새 기능입니다.

static count(*col, weightCol=None*)

count의 요약 열을 반환합니다.

버전 2.4.0의 새 기능입니다.

static max(*col, weightCol=None*)

max의 요약 열을 반환합니다.

버전 2.4.0의 새 기능입니다.

static mean(*col, weightCol=None*)

mean의 요약 열을 반환합니다.

버전 2.4.0의 새 기능입니다.

static metrics(*metrics)

메트릭(metrics) 목록이 지정된 경우 열에서 메트릭을 계산하는 빌더(builder)를

제공합니다. 예는 [[Summarizer]] 의 설명서를 참조하시길 바랍니다.

허용되는 메트릭은 다음과 같습니다(대소문자 구분):

- mean: 계수별(coefficient-wise) 평균을 포함하는 벡터
- variance: 계수별 분산을 포함하는 벡터
- count: 보이는 모든 벡터의 수
- numNonzeros: 각 계수에 대해 0이 아닌 개수를 갖는 벡터
- max: 각 계수의 최대값
- min: 각 계수의 최소값
- normL2: 각 계수에 대한 유clidean 놈

- normL1: 각 계수의 L1 놈(절대값들의 합)

파라미터 **metrics** – 제공될 수 있는 메트릭

반환 `pyspark.ml.stat.SummaryBuilder`의 객체

노트: 현재 이 인터페이스의 성능은 RDD인터페이스를 사용할 때보다 약 2~3배 느립니다.

버전 2.4.0의 새 기능입니다.

static min(*col*, *weightCol=None*)

min의 요약 열을 반환합니다.

버전 2.4.0의 새 기능입니다.

static normL1(*col*, *weightCol=None*)

normL1의 요약 열을 반환합니다.

버전 2.4.0의 새 기능입니다.

static normL2(*col*, *weightCol=None*)

normL2의 요약 열을 반환합니다.

버전 2.4.0의 새 기능입니다.

static numNonZeros(*col*, *weightCol=None*)

numNonZero의 요약 열을 반환합니다.

버전 2.4.0의 새 기능입니다.

static variance(*col*, *weightCol=None*)

variance의 요약 열을 반환합니다.

버전 2.4.0의 새 기능입니다.

class pyspark.ml.stat.SummaryBuilder(*jSummaryBuilder*)

노트: 실험적

지정된 열에 대한 요약 통계를 제공하는 builder 객체입니다.

사용자는 이러한 builder를 직접 만들지 않고 `pyspark.ml.stat.Summarizer`의 메서드 중 하나를 사용해야 합니다.

버전 2.4.0의 새 기능입니다.

summary(*featuresCol*, *weightCol=None*)

요청된 메트릭과 함께 열의 요약을 포함하는 집계 객체를 반환합니다.

파라미터

- **featuresCol** – 특징 벡터 객체가 들어 있는 열입니다
- **weightCol** – 가중치 값이 들어 있는 열. 기본 가중치는 1.0입니다

반환 통계를 포함하는 집계 열입니다. 이 구조의 정확한 내용은 Builder를 작성하는 동안 결정됩니다.

버전 2.4.0의 새 기능입니다.

27.2 회귀분석 API

```
class pyspark.ml.regression.AFTSurvivalRegression(*args, **kwargs)
```

노트: 실험적

가속 고장 시간(AFT, Accelerated Failure Time) 모형 생존분석

생존 시간의 Weibull 분포를 기반으로 파라미터의 AFT 생존 회귀 모형을 적합합니다.

참고 항목:

AFT 모델

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0), 1.0),
...     (1e-40, Vectors.sparse(1, [], []), 0.0)], ["label", "features",
... "censor"])
>>> afts = AFTSurvivalRegression()
>>> model = afts.fit(df)
>>> model.predict(Vectors.dense(6.3))
1.0
>>> model.predictQuantiles(Vectors.dense(6.3))
DenseVector([0.0101, 0.0513, 0.1054, 0.2877, 0.6931, 1.3863, 2.3026, 2.
˓→9957, 4.6052])
>>> model.transform(df).show()
+-----+-----+-----+
| label | features | censor | prediction |
+-----+-----+-----+
| 1.0 | [1.0] | 1.0 | 1.0 |
| 1.0E-40 | (1, [], []) | 0.0 | 1.0 |
+-----+-----+-----+
...
>>> afts_path = temp_path + "/afts"
>>> afts.save(afts_path)
>>> afts2 = AFTSurvivalRegression.load(afts_path)
>>> afts2.getMaxIter()
100
>>> model_path = temp_path + "/afts_model"
>>> model.save(model_path)
>>> model2 = AFTSurvivalRegressionModel.load(model_path)
>>> model.coefficients == model2.coefficients
True
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
>>> model.intercept == model2.intercept
True
>>> model.scale == model2.scale
True
```

버전 1.6.0의 새 기능입니다.

getCensorCol()

censorCol의 값 또는 기본값을 가져옵니다.

버전 1.6.0의 새 기능입니다.

getQuantileProbabilities()

quantileProbabilities의 값 또는 기본값을 가져옵니다.

버전 1.6.0의 새 기능입니다.

getQuantilesCol()

quantilesCol의 값 또는 기본값을 가져옵니다.

버전 1.6.0의 새 기능입니다.

setCensorCol(value)

censorCol의 값을 설정합니다.

버전 1.6.0의 새 기능입니다.

setParams(featuresCol='features', labelCol='label', predictionCol='prediction', fitIntercept=True, maxIter=100, tol=1e-06, censorCol='censor', quantileProbabilities=[0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99], quantilesCol=None, aggregationDepth=2)

setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", fitIntercept=True, maxIter=100, tol=1E-6, censorCol="censor", quantileProbabilities=[0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99], quantilesCol=None, aggregationDepth=2):

버전 1.6.0의 새 기능입니다.

setQuantileProbabilities(value)

quantileProbabilities의 값을 설정합니다.

버전 1.6.0의 새 기능입니다.

setQuantilesCol(value)

quantilesCol의 값을 설정합니다.

버전 1.6.0의 새 기능입니다.

```
class pyspark.ml.regression.AFTSurvivalRegressionModel (java_model=None)
```

노트: 실험적

*AFTSurvivalRegression*에 의해 적합된 모형

버전 1.6.0의 새 기능입니다.

property coefficients

모형 계수

버전 2.0.0의 새 기능입니다.

property intercept

모형 절편

버전 1.6.0의 새 기능입니다.

predict (features)

예측값

버전 2.0.0의 새 기능입니다.

predictQuantiles (features)

예측 분위수

버전 2.0.0의 새 기능입니다.

property scale

모형 척도 모수

버전 1.6.0의 새 기능입니다.

class pyspark.ml.regression.DecisionTreeRegressionModel (*java_model=None*)

*DecisionTreeRegressor*에 의해 적합된 모델입니다.

버전 1.4.0의 새 기능입니다.

property featureImportances

각 특징의 중요도를 추정합니다.

이것은 레오 브레이먼(Leo Breiman)과 아델 커틀러(Adele Cutler)의 "랜덤 포레스트(Random Forests)" 문서에서 지니의 중요성에 대한 설명과 scikit-learn의 구현에 따라 "지니"의 중요성에 대한 개념을 다른 손실로 일반화합니다.

특징 중요도는 다음과 같이 계산됩니다:

- 중요도(특징 j) = gain의 합계 (특징 j 에서 분할되는 노드에서), 여기서 gain은 노드를 통과하는 인스턴스의 수에 따라 비례 축소됩니다.
- 트리의 중요도를 “1/합계”로 정규화합니다.

노트: 단일 결정 트리의 특징 중요도는 상관된 예측 변수들로 인해 높은 분산을 가질 수 있습니다. 대신 *RandomForestRegressor*를 사용하여 특징 중요도를 결정하는 것을 고려해 보시길 바랍니다.

버전 2.0.0의 새 기능입니다.

class pyspark.ml.regression.DecisionTreeRegressor (*args, **kwargs)

회귀 분석을 위한 의사결정나무 학습 알고리즘으로 연속적인 특징과 범주적인 특징을 모두 지원합니다.

```

>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], [])), ["label", "features"])
>>> dt = DecisionTreeRegressor(maxDepth=2, varianceCol="variance")
>>> model = dt.fit(df)
>>> model.depth
1
>>> model.numNodes
3
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> model.numFeatures
1
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], [
...     "features"])
>>> model.transform(test1).head().prediction
1.0
>>> dtr_path = temp_path + "/dtr"
>>> dt.save(dtr_path)
>>> dt2 = DecisionTreeRegressor.load(dtr_path)
>>> dt2.getMaxDepth()
2
>>> model_path = temp_path + "/dtr_model"
>>> model.save(model_path)
>>> model2 = DecisionTreeRegressionModel.load(model_path)
>>> model.numNodes == model2.numNodes
True
>>> model.depth == model2.depth
True
>>> model.transform(test1).head().variance
0.0

```

버전 1.4.0의 새 기능입니다.

setParams (self, featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='variance', seed=None, varianceCol=None) Sets params for DecisionTreeRegressor에 대한 파라미터를 설정합니다.

버전 1.4.0의 새 기능입니다.

class pyspark.ml.regression.GBTRegressionModel (java_model=None)
*GBTRegressor*에 의해 적합된 모델입니다.

버전 1.4.0의 새 기능입니다.

evaluateEachIteration (dataset, loss)

그래디언트 부스팅을 반복할 때마다 에러 또는 손실을 계산하는 방법

파라미터

- **dataset** – 모델을 평가하기 위해 데이터 셋을 테스트합니다. 여기서 데이터 셋은 `pyspark.sql.DataFrame` 인스턴스입니다.
- **loss** – 오류 계산에 사용되는 손실 함수입니다. 지원되는 옵션: 제곱, 절대값

버전 2.4.0의 새 기능입니다.

property featureImportances

각 특징의 중요도를 추정합니다.

각 특징의 중요도는 앙상블에서 모든 트리에 걸친 중요도의 평균입니다. 중요도 벡터는 “1/합계”로 정규화됩니다. 이 방법은 Hastie et al. (Hastie, Tibshirani, Friedman. “통계 학습의 요소”, 2판.” 2001.)에 의해 제안되었습니다. 그리고 scikit-learn의 구현을 따릅니다.

참고 항목:

`DecisionTreeRegressionModel.featureImportances`

버전 2.0.0의 새 기능입니다.

property trees

null 값을 갖는 모(母) 추정치를 갖습니다.

버전 2.0.0의 새 기능입니다.

타입 앙상블에서의 트리. 경고

`class pyspark.ml.regression.GBTRegressor(*args, **kwargs)`

회귀 분석을 위한 GBTs(Gradient-Boosted Trees) 학습 알고리즘. 이는 연속형과 범주형 특징 둘 다 지원합니다.

```
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], [])), ["label", "features"])
>>> gbt = GBTRegressor(maxIter=5, maxDepth=2, seed=42)
>>> print(gbt.getImpurity())
variance
>>> print(gbt.getFeatureSubsetStrategy())
all
>>> model = gbt.fit(df)
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> model.numFeatures
1
>>> allclose(model.treeWeights, [1.0, 0.1, 0.1, 0.1, 0.1])
True
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
```

(다음 페이지에 계속)

(continued from previous page)

```

>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], [{"label": "features"}])
>>> model.transform(test1).head().prediction
1.0
>>> gbtr_path = temp_path + "gbtr"
>>> gbt.save(gbtr_path)
>>> gbt2 = GBTRegressor.load(gbtr_path)
>>> gbt2.getMaxDepth()
2
>>> model_path = temp_path + "gbtr_model"
>>> model.save(model_path)
>>> model2 = GBTRgressionModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
>>> model.treeWeights == model2.treeWeights
True
>>> model.trees
[DecisionTreeRegressionModel (uid=...) of depth..., DecisionTreeRegressionModel...]
>>> validation = spark.createDataFrame([(0.0, Vectors.dense(-1.0))], [{"label": "label"}, {"features": "features"}])
>>> model.evaluateEachIteration(validation, "squared")
[0.0, 0.0, 0.0, 0.0, 0.0]

```

버전 1.4.0의 새 기능입니다.

getLossType()

lossType 값 또는 기본값을 가져옵니다.

버전 1.4.0의 새 기능입니다.

setFeatureSubsetStrategy(value)

featureSubsetStrategy의 값을 설정합니다.

버전 1.4.0의 새 기능입니다.

setLossType(value)

lossType의 값을 설정합니다.

버전 1.4.0의 새 기능입니다.

setParams(self, featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, subsamplingRate=1.0, checkpointInterval=10, lossType='squared', maxIter=20, stepSize=0.1, seed=None, impurity='variance', featureSubsetStrategy='all')

그래디언트 부스팅 트리 회귀 분석에 대한 파라미터를 설정합니다.

버전 1.4.0의 새 기능입니다.

```

class pyspark.ml.regression.GeneralizedLinearRegression(*args,
**kwargs)

```

노트: 실험적

일반화 선형 회귀 분석.

선형 예측 변수(링크 함수)에 대한 기호 설명과 오차 분포(패밀리)에 대한 설명을 제공하여 지정된 일반화 선형 모델을 적합합니다. 이 모델은 "가우시안", "이항", "포아송", "감마", "tweedie"를 패밀리로 합니다. 각 패밀리에 대한 유효한 링크 함수는 아래와 같습니다. 각 패밀리의 첫 번째 링크 함수는 기본 링크 함수입니다.

- “gaussian” -> “identity”, “log”, “inverse”
- “binomial” -> “logit”, “probit”, “cloglog”
- “poisson” -> “log”, “identity”, “sqrt”
- “gamma” -> “inverse”, “identity”, “log”
- “tweedie” -> "linkPower"를 통해 지정된 power 연결 함수. tweedie 패밀리의 기본 연결 power는 1 – variancePower입니다.

참고 항목:

GLM

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(0.0, 0.0)),
...     (1.0, Vectors.dense(1.0, 2.0)),
...     (2.0, Vectors.dense(0.0, 0.0)),
...     (2.0, Vectors.dense(1.0, 1.0))), ["label", "features"])
>>> glr = GeneralizedLinearRegression(family="gaussian", link="identity",
...     linkPredictionCol="p")
>>> model = glr.fit(df)
>>> transformed = model.transform(df)
>>> abs(transformed.head().prediction - 1.5) < 0.001
True
>>> abs(transformed.head().p - 1.5) < 0.001
True
>>> model.coefficients
DenseVector([1.5..., -1.0...])
>>> model.numFeatures
2
>>> abs(model.intercept - 1.5) < 0.001
True
>>> glr_path = temp_path + "/glr"
>>> glr.save(glr_path)
>>> glr2 = GeneralizedLinearRegression.load(glr_path)
>>> glr.getFamily() == glr2.getFamily()
True
>>> model_path = temp_path + "/glr_model"
>>> model.save(model_path)
>>> model2 = GeneralizedLinearRegressionModel.load(model_path)
>>> model.intercept == model2.intercept
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
True  
>>> model.coefficients[0] == model2.coefficients[0]  
True
```

버전 2.0.0의 새 기능입니다.

getFamily()

패밀리 값 또는 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

getLink()

링크 값 또는 해당 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

getLinkPower()

linkPower 값 또는 기본값을 가져옵니다.

버전 2.2.0의 새 기능입니다.

getLinkPredictionCol()

linkPredictionCol 값 또는 해당 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

getOffsetCol()

offsetCol 값 또는 해당 기본값을 가져옵니다.

버전 2.3.0의 새 기능입니다.

getVariancePower()

variancePower 값 또는 기본값을 가져옵니다.

버전 2.2.0의 새 기능입니다.

setFamily(value)

패밀리의 값을 설정합니다.

버전 2.0.0의 새 기능입니다.

setLink(value)

링크의 값을 설정합니다.

버전 2.0.0의 새 기능입니다.

setLinkPower(value)

linkPower의 값을 설정합니다.

버전 2.2.0의 새 기능입니다.

setLinkPredictionCol(value)

linkPredictionCol의 값을 설정합니다.

버전 2.2.0의 새 기능입니다.

setOffsetCol (value)

offsetCol의 값을 설정합니다.

버전 2.3.0의 새 기능입니다.

setParams (self, labelCol='label', featuresCol='features', predictionCol='prediction', family='gaussian', link=None, fitIntercept=True, maxIter=25, tol=1e-06, regParam=0.0, weightCol=None, solver='irls', linkPredictionCol=None, variancePower=0.0, linkPower=None, offsetCol=None)

일반 선형 회귀 분석을 위한 파라미터를 설정합니다.

버전 2.0.0의 새 기능입니다.

setVariancePower (value)

variancePower의 값을 설정합니다.

버전 2.0.0의 새 기능입니다

```
class pyspark.ml.regression.GeneralizedLinearRegressionModel (java_model=None)
```

노트: 실험적

*GeneralizedLinearRegression*에 의해 적합된 모형

버전 2.0.0의 새 기능입니다.

property coefficients

모형 계수

버전 2.0.0의 새 기능입니다.

evaluate (dataset)

검증 데이터 셋에서 모델을 평가합니다.

파라미터 **dataset** – 모델을 평가하기 위한 데이터 셋을 검증합니다. 데이터 셋은 `pyspark.sql.DataFrame`의 인스턴스입니다.

버전 2.0.0의 새 기능입니다.

property hasSummary

이 모델 인스턴스에 대한 훈련 요약이 있는지 여부를 나타냅니다.

버전 2.0.0의 새 기능입니다.

property intercept

모형 절편

버전 2.0.0의 새 기능입니다.

property summary

훈련 집합에 대한 모형의 요약(예: 잔차, 편차, pValues)을 가져옵니다.

`trainingSummaryisNone`이면 예외가 발생합니다.

버전 2.0.0의 새 기능입니다.

```
class pyspark.ml.regression.GeneralizedLinearRegressionSummary(java_obj=None)
```

노트: 실험적

데이터 셋에서 평가된 일반화된 선형 회귀 결과.

버전 2.0.0의 새 기능입니다.

property aic

Akaike의 적합 모델에 대한 "정보 기준(AIC)"

버전 2.0.0의 새 기능입니다.

property degreesOfFreedom

자유도

버전 2.0.0의 새 기능입니다.

property deviance

적합 모형의 적합도(deviance)

버전 2.0.0의 새 기능입니다.

property dispersion

적합 모형의 산포입니다. "이항식" 및 "포아송" 패밀리의 경우 1.0으로 간주되며, 그렇지 않으면 잔차인 피어슨 카이 제곱 통계량(피어슨 잔차의 제곱합으로 정의됨)을 잔차 자유도로 나눈 값으로 추정됩니다.

버전 2.0.0의 새 기능입니다.

property nullDeviance

귀무 가설 모델의 적합도

버전 2.0.0의 새 기능입니다.

property numInstances

데이터 프레임 예측의 인스턴스 수입니다.

버전 2.0.0의 새 기능입니다.

property predictionCol

각 인스턴스의 예측 값을 제공하는 *predictions* 필드입니다. 원래 모델의 *predictionCol* 가 설정되어 있지 않으면 새 열 이름으로 설정됩니다.

버전 2.0.0의 새 기능입니다.

property predictions

모형의 *transform* 방법에 의해 출력되는 예측값입니다.

버전 2.0.0의 새 기능입니다.

property rank

적합 선형 모형의 수치적 순위입니다.

버전 2.0.0의 새 기능입니다.

property residualDegreeOfFreedom
잔차 자유도

버전 2.0.0의 새 기능입니다.

property residualDegreeOfFreedomNull
귀무 가설 모형의 잔차 자유도입니다.

버전 2.0.0의 새 기능입니다.

residuals (*residualsType='deviance'*)
유형별로 적합 모형의 잔차를 가져옵니다.

파라미터 **residualsType** – 반환해야 하는 잔차의 유형입니다.
지원되는 옵션: deviance (기본값), pearson, working 및 response.

버전 2.0.0의 새 기능입니다.

class pyspark.ml.regression.GeneralizedLinearRegressionTrainingSummary (*java_obj=None*)

노트: 실험적

일반화된 선형 회귀 훈련 결과.

버전 2.0.0의 새 기능입니다.

property coefficientStandardErrors
추정 계수 및 절편의 표준 오차입니다.

GeneralizedLinearRegression.fitIntercept 가 True로 설정되어 있으면 마지막으로 반환된 요소가 절편에 해당합니다.

버전 2.0.0의 새 기능입니다.

property numIterations
훈련 반복 횟수

버전 2.0.0의 새 기능입니다.

property pValues
추정된 계수와 절편의 양측 p-value입니다.

GeneralizedLinearRegression.fitIntercept 가 True로 설정되어 있으면 마지막으로 반환된 요소가 절편에 해당합니다.

버전 2.0.0의 새 기능입니다.

property solver
훈련에 사용되는 수치형 solver입니다.

버전 2.0.0의 새 기능입니다.

property tValues
추정 계수 및 절편의 T-통계량

GeneralizedLinearRegression.fitIntercept가 True로 설정되어 있으면 마지막으로 반환된 요소가 절편에 해당합니다.

버전 2.0.0의 새 기능입니다.

class pyspark.ml.regression.IsotonicRegression(*args, **kwargs)

현재 병렬화된 풀 인접 위반자(parallelized pool adjacent violators) 알고리즘을 사용하여 구현되었으며, 단변량(단일 특징) 알고리즘만 지원됩니다.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> ir = IsotonicRegression()
>>> model = ir.fit(df)
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
>>> model.boundaries
DenseVector([0.0, 1.0])
>>> ir_path = temp_path + "/ir"
>>> ir.save(ir_path)
>>> ir2 = IsotonicRegression.load(ir_path)
>>> ir2.getIsotonic()
True
>>> model_path = temp_path + "/ir_model"
>>> model.save(model_path)
>>> model2 = IsotonicRegressionModel.load(model_path)
>>> model.boundaries == model2.boundaries
True
>>> model.predictions == model2.predictions
True
```

버전 1.6.0의 새 기능입니다.

getFeatureIndex()

featureIndex 값 또는 기본값을 가져옵니다

getIsotonic()

isotonic 값 또는 해당 기본값을 가져옵니다

setFeatureIndex(value)

featureIndex의 값을 설정합니다

setIsotonic(value)

isotonic의 값을 설정합니다

setParams(featuresCol='features', labelCol='label', predictionCol='prediction', weightCol=None, isotonic=True, featureIndex=0)

setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", weightCol=None, isotonic=True, featureIndex=0): IsotonicRegression에 대한 파라미터를 설정합니다.

class pyspark.ml.regression.IsotonicRegressionModel(java_model=None)

*IsotonicRegression*에 의해 적합된 모형

버전 1.6.0의 새 기능입니다.

property boundaries

예측값을 알 수 있는 증가하는 순서의 경계입니다.

버전 1.6.0의 새 기능입니다.

property predictions

동일한 인덱스에서 경계와 연관된 예측으로 isotonic 회귀분석 때문에 단조로운 값을 갖습니다.

버전 1.6.0의 새 기능입니다.

```
class pyspark.ml.regression.LinearRegression(*args, **kwargs)
```

선형 회귀 분석.

학습 목표는 정칙화를 통해 지정된 손실 함수를 최소화하는 것입니다. 이는 두 가지 유형의 손실을 지원합니다:

- 제곱 오차(제곱 손실이라고 알려짐)
- huber (상대적으로 작은 오차의 제곱 오차와 상대적으로 큰 오차의 절대 오차의 혼합, 훈련 데이터로부터 척도 모수를 추정합니다)

여러 유형의 정칙화를 지원합니다:

- none (일명 최소제곱법)
- L2 (ridge 회귀)
- L1 (Lasso)
- L2 + L1 (엘라스틱 넷)

노트: huber loss 적합 시 none 과 L2 정칙화만 지원됩니다

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, 2.0, Vectors.dense(1.0)),
...     (0.0, 2.0, Vectors.sparse(1, [], [])), ["label", "weight",
...     "features"])
>>> lr = LinearRegression(maxIter=5, regParam=0.0, solver="normal",
... weightCol="weight")
>>> model = lr.fit(df)
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> abs(model.transform(test0).head().prediction - (-1.0)) < 0.001
True
>>> abs(model.coefficients[0] - 1.0) < 0.001
True
>>> abs(model.intercept - 0.0) < 0.001
True
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], [
...     "features"])
>>> abs(model.transform(test1).head().prediction - 1.0) < 0.001
True
>>> lr.setParams("vector")
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

Traceback (most recent call last):
...
TypeError: Method setParams forces keyword arguments.
>>> lr_path = temp_path + "/lr"
>>> lr.save(lr_path)
>>> lr2 = LinearRegression.load(lr_path)
>>> lr2.getMaxIter()
5
>>> model_path = temp_path + "/lr_model"
>>> model.save(model_path)
>>> model2 = LinearRegressionModel.load(model_path)
>>> model.coefficients[0] == model2.coefficients[0]
True
>>> model.intercept == model2.intercept
True
>>> model.numFeatures
1
>>> model.write().format("pmm1").save(model_path + "_2")

```

버전 1.4.0의 새 기능입니다.

getEpsilon()

엡실론(오차항) 값 또는 기본값을 가져옵니다.

버전 2.3.0의 새 기능입니다.

setEpsilon(value)

엡실론(오차항)의 값을 설정합니다.

버전 2.3.0의 새 기능입니다

setParams(self, featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-06, fitIntercept=True, standardization=True, solver='auto', weightCol=None, aggregationDepth=2, loss='squaredError', epsilon=1.35)

선형 회귀 분석에 대한 파라미터를 설정합니다

다버전 1.4.0의 새 기능입니다.

class pyspark.ml.regression.LinearRegressionModel(java_model=None)
`LinearRegression`에 의해 적합된 모형

버전 1.4.0의 새 기능입니다.

property coefficients

모형 계수

버전 2.0.0의 새 기능입니다.

evaluate(dataset)

검증 데이터 셋에서 모델을 평가합니다.

Parameters dataset – 모델을 평가할 데이터 셋을 검증합니다. 여기서 데이터 집합은 `pyspark.sql.DataFrame`의 인스턴스입니다.

버전 2.0.0의 새 기능입니다.

property hasSummary

모델 인스턴스에 대한 훈련 요약이 있는지 여부를 나타냅니다.

버전 2.0.0의 새 기능입니다.

property intercept

모형의 절편

버전 1.4.0의 새 기능입니다.

property scale

순실이 "uber"일 때 $\|y - Xw\|$ 가 축소되는 값이며, 그렇지 않으면 1.0입니다.

버전 2.3.0의 새 기능입니다.

property summary

훈련 집합에 대한 모형의 요약(예: 잔차, mse, r-제곱)을 가져옵니다.

*trainingSummary*은 `None`인 경우 예외가 발생합니다.

버전 2.0.0의 새 기능입니다.

class pyspark.ml.regression.LinearRegressionSummary (java_obj=None)

노트: 실험적

데이터 셋에서 평가된 선형 회귀 결과입니다.

버전 2.0.0의 새 기능입니다.

property coefficientStandardErrors

추정된 계수와 절편의 표준 오차입니다. 이 값은 "normal" solver를 사용할 때만 사용할 수 있습니다.

`LinearRegression.fitIntercept` 가 `True`로 설정되어 있으면 마지막으로 반환된 요소가 절편에 해당합니다.

참고 항목:

`LinearRegression.solver`

버전 2.0.0의 새 기능입니다.

property degreesOfFreedom

자유도

버전 2.2.0의 새 기능입니다.

property devianceResiduals

가중 잔차, 일반적인 잔차는 인스턴스 가중치의 제곱근으로 재스케일링됩니다.

버전 2.0.0의 새 기능입니다.

property explainedVariance

설명된 분산 회귀 분석 점수를 반환합니다. 설명된 분산 = $1 - \frac{\text{variance}(y - \hat{y})}{\text{variance}(y)}$
참고 항목:

위키피디아 설명된 분산

노트 이는 *LinearRegression.weightCol*의 인스턴스 가중치(모두 1.0으로 설정)를 무시합니다. 이는 이후 스파크 버전에서 변경될 것입니다.

버전 2.0.0의 새 기능입니다.

property featuresCol

각 인스턴스의 특징을 벡터로 제공하는 “예측(predictions)” 필드입니다.

버전 2.0.0의 새 기능입니다.

property labelCol

각 인스턴스의 실제 라벨을 제공하는 “예측(predictions)” 필드입니다.

버전 2.0.0의 새 기능입니다.

property meanAbsoluteError

절대 오차 손실 또는 l1-norm 손실의 예상 값에 해당하는 위험 함수인 평균 절대 오차를 반환합니다.

노트 *LinearRegression.weightCol*의 인스턴스 가중치(모두 1.0으로 설정)를 무시합니다. 이는 이후 스파크 버전에서 변경될 것입니다.

버전 2.0.0의 새 기능입니다.

property meanSquaredError

제곱 오차 손실 또는 이차 손실의 예상 값에 해당하는 위험 함수인 평균 제곱 오차를 반환합니다.

노트: 이는 *LinearRegression.weightCol*의 인스턴스 가중치(모두 1.0으로 설정)를 무시합니다. 이는 이후 스파크 버전에서 변경될 것입니다.

버전 2.0.0의 새 기능입니다.

property numInstances

데이터 프레임 예측의 인스턴스 수

버전 2.0.0의 새 기능입니다.

property pValues

추정된 계수와 절편의 양측 p-value입니다. 이 값은 "normal" solver를 사용할 때만 사용할 수 있습니다.

*LinearRegression.fitIntercept*가 True로 설정되어 있으면 마지막으로 반환된 요소가 절편에 해당합니다.

참고 항목:

LinearRegression.solver

버전 2.0.0의 새 기능입니다.

property predictionCol

각 인스턴스에서 라벨의 예측 값을 제공하는 "예측(predictions)" 필드입니다.

버전 2.0.0의 새 기능입니다.

property predictions

모형의 *transform* 방법에 의해 출력되는 데이터 프레임

버전 2.0.0의 새 기능입니다.

property r2

결정계수인 R^2를 반환합니다.

참고 항목:

위키피디아 결정 계수

노트: 이는 *LinearRegression.weightCol*의 인스턴스 가중치(모두 1.0으로 설정)를 무시합니다. 이는 이후 스파크 버전에서 변경될 것입니다.

버전 2.0.0의 새 기능입니다

property r2adj

조정된 결정 계수인 Adjusted R^2를 반환합니다.

참고 항목:

위키피디아 조정된 결정계수, Adjusted R^2

노트: 이는 *LinearRegression.weightCol*의 인스턴스 가중치(모두 1.0으로 설정)를 무시합니다. 이는 이후 스파크 버전에서 변경될 것입니다.

버전 2.4.0의 새 기능입니다

property residuals

잔차(라벨 - 예측값)

버전 2.0.0의 새 기능입니다.

property rootMeanSquaredError

평균 제곱 오차의 제곱근으로 정의되는 제곱근 평균 제곱 오차를 반환합니다.

노트: 이는 *LinearRegression.weightCol*의 인스턴스 가중치(모두 1.0으로 설정)를 무시합니다. 이는 이후 스파크 버전에서 변경될 것입니다.

버전 2.0.0의 새 기능입니다.

property tValues

추정된 계수와 절편에 대한 T-통계량. 이 값은 "normal" solver를 사용할 때만 사용할 수 있습니다.

LinearRegression.fitIntercept 가 True로 설정되어 있으면 마지막으로 반환된 요소가 절편에 해당합니다.

참고 항목:

LinearRegression.solver

버전 2.0.0의 새 기능입니다.

class pyspark.ml.regression.LinearRegressionTrainingSummary (java_obj=None)

노트: 실험적

선형 회귀 훈련 결과. 현재 훈련 요약은 목적 트레이스(trace)를 제외하고는 훈련 가중치를 무시하고 있습니다.

버전 2.0.0의 새 기능입니다.

property objectiveHistory

각 반복에서 목적 함수(스케일 된 손실 + 정칙화). 이 값은 "l-bfgs" solver를 사용할 때만 사용할 수 있습니다.

참고 항목:

LinearRegression.solver

버전 2.0.0의 새 기능입니다.

property totalIterations

종료 전까지 훈련을 반복한 횟수입니다. 이 값은 "l-bfgs" solver를 사용할 때만 사용할 수 있습니다.

참고 항목:

LinearRegression.solver

버전 2.0.0의 새 기능입니다..

class pyspark.ml.regression.RandomForestRegressionModel (java_model=None)

*RandomForestRegressor*에 의해 적합된 모형

버전 1.4.0의 새 기능입니다.

property featureImportances

각 특징의 중요도를 추정합니다.

각 특징의 중요도는 앙상블의 모든 트리에 걸친 중요도의 평균입니다. 중요도 벡터는 1/합계로 정규화됩니다. 이 방법은 Hastie et al. (Hastie, Tibshirani, Friedman. "통계 학습의

요소, 2판." 2001.)에 의해 제안되었으며, scikit-learn의 구현을 따릅니다.

참고 항목:

DecisionTreeRegressionModel.featureImportances

버전 2.0.0의 새 기능입니다.

property trees

이들은 null값을 갖는 모 추정치입니다.

버전 2.0.0의 새 기능입니다.

타입 양상블에서 트리.경고

class pyspark.ml.regression.**RandomForestRegressor**(*args, **kwargs)

회귀 분석을 위한 랜덤 포레스트 학습 알고리즘으로 연속적인 특징과 범주적인 특징을 모두 지원합니다.

```
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], [])), ["label", "features"])
>>> rf = RandomForestRegressor(numTrees=2, maxDepth=2, seed=42)
>>> model = rf.fit(df)
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> allclose(model.treeWeights, [1.0, 1.0])
True
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
>>> model.numFeatures
1
>>> model.trees
[DecisionTreeRegressionModel (uid=...) of depth..., 
 DecisionTreeRegressionModel...]
>>> model.getNumTrees
2
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], [
... "features"])
>>> model.transform(test1).head().prediction
0.5
>>> rfr_path = temp_path + "/rfr"
>>> rf.save(rfr_path)
>>> rf2 = RandomForestRegressor.load(rfr_path)
>>> rf2.getNumTrees()
2
>>> model_path = temp_path + "/rfr_model"
>>> model.save(model_path)
>>> model2 = RandomForestRegressionModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
```

버전 1.4.0의 새 기능입니다.

setFeatureSubsetStrategy (value)

featureSubsetStrategy의 값을 설정합니다.

버전 2.4.0의 새 기능입니다.

setParams (self, featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='variance', subsamplingRate=1.0, seed=None, numTrees=20, featureSubsetStrategy='auto')

선행 회귀 분석에 대한 파라미터를 설정합니다.

버전 1.4.0의 새 기능입니다

27.3 분류 API

class pyspark.ml.classification.BinaryLogisticRegressionSummary (java_obj=None)

노트: 실험적

지정된 모형에 대한 이항 로지스틱 회귀 분석 결과입니다.

버전 2.0.0의 새 기능입니다.

property areaUnderROC

수신자 작동 특성 Receiver Operating Characteristic (ROC) 곡선 아래의 영역을 계산합니다.

노트: 이는 *LogisticRegression.weightCol*의 인스턴스 가중치(모두 1.0으로 설정)를 무시합니다. 이는 이후 스파크 버전에서 변경됩니다.

버전 2.0.0의 새 기능입니다.

property fMeasureByThreshold

beta = 1.0 인 두 필드(임계값, F-측도) 곡선이 있는 데이터 프레임을 반환합니다

노트: 이는 *LogisticRegression.weightCol*의 인스턴스 가중치(모두 1.0으로 설정)를 무시합니다. 이는 이후 스파크 버전에서 변경될 것입니다.

버전 2.0.0의 새 기능입니다.

property pr

정밀도-재현율 곡선을 반환합니다. 이 곡선은 곡선 앞에 (0.0, 1.0)을 붙이고 두 필드 재현율과 정밀도를 포함하는 데이터 프레임입니다.

노트: 이는 *LogisticRegression.weightCol*의 인스턴스 가중치(모두 1.0으로 설정)를 무시합니다. 이는 이후 스파크 버전에서 변경됩니다.

버전 2.0.0의 새 기능입니다.

property precisionByThreshold

두 개의 필드(임계값, 정밀도) 곡선이 있는 데이터 프레임을 반환합니다. 모든 가능한 확률은 정밀도 계산에 사용되는 임계값으로 사용되는 데이터 셋을 변환할 때 얻어집니다.

노트: 이는 *LogisticRegression.weightCol*의 인스턴스 가중치(모두 1.0으로 설정)를 무시합니다. 이는 이후 스파크 버전에서 변경됩니다.

버전 2.0.0의 새 기능입니다.

property recallByThreshold

두 개의 필드(임계값, 재현율)가 있는 데이터 프레임을 반환합니다. 모든 가능한 확률은 재현율 계산에 사용되는 임계값으로 사용되는 데이터 셋을 변환할 때 얻어집니다.

노트: 이는 *LogisticRegression.weightCol*의 인스턴스 가중치(모두 1.0으로 설정)를 무시합니다. 이는 이후 스파크 버전에서 변경됩니다.

버전 2.0.0의 새 기능입니다.

property roc

수신자 작동 특성 (ROC) 곡선을 반환합니다. 이 곡선은 두 개의 필드(FPR, TPR) 앞에 (0.0, 0.0)이 붙고 (1.0, 1.0)이 추가된 데이터 프레임입니다.

참고 항목:

[위키피디아 참조](#)

노트: 이는 *LogisticRegression.weightCol*의 인스턴스 가중치(모두 1.0으로 설정)를 무시합니다. 이는 이후 스파크 버전에서 변경됩니다.

버전 2.0.0의 새 기능입니다.

class pyspark.ml.classification.BinaryLogisticRegressionTrainingSummary (*java_obj=None*)

노트: 실험적

지정된 모형에 대한 이항 로지스틱 회귀 분석 훈련 결과입니다.

버전 2.0.0의 새 기능입니다.

class pyspark.ml.classification.DecisionTreeClassificationModel (*java_model=None*)
DecisionTreeClassifier에 의해 적합된 모형입니다.

버전 1.4.0의 새 기능입니다.

property featureImportances

각 특징의 중요도를 추정합니다.

이것은 레오 브레이먼(Leo Breiman)과 아델 커틀러(Adele Cutler)의 "랜덤 포레스트(Random Forests)" 문서에서 지니의 중요성에 대한 설명과 scikit-learn의 구현에 따라 "지니"의 중요성에 대한 개념을 다른 손실로 일반화합니다.

특징 중요도는 다음과 같이 계산됩니다:

- 중요도(특징 j) = gain의 합계 (특징 j 에서 분할되는 노드에서), 여기서 gain은 노드를 통과하는 인스턴스의 수에 따라 비례 축소됩니다.
- 트리의 중요도를 "1/합계"로 정규화합니다.

노트: 단일 결정 트리의 특징 중요도는 상관된 예측 변수들로 인해 높은 분산을 가질 수 있습니다. 대신 RandomForestClassifier를 사용하여 특징 중요도를 결정하는 것을 고려해 보십시오.

버전 2.0.0의 새 기능입니다.

class pyspark.ml.classification.DecisionTreeClassifier (*args, **kwargs)
회귀분석을 위한 의사결정나무 학습 알고리즘으로 연속적인 특징과 범주적인 특징을 모두 지원합니다.

```
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
>>> si_model = stringIndexer.fit(df)
>>> td = si_model.transform(df)
>>> dt = DecisionTreeClassifier(maxDepth=2, labelCol="indexed")
>>> model = dt.fit(td)
>>> model.numNodes
3
>>> model.depth
1
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> model.numFeatures
1
>>> model.numClasses
2
>>> print(model.toDebugString)
DecisionTreeClassificationModel (uid=...) of depth 1 with 3 nodes...
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
```

(다음 페이지에 계속)

(○] 전 페이지에서 계속)

```
>>> result = model.transform(test0).head()
>>> result.prediction
0.0
>>> result.probability
DenseVector([1.0, 0.0])
>>> result.rawPrediction
DenseVector([1.0, 0.0])
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], [
    "features"])
>>> model.transform(test1).head().prediction
1.0
```

```
>>> dtc_path = temp_path + "/dtc"
>>> dt.save(dtc_path)
>>> dt2 = DecisionTreeClassifier.load(dtc_path)
>>> dt2.getMaxDepth()
2
>>> model_path = temp_path + "/dtc_model"
>>> model.save(model_path)
>>> model2 = DecisionTreeClassificationModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
```

버전 1.4.0의 새 기능입니다.

setParams (*self, featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability', rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='gini', seed=None*)
DecisionTreeClassifier에 대한 파라미터를 설정합니다.

버전 1.4.0의 새 기능입니다.

class pyspark.ml.classification.**GBTClassificationModel** (*java_model=None*)
GBTClassifier에 의해 적합된 모델입니다.

버전 1.4.0의 새 기능입니다.

evaluateEachIteration (*dataset*)

그래디언트 부스팅을 반복할 때마다 에러 또는 손실을 계산하는 방법

파라미터 dataset – 모델을 평가하기 위해 데이터 셋을 검증합니다. 여기서 데이터 셋은 pyspark.sql.DataFrame의 인스턴스입니다.

버전 2.4.0의 새 기능입니다.

property featureImportances

각 특징의 중요도를 추정합니다.

각 특징의 중요도는 양상블에서 모든 트리에 걸친 중요도의 평균입니다. 중요도 벡터는 “1/합계”로 정규화됩니다. 이 방법은 Hastie et al. (Hastie, Tibshirani, Friedman. “통계 학습의 요소, 2판.” 2001.)에 의해 제안되었습니다. 그리고 scikit-learn의 구현을 따릅니다.

참고 항목:

`DecisionTreeClassificationModel.featureImportances`

버전 2.0.0의 새 기능입니다.

property trees

null 값을 갖는 모(母) 추정치를 갖습니다.

버전 2.0.0의 새 기능입니다.

타입 양상불에서의 트리. 경고

class pyspark.ml.classification.**GBTClassifier**(*args, **kwargs)

분류를 위한 GBTs(Gradient-Boosted Trees) 학습 알고리즘으로 이진 라벨과 연속 및 범주형 특징을 모두 지원합니다.

구현은 다음에 기반을 두고 있습니다: J.H. Friedman. “확률 그래디언트 부스팅.” 1999.

그래디언트 부스팅 vs. TreeBoost에 대한 참고: - 이 구현은 확률적 그래디언트 부스팅을 위한 것이지, TreeBoost를 위한 것은 아닙니다. - 두 알고리즘 모두 손실 함수를 최소화하여 트리 양상불을 학습합니다. - TreeBoost (Friedman, 1999) 은 손실 함수에 기초하여 트리 리프 노드들의 출력들을 추가적으로 수정하지만, 기존의 그래디언트 부스팅 방법은 그렇지 않습니다. - 향후에 TreeBoost 부스트를 구현하는 것을 기대합니다: SPARK-4240

노트: 다중 클래스 라벨은 현재 지원되지 않습니다.

```
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
>>> si_model = stringIndexer.fit(df)
>>> td = si_model.transform(df)
>>> gbt = GBTCClassifier(maxIter=5, maxDepth=2, labelCol="indexed", seed=42)
>>> gbt.getFeatureSubsetStrategy()
'all'
>>> model = gbt.fit(td)
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> allclose(model.treeWeights, [1.0, 0.1, 0.1, 0.1, 0.1])
True
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

>>> model.transform(test1).head().prediction
1.0
>>> model.totalNumNodes
15
>>> print(model.toDebugString)
GBTClassificationModel (uid=...)...with 5 trees...
>>> gbtc_path = temp_path + "gbtc"
>>> gbt.save(gbtc_path)
>>> gbt2 = GBTClassifier.load(gbtc_path)
>>> gbt2.getMaxDepth()
2
>>> model_path = temp_path + "gbtc_model"
>>> model.save(model_path)
>>> model2 = GBTClassificationModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
>>> model.treeWeights == model2.treeWeights
True
>>> model.trees
[DecisionTreeRegressionModel (uid=...) of depth..., ↴
 DecisionTreeRegressionModel...]
>>> validation = spark.createDataFrame([(0.0, Vectors.dense(-1.0),)],
...     ["indexed", "features"])
>>> model.evaluateEachIteration(validation)
[0.25..., 0.23..., 0.21..., 0.19..., 0.18...]
>>> model.numClasses
2

```

버전 1.4.0 새 기능입니다.

getLossType()

lossType 값 또는 기본값을 가져옵니다.

버전 1.4.0의 새 기능입니다

setFeatureSubsetStrategy(value)

featureSubsetStrategy의 값을 설정합니다.

버전 2.4.0의 새 기능입니다.

setLossType(value)

lossType의 값을 설정합니다.

버전 1.4.0의 새 기능입니다

setParams(self, featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, lossType='logistic', maxIter=20, stepSize=0.1, seed=None, subsamplingRate=1.0, featureSubsetStrategy='all')

그래디언트 부스트 트리 분류에 대한 매개변수를 설정합니다.

버전 1.4.0의 새 기능입니다.

```
class pyspark.ml.classification.LinearSVC(*args, **kwargs)
```

노트: 실험적

선형 SVM 분류기

이 이진 분류기는 OWLQN 옵티마이저를 사용하여 Hinge 손실을 최적화합니다. 현재 L2 정규화만 지원합니다.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> df = sc.parallelize([
...     Row(label=1.0, features=Vectors.dense(1.0, 1.0, 1.0)),
...     Row(label=0.0, features=Vectors.dense(1.0, 2.0, 3.0))]).toDF()
>>> svm = LinearSVC(maxIter=5, regParam=0.01)
>>> model = svm.fit(df)
>>> model.coefficients
DenseVector([0.0, -0.2792, -0.1833])
>>> model.intercept
1.0206118982229047
>>> model.numClasses
2
>>> model.numFeatures
3
>>> test0 = sc.parallelize([Row(features=Vectors.dense(-1.0, -1.0, -1.
... ↵0))]).toDF()
>>> result = model.transform(test0).head()
>>> result.prediction
1.0
>>> result.rawPrediction
DenseVector([-1.4831, 1.4831])
>>> svm_path = temp_path + "/svm"
>>> svm.save(svm_path)
>>> svm2 = LinearSVC.load(svm_path)
>>> svm2.getMaxIter()
5
>>> model_path = temp_path + "/svm_model"
>>> model.save(model_path)
>>> model2 = LinearSVCModel.load(model_path)
>>> model.coefficients[0] == model2.coefficients[0]
True
>>> model.intercept == model2.intercept
True
```

버전 2.2.0의 새 기능입니다.

```
setParams(featuresCol='features', labelCol='label', predictionCol='prediction', max-
    Iter=100, regParam=0.0, tol=1e-06, rawPredictionCol='rawPrediction',
    fitIntercept=True, standardization=True, threshold=0.0, weightCol=None,
    aggregationDepth=2)
setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", max-
```

Iter=100, regParam=0.0, tol=1e-6, rawPredictionCol="rawPrediction", fitIntercept=True, standardization=True, threshold=0.0, weightCol=None, aggregationDepth=2): 선형 SVM 분류기에 대한 파라미터를 설정합니다.

버전 2.2.0의 새 기능입니다.

```
class pyspark.ml.classification.LinearSVCModel (java_model=None)
```

노트: 실험적

선형SVC에 의해 적합된 모델

버전 2.2.0의 새 기능입니다.

property coefficients

선형 SVM 분류기의 모형 계수

버전 2.2.0의 새 기능입니다.

property intercept

선형 SVM 분류기의 모형 절편

버전 2.2.0의 새 기능입니다.

```
class pyspark.ml.classification.LogisticRegression (*args, **kwargs)
```

로지스틱 회귀분석. 이 클래스는 다항 로지스틱(소프트맥스) 및 이항 로지스틱 회귀 분석을 지원합니다.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> bdf = sc.parallelize([
...     Row(label=1.0, weight=1.0, features=Vectors.dense(0.0, 5.0)),
...     Row(label=0.0, weight=2.0, features=Vectors.dense(1.0, 2.0)),
...     Row(label=1.0, weight=3.0, features=Vectors.dense(2.0, 1.0)),
...     Row(label=0.0, weight=4.0, features=Vectors.dense(3.0, 3.0))].
    toDF()
>>> blor = LogisticRegression(regParam=0.01, weightCol="weight")
>>> blorModel = blor.fit(bdf)
>>> blorModel.coefficients
DenseVector([-1.080..., -0.646...])
>>> blorModel.intercept
3.112...
>>> data_path = "data/mllib/sample_multiclass_classification_data.txt"
>>> mdf = spark.read.format("libsvm").load(data_path)
>>> mlor = LogisticRegression(regParam=0.1, elasticNetParam=1.0, family=
    "multinomial")
>>> mlorModel = mlor.fit(mdf)
>>> mlorModel.coefficientMatrix
SparseMatrix(3, 4, [0, 1, 2, 3], [3, 2, 1], [1.87..., -2.75..., -0.50...
    ], 1)
>>> mlorModel.interceptVector
DenseVector([0.04..., -0.42..., 0.37...])
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

>>> test0 = sc.parallelize([Row(features=Vectors.dense(-1.0, 1.0))]) .
    >>> toDF()
    >>> result = blorModel.transform(test0).head()
    >>> result.prediction
1.0
    >>> result.probability
DenseVector([0.02..., 0.97...])
    >>> result.rawPrediction
DenseVector([-3.54..., 3.54...])
    >>> test1 = sc.parallelize([Row(features=Vectors.sparse(2, [0], [1.
        >>> toDF())
    >>> blorModel.transform(test1).head().prediction
1.0
    >>> blor.setParams("vector")
    Traceback (most recent call last):
    ...
TypeError: Method setParams forces keyword arguments.
    >>> lr_path = temp_path + "/lr"
    >>> blor.save(lr_path)
    >>> lr2 = LogisticRegression.load(lr_path)
    >>> lr2.getRegParam()
0.01
    >>> model_path = temp_path + "/lr_model"
    >>> blorModel.save(model_path)
    >>> model2 = LogisticRegressionModel.load(model_path)
    >>> blorModel.coefficients[0] == model2.coefficients[0]
True
    >>> blorModel.intercept == model2.intercept
True
    >>> model2
LogisticRegressionModel: uid = ..., numClasses = 2, numFeatures = 2

```

버전 1.3.0의 새 기능입니다.

getFamily()

family 값 또는 기본값을 가져옵니다.

버전 2.1.0의 새 기능입니다.

getLowerBoundsOnCoefficients()

lowerBoundsOnCoefficients 값을 가져옵니다.

버전 2.3.0의 새 기능입니다.

getLowerBoundsOnIntercepts()

lowerBoundsOnIntercepts 값을 가져옵니다.

버전 2.3.0의 새 기능입니다.

getThreshold()

이진 분류를 위한 임계값을 가져옵니다.

thresholds의 길이 2로 설정되어 있으면 (즉, 이진 분류), 다음과 동일한 임계값을 반환합니다: $\frac{1}{1 + \frac{\text{thresholds}(0)}{\text{thresholds}(1)}}$. 반면, 설정된 경우 thresholds를 반환하거나 설정

되지 않은 경우 기본값을 반환합니다.

버전 1.4.0의 새 기능입니다.

getThresholds()

thresholds 이 설정되어 있으면 해당 값을 반환합니다. 그렇지 않으면 threshold 이 설정되어 있으면 이진 분류에 해당하는 임계값을 반환합니다: (1-임계값, 임계값). 둘 다 설정되어 있지 않으면 오류가 발생합니다.

버전 1.5.0의 새 기능입니다.

getUpperBoundsOnCoefficients()

upperBoundsOnCoefficients 의 값을 설정합니다

버전 2.3.0의 새 기능입니다.

getUpperBoundsOnIntercepts()

upperBoundsOnIntercepts 의 값을 설정합니다

버전 2.3.0의 새 기능입니다.

setFamily(*value*)

family의 값을 설정합니다.

버전 2.1.0의 새 기능입니다.

setLowerBoundsOnCoefficients(*value*)

lowerBoundsOnCoefficients의 값을 설정합니다.

버전 2.3.0의 새 기능입니다.

setLowerBoundsOnIntercepts(*value*)

lowerBoundsOnIntercepts 값을 가져옵니다.

버전 2.3.0의 새 기능입니다.

setParams(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *maxIter=100*, *regParam=0.0*, *elasticNetParam=0.0*, *tol=1e-06*, *fitIntercept=True*, *threshold=0.5*, *thresholds=None*, *probabilityCol='probability'*, *rawPredictionCol='rawPrediction'*, *standardization=True*, *weightCol=None*, *aggregationDepth=2*, *family='auto'*, *lowerBoundsOnCoefficients=None*, *upperBoundsOnCoefficients=None*, *lowerBoundsOnIntercepts=None*, *upperBoundsOnIntercepts=None*)

setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-6, fitIntercept=True, threshold=0.5, thresholds=None, probabilityCol="probability", rawPredictionCol="rawPrediction", standardization=True, weightCol=None, aggregationDepth=2, family="auto", lowerBoundsOnCoefficients=None, upperBoundsOnCoefficients=None, lowerBoundsOnIntercepts=None, upperBoundsOnIntercepts=None): 로지스틱 회귀분석에 사용할 파라미터를 설정합니다. 파라미터와 임계점이 모두 설정되어 있으면 이들은 동일해야 합니다.

버전 1.3.0의 새 기능입니다.

setThreshold(*value*)

threshold을 설정합니다. 임계값이 설정된 경우 thresholds을 지웁니다.

버전 1.4.0의 새 기능입니다.

setThresholds (value)

thresholds를 설정합니다. 임계값이 설정된 경우 threshold를 지웁니다.

버전 1.5.0의 새 기능입니다.

setUpUpperBoundsOnCoefficients (value)

upperBoundsOnCoefficients의 값을 설정합니다.

버전 2.3.0의 새 기능입니다.

setUpUpperBoundsOnIntercepts (value)

upperBoundsOnIntercepts의 값을 설정합니다.

버전 2.3.0의 새 기능입니다.

class pyspark.ml.classification.LogisticRegressionModel (java_model=None)

로지스틱 회귀분석에 의해 적합된 모형

버전 1.3.0의 새 기능입니다.

property coefficientMatrix

모형 계수

버전 2.1.0의 새 기능입니다

property coefficients

이항 로지스틱 회귀 분석의 모형 계수입니다. 다항 로지스틱 회귀 분석의 경우에는 예외가 적용됩니다.

버전 2.0.0의 새 기능입니다.

evaluate (dataset)

검증 데이터 셋에서 모델을 평가합니다.

파라미터 **dataset** – 모델을 평가할 데이터 셋을 검증합니다. 여기서 데이터 셋은 pyspark.sql.DataFrame의 인스턴스입니다.

버전 2.0.0의 새 기능입니다.

property hasSummary

이 모델 인스턴스에 대한 훈련 요약이 있는지 여부를 나타냅니다.

버전 2.0.0의 새 기능입니다.

property intercept

이항 로지스틱 회귀 분석의 모형 절편입니다. 다항 로지스틱 회귀 분석의 경우에는 예외가 적용됩니다.

버전 1.4.0의 새 기능입니다.

property interceptVector

모형 절편

버전 2.1.0의 새 기능입니다.

property summary

훈련 셋에서 훈련된 모델의 요약(예: 정확도/정밀도/재현율/목적 이력/총 반복)을 가져옵니다. *trainingSummaryisNone*이면 예외가 발생합니다.

버전 2.0.0의 새 기능입니다.

```
class pyspark.ml.classification.LogisticRegressionSummary(java_obj=None)
```

노트: 실험적

주어진 모형에 대한 로지스틱 회귀 분석 결과에 대한 추출
버전 2.0.0은 새 기능입니다.

property accuracy

정확도를 반환합니다(전체 인스턴스 수 중 정확하게 분류된 인스턴스의 총 개수와 같습니다).

버전 2.3.0의 새 기능입니다.

fMeasureByLabel (beta=1.0)

각 라벨(범주)에 대한 f-점수(f-score or f-measure)를 반환합니다

버전 2.3.0의 새 기능입니다.

property falsePositiveRateByLabel

각 라벨(범주)에 대해 긍정 오류의 비율을 반환합니다.

버전 2.3.0의 새 기능입니다.

property featuresCol

각 인스턴스의 특징을 벡터로 제공하는 "예측" 필드입니다.

버전 2.0.0의 새 기능입니다.

property labelCol

각 인스턴스의 실제 라벨을 제공하는 "예측" 필드입니다.

버전 2.0.0의 새 기능입니다.

property labels

라벨의 순서를 오름차순으로 반환합니다. 이 순서는 라벨에 배열로 지정된 메트릭에 사용되는 순서와 일치합니다, 예, truePositiveRateByLabel.

노트: 대부분의 경우 $\{0.0, 1.0, \dots, \text{numClasses}-1\}$ 값이 됩니다. 그러나 훈련 세트에 라벨이 없는 경우 라벨에 모든 배열은 (예, from truePositiveRateByLabel) 기대되는 numClasses 가 아닌 numClasses-1 길이가 됩니다.

버전 2.3.0의 새 기능입니다.

property precisionByLabel

각 라벨(범주)에 대한 정밀도를 반환합니다.

버전 2.3.0의 새 기능입니다.

property predictionCol

각 클래스의 예측을 제공하는 "예측" 필드입니다.

버전 2.3.0의 새 기능입니다.

property predictions

모형 *transform* 방법에 의해 출력되는 데이터 프레임

버전 2.0.0의 새 기능입니다.

property probabilityCol

각 클래스의 확률을 벡터로 제공하는 "예측" 필드

버전 2.0.0의 새 기능입니다.

property recallByLabel

각 라벨(범주)에 대한 재현율을 반환합니다.

버전 2.3.0의 새 기능입니다.

property truePositiveRateByLabel

각 라벨(범주)에 대해 실제 양성 비율(TPR)을 반환합니다.

버전 2.3.0의 새 기능입니다.

weightedFMeasure (*beta=1.0*)

가중 평균 f-점수를 반환합니다.

버전 2.3.0의 새 기능입니다.

property weightedFalsePositiveRate

가중 거짓 양성 비율을 반환합니다.

버전 2.3.0의 새 기능입니다

property weightedPrecision

가중 평균 정밀도를 반환합니다.

버전 2.3.0의 새 기능입니다.

property weightedRecall

가중 평균 재현율을 반환합니다(정밀도, 재현율 그리고 f-점수와 동일합니다).

버전 2.3.0의 새 기능입니다.

property weightedTruePositiveRate

가중 실제 양성 비율을 반환합니다(정밀도, 재현율 그리고 f-점수와 동일합니다).

버전 2.3.0의 새 기능입니다.

class pyspark.ml.classification.LogisticRegressionTrainingSummary (*java_obj=None*)

노트: 실험적

다항 로지스틱 회귀 분석 훈련 결과에 대한 추출. 현재 훈련 요약은 목적 트레이스를 제외한 훈련 가중치를 무시합니다. 버전 2.0.0의 새 기능입니다.

property objectiveHistory

각 반복에서의 목적 함수(비례 축소된 손실 + 정규화)

버전 2.0.0의 새 기능입니다.

property totalIterations

종료 전까지 훈련을 반복한 횟수입니다.

버전 2.0.0의 새 기능입니다.

class pyspark.ml.classification.MultilayerPerceptronClassificationModel (java_model=None)
MultilayerPerceptronClassifier에 의해 적합된 모형

버전 1.6.0의 새 기능입니다.

property layers

입력 레이어 및 출력 레이어를 포함한 레이어 크기 배열

버전 1.6.0의 새 기능입니다.

property weights

레이어의 가중치

버전 2.0.0의 새 기능입니다

class pyspark.ml.classification.MultilayerPerceptronClassifier (*args, **kwargs)

다층 퍼셉트론에 기초한 분류기 훈련기. 각 레이어는 시그모이드 활성화 함수를 가지고, 출력 레이어는 소프트맥스를 가지고 있습니다. 입력의 수는 특징 벡터의 크기와 같아야 합니다. 출력의 수는 전체 라벨의 수와 같아야 합니다.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (0.0, Vectors.dense([0.0, 0.0])),
...     (1.0, Vectors.dense([0.0, 1.0])),
...     (1.0, Vectors.dense([1.0, 0.0])),
...     (0.0, Vectors.dense([1.0, 1.0])),], ["label", "features"])
>>> mlp = MultilayerPerceptronClassifier(maxIter=100, layers=[2, 2, 2],  
    blockSize=1, seed=123)
>>> model = mlp.fit(df)
>>> model.layers
[2, 2, 2]
>>> model.weights.size
12
>>> testDF = spark.createDataFrame([
...     (Vectors.dense([1.0, 0.0]),),
...     (Vectors.dense([0.0, 0.0]),), ["features"])
>>> model.transform(testDF).select("features", "prediction").show()
+-----+-----+
| features|prediction|
+-----+-----+
|[1.0,0.0]|      1.0|
|[0.0,0.0]|      0.0|
+-----+-----+
...
>>> mlp_path = temp_path + "/mlp"
>>> mlp.save(mlp_path)
>>> mlp2 = MultilayerPerceptronClassifier.load(mlp_path)
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
>>> mlp2.getBlockSize()
1
>>> model_path = temp_path + "/mlp_model"
>>> model.save(model_path)
>>> model2 = MultilayerPerceptronClassificationModel.load(model_path)
>>> model.layers == model2.layers
True
>>> model.weights == model2.weights
True
>>> mlp2 = mlp2.setInitialWeights(list(range(0, 12)))
>>> model3 = mlp2.fit(df)
>>> model3.weights != model2.weights
True
>>> model3.layers == model.layers
True
```

버전 1.6.0의 새 기능입니다.

getBlockSize()

blockSize 값 또는 기본값을 가져옵니다.

버전 1.6.0의 새 기능입니다.

getInitialWeights()

initialWeights 값 또는 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

getLayers()

layers 값 또는 기본값을 가져옵니다.

버전 1.6.0의 새 기능입니다.

getStepSize()

stepSize 값 또는 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

setBlockSize(*value*)

blockSize의 값을 설정합니다.

버전 1.6.0의 새 기능입니다.

setInitialWeights(*value*)

initialWeights의 값을 설정합니다.

버전 2.0.0의 새 기능입니다.

setLayers(*value*)

layers의 값을 설정합니다.

버전 1.6.0의 새 기능입니다.

setParams (*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *maxIter=100*, *tol=1e-06*, *seed=None*, *layers=None*, *blockSize=128*, *stepSize=0.03*, *solver='l-bfgs'*, *initialWeights=None*, *probabilityCol='probability'*, *rawPredictionCol='rawPrediction'*)
 setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", maxIter=100, tol=1e-6, seed=None, layers=None, blockSize=128, stepSize=0.03, solver="l-bfgs", initialWeights=None, probabilityCol="probability", rawPredictionCol="rawPrediction"):
 MultilayerPerceptronClassifier에 대한 파라미터를 설정합니다.
 버전 1.6.0의 새 기능입니다.

setStepSize (*value*)
 stepSize의 값을 설정합니다.
 버전 2.0.0의 새 기능입니다.

class pyspark.ml.classification.NaiveBayes (*args, **kwargs)
 나이브 베이즈 분류기. 다항식과 베르누이 NB를 모두 지원합니다. **다항식 NB**는 유한하게 지원되는 이산 데이터를 다룰 수 있습니다. 예를 들어, 문서를 TF-IDF 벡터로 변환하여 문서 분류에 사용할 수 있습니다. 모든 벡터를 이진(0/1) 데이터로 만들어 베르누이 NB와 같이 사용할 수 있습니다. 입력 특징 값들이 음수가 아니어야 합니다.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     Row(label=0.0, weight=0.1, features=Vectors.dense([0.0, 0.0])),
...     Row(label=0.0, weight=0.5, features=Vectors.dense([0.0, 1.0])),
...     Row(label=1.0, weight=1.0, features=Vectors.dense([1.0, 0.0]))])
>>> nb = NaiveBayes(smoothing=1.0, modelType="multinomial", weightCol=
  "weight")
>>> model = nb.fit(df)
>>> model.pi
DenseVector([-0.81..., -0.58...])
>>> model.theta
DenseMatrix(2, 2, [-0.91..., -0.51..., -0.40..., -1.09...], 1)
>>> test0 = sc.parallelize([Row(features=Vectors.dense([1.0, 0.0]))]).
  toDF()
>>> result = model.transform(test0).head()
>>> result.prediction
1.0
>>> result.probability
DenseVector([0.32..., 0.67...])
>>> result.rawPrediction
DenseVector([-1.72..., -0.99...])
>>> test1 = sc.parallelize([Row(features=Vectors.sparse(2, [0], [1.
  -0]))]).toDF()
>>> model.transform(test1).head().prediction
1.0
>>> nb_path = temp_path + "/nb"
>>> nb.save(nb_path)
>>> nb2 = NaiveBayes.load(nb_path)
>>> nb2.getSmoothing()
1.0
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
>>> model_path = temp_path + "/nb_model"
>>> model.save(model_path)
>>> model2 = NaiveBayesModel.load(model_path)
>>> model.pi == model2.pi
True
>>> model.theta == model2.theta
True
>>> nb = nb.setThresholds([0.01, 10.00])
>>> model3 = nb.fit(df)
>>> result = model3.transform(test0).head()
>>> result.prediction
0.0
```

버전 1.5.0의 새 기능입니다.

getModelType()

modelType 값 또는 기본값을 가져옵니다.

버전 1.5.0의 새 기능입니다.

getSmoothing()

smoothing(평활값) 또는 기본값을 가져옵니다.

버전 1.5.0의 새 기능입니다.

setModelType(value)

modelType의 값을 설정합니다.

버전 1.5.0의 새 기능입니다.

setParams(self, featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability', rawPredictionCol='rawPrediction', smoothing=1.0, modelType='multinomial', thresholds=None, weightCol=None)

나이브 베이즈의 파라미터를 설정합니다.

버전 1.5.0의 새 기능입니다.

setSmoothing(value)

smoothing의 값을 설정합니다.

버전 1.5.0의 새 기능입니다.

class pyspark.ml.classification.NaiveBayesModel(java_model=None)

나이브 베이즈에 의해 적합된 모형

버전 1.5.0의 새 기능입니다.

property pi

클래스 사전들의(priors) 로그

버전 2.0.0의 새 기능입니다.

property theta

클래스 조건부 확률들의 로그

버전 2.0.0의 새 기능입니다.

```
class pyspark.ml.classification.OneVsRest(*args, **kwargs)
```

노트: 실험적

다중 클래스 분류를 이진 분류로 축소합니다. 모든 전략에 대해 하나를 사용하여 축소를 수행합니다. k개의 클래스가 있는 다중 클래스 분류의 경우, k개의 모델(클래스당 하나)을 훈련 시킵니다. 각 예제는 모든 k개의 모델에 대해 점수를 매기고 가장 높은 점수를 받은 모델이 라벨 예제에 지정됩니다.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> data_path = "data/mllib/sample_multiclass_classification_data.txt"
>>> df = spark.read.format("libsvm").load(data_path)
>>> lr = LogisticRegression(regParam=0.01)
>>> ovr = OneVsRest(classifier=lr)
>>> model = ovr.fit(df)
>>> model.models[0].coefficients
DenseVector([0.5..., -1.0..., 3.4..., 4.2...])
>>> model.models[1].coefficients
DenseVector([-2.1..., 3.1..., -2.6..., -2.3...])
>>> model.models[2].coefficients
DenseVector([0.3..., -3.4..., 1.0..., -1.1...])
>>> [x.intercept for x in model.models]
[-2.7..., -2.5..., -1.3...]
>>> test0 = sc.parallelize([Row(features=Vectors.dense(-1.0, 0.0, 1.0, 1.
->0))]).toDF()
>>> model.transform(test0).head().prediction
0.0
>>> test1 = sc.parallelize([Row(features=Vectors.sparse(4, [0], [1.
->0]))]).toDF()
>>> model.transform(test1).head().prediction
2.0
>>> test2 = sc.parallelize([Row(features=Vectors.dense(0.5, 0.4, 0.3, 0.
->2))]).toDF()
>>> model.transform(test2).head().prediction
0.0
>>> model_path = temp_path + "/ovr_model"
>>> model.save(model_path)
>>> model2 = OneVsRestModel.load(model_path)
>>> model2.transform(test0).head().prediction
0.0
```

버전 2.0.0의 새 기능입니다.

copy (extra=None)

임의로 생성된 uid와 일부 추가 매개 변수를 사용하여 이 인스턴스의 복사본을 만듭니다. 그러면 내장된 paramMap의 구조 자체뿐만 아니라 연결되는 모든 구조를 복제하는 데이터 구조의 복사본이 생성되고 내장된 추가 매개 변수를 복사합니다.

파라미터 extra – 새 인스턴스에 복사할 추가 파라미터 반환

반환 인스턴스 복사본

버전 2.0.0의 새 기능입니다.

setParams (*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *classifier=None*, *weightCol=None*, *parallelism=1*)
setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", classifier=None, weightCol=None, parallelism=1): OneVsRest의 파라미터를 설정합니다.

버전 2.0.0의 새 기능입니다.

class pyspark.ml.classification.**OneVsRestModel** (*models*)

노트: 실험적

OneVsRest에 의해 적합된 모델입니다. 이것은 k개의 이진 분류기를 학습한 결과의 모델들을 각 클래스에 하나씩 저장합니다. 각 예제는 모든 k개의 모델에 대해 점수가 매겨지고 가장 높은 점수를 받은 모델이 라벨 예제에 지정됩니다.

버전 2.0.0의 새 기능입니다.

copy (*extra=None*)

임의로 생성된 uid와 일부 추가 매개 변수를 사용하여 이 인스턴스의 복사본을 만듭니다. 그러면 내장된 paramMap의 구조 자체뿐만 아니라 연결되는 모든 구조를 복제하는 데이터 구조의 복사본이 생성되고 내장된 추가 매개 변수를 복사합니다.

파라미터 **extra** – 새 인스턴스에 복사할 추가 파라미터

반환 인스턴스의 복사본

버전 2.0.0의 새 기능입니다

class pyspark.ml.classification.**RandomForestClassificationModel** (*java_model=None*)
RandomForestClassifier에 의해 적합된 모형

버전 1.4.0의 새 기능입니다.

property featureImportances

각 특징의 중요도를 추정합니다.

각 특징의 중요도는 앙상블에서 모든 트리에 걸친 중요도의 평균입니다. 중요도 벡터는 “1/합계”로 정규화됩니다. 이 방법은 Hastie et al. (Hastie, Tibshirani, Friedman. “통계 학습의 요소, 2판.” 2001.)에 의해 제안되었습니다. 그리고 scikit-learn의 구현을 따릅니다.

참고 항목:

DecisionTreeClassificationModel.featureImportances

버전 2.0.0의 새 기능입니다.

property trees

이들은 null값을 갖는 모 추정치들입니다.

버전 2.0.0의 새 기능입니다.

타입 앙상블에서 트리, 경고

class pyspark.ml.classification.**RandomForestClassifier**(*args, **kwargs)
 분류를 위한 랜덤 포레스트 학습 알고리즘. 이진 및 다중 클래스 라벨과 연속 및 범주형 특징을 모두 지원합니다.

```
>>> import numpy
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], [])), ["label", "features"])
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
>>> si_model = stringIndexer.fit(df)
>>> td = si_model.transform(df)
>>> rf = RandomForestClassifier(numTrees=3, maxDepth=2, labelCol="indexed"
...     , seed=42)
>>> model = rf.fit(td)
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> allclose(model.treeWeights, [1.0, 1.0, 1.0])
True
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> result = model.transform(test0).head()
>>> result.prediction
0.0
>>> numpy.argmax(result.probability)
0
>>> numpy.argmax(result.rawPrediction)
0
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], [
...     "features"])
>>> model.transform(test1).head().prediction
1.0
>>> model.trees
[DecisionTreeClassificationModel (uid=...) of depth...,_
DecisionTreeClassificationModel...]
>>> rfc_path = temp_path + "/rfc"
>>> rf.save(rfc_path)
>>> rf2 = RandomForestClassifier.load(rfc_path)
>>> rf2.getNumTrees()
3
>>> model_path = temp_path + "/rfc_model"
>>> model.save(model_path)
>>> model2 = RandomForestClassificationModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
```

버전 1.4.0의 새 기능입니다.

setFeatureSubsetStrategy(value)

featureSubsetStrategy의 값을 설정합니다.

버전 2.4.0의 새 기능입니다.

```
setParams(self, featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability', rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, seed=None, impurity='gini', numTrees=20, featureSubsetStrategy='auto', subsamplingRate=1.0)
```

선형 분류의 파라미터를 설정합니다.

버전 1.4.0의 새 기능입니다

27.4 군집 분류 API

```
class pyspark.ml.clustering.BisectingKMeans(*args, **kwargs)
```

Steinbach, Karypis, Kumar의 논문 "문서 군집화 기술의 비교"를 바탕으로 한 이등분 k-평균 알고리즘은 스파크에 맞게 수정한 것입니다. 알고리즘은 모든 점을 포함하는 단일 클러스터에서 시작합니다. 총 k leaf 클러스터들이 있거나 나뉠 수 없을 때까지 반복해서 맨 아래 수준에서 나뉠 수 있는 클러스터들을 찾고 k-means을 사용하여 클러스터들 각각을 이등분합니다. 같은 수준의 클러스터를 이등분하는 단계는 병렬성을 높이기 위해 함께 그룹화됩니다. k leaf 군집보다 더 많은 맨 아래 수준에서 모든 분할 가능한 군집을 이등분하면 큰 군집이 높은 우선 순위를 갖습니다.

```
>>> from pyspark.ml.linalg import Vectors
>>> data = [(Vectors.dense([0.0, 0.0])), (Vectors.dense([1.0, 1.0])), ...
...           (Vectors.dense([9.0, 8.0])), (Vectors.dense([8.0, 9.0]))]
>>> df = spark.createDataFrame(data, ["features"])
>>> bkm = BisectingKMeans(k=2, minDivisibleClusterSize=1.0)
>>> model = bkm.fit(df)
>>> centers = model.clusterCenters()
>>> len(centers)
2
>>> model.computeCost(df)
2.000...
>>> model.hasSummary
True
>>> summary = model.summary
>>> summary.k
2
>>> summary.clusterSizes
[2, 2]
>>> transformed = model.transform(df).select("features", "prediction")
>>> rows = transformed.collect()
>>> rows[0].prediction == rows[1].prediction
True
>>> rows[2].prediction == rows[3].prediction
True
>>> bkm_path = temp_path + "/bkm"
>>> bkm.save(bkm_path)
>>> bkm2 = BisectingKMeans.load(bkm_path)
>>> bkm2.getK()
2
>>> bkm2.getDistanceMeasure()
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
'euclidean'
>>> model_path = temp_path + "/bkm_model"
>>> model.save(model_path)
>>> model2 = BisectingKMeansModel.load(model_path)
>>> model2.hasSummary
False
>>> model.clusterCenters() [0] == model2.clusterCenters() [0]
array([ True,  True], dtype=bool)
>>> model.clusterCenters() [1] == model2.clusterCenters() [1]
array([ True,  True], dtype=bool)
```

버전 2.0.0의 새 기능입니다.

getDistanceMeasure()

distanceMeasure 값 혹은 기본값을 갖습니다.

버전 2.0.0의 새 기능입니다.

getK()

k 값 혹은 기본값을 갖습니다.

버전 2.0.0의 새 기능입니다.

getMinDivisibleClusterSize()

minDivisibleClusterSize 값 혹은 기본값을 갖습니다.

버전 2.0.0의 새 기능입니다.

setDistanceMeasure(*value*)

*distanceMeasure*의 값을 설정합니다.

버전 2.4.0의 새 기능입니다.

setK(*value*)

*k*의 값을 설정합니다.

버전 2.0.0의 새 기능입니다.

setMinDivisibleClusterSize(*value*)

*minDivisibleClusterSize*의 값을 설정합니다.

버전 2.0.0의 새 기능입니다.

setParams(*self*, *featuresCol='features'*, *predictionCol='prediction'*, *maxIter=20*,

seed=None, *k=4*, *minDivisibleClusterSize=1.0*, *distanceMeasure='euclidean'*)

BisectingKMeans의 파라미터를 설정합니다.

버전 2.0.0의 새 기능입니다.

class pyspark.ml.clustering.BisectingKMeansModel (*java_model=None*)

BisectingKMeans에 의해 적합된 모형

버전 2.0.0의 새 기능입니다.

clusterCenters()

NumPy 배열 목록으로 표시되는 클러스터 중심을 가져옵니다.

버전 2.0.0의 새 기능입니다.

computeCost (dataset)

입력 점들과 대응되는 클러스터 중심 사이의 거리 제곱의 합을 계산합니다.

버전 2.0.0의 새 기능입니다.

property hasSummary

모델 인스턴스에 대한 훈련 요약이 있는지 여부를 나타냅니다.

버전 2.1.0의 새 기능입니다.

property summary

훈련 셋에서 훈련된 모델의 요약(예: 클러스터 할당, 클러스터 크기)을 가져옵니다.

요약이 없는 경우 예외가 발생합니다.

버전 2.1.0의 새 기능입니다.

class pyspark.ml.clustering.BisectingKMeansSummary (java_obj=None)

노트: 실험적

주어진 모형에서 Bisecting Kmeans 군집 분류 결과

버전 2.1.0의 새 기능입니다.

class pyspark.ml.clustering.DistributedLDAModel (java_model=None)

LDA에 의해 적합된 분산형 모델로, 현재 EM(Expectation-Maximization)에 의해서만 생성되는 모델 타입입니다.

LDA에 의해 적합된 분산형 모델로, 현재 EM(Expectation-Maximization)에 의해서만 생성되는 모델 타입입니다.

버전 2.0.0의 새 기능입니다.

getCheckpointFiles ()

체크포인팅을 사용하고 LDA.keepLastCheckpoint를 true로 설정하면 체크포인트 파일들이 저장될 수 있습니다. 이 방법은 사용자가 해당 파일을 관리할 수 있도록 제공됩니다.

노트: 파티션이 손실되고 특정 *DistributedLDAModel* 방법에 따라 필요한 경우 체크포인트를 제거하면 장애가 발생할 수 있습니다. 참조 계산은 이 모델과 파생 데이터가 범위를 벗어날 때 체크포인트를 지울 것입니다.

:훈련에서 체크포인트 파일 목록을 반환합니다.

버전 2.0.0의 새 기능입니다.

logPrior ()

현재 파라미터 추정치의 로그 확률: $\log P(\text{토픽}, \text{문서의 토픽 분포} | \alpha, \eta)$

버전 2.0.0의 새 기능입니다.

toLocal()

분산 모델을 로컬 표현으로 변환합니다. 이렇게 하면 훈련 데이터 셋에 대한 정보가 삭제됩니다.

경고: 이는 드라이버에게 큰 `topicsMatrix()`를 수집하는 것을 수반합니다.

버전 2.0.0의 새 기능입니다.

trainingLogLikelihood()

현재 파라미터 추정치가 주어진 경우 훈련 셋에서 관측된 토큰들의 로그 우도: $\log P(\text{문서들} | \text{토픽들}, \text{문서들의 토픽 분포들, 디리클레 하이퍼-파라미터})$

노트:

- 여기서는 사전(prior)을 제외하므로 `logPrior()`를 사용합니다.
- `logPrior()`를 사용하더라도 이는 주어진 하이퍼-파라미터에서 데이터 로그 우도와 동일하지 않습니다.
- 이는 훈련 중에 계산된 토픽 분포로부터 계산됩니다. 동일한 훈련 데이터 세트에서 `logLikelihood()`를 호출하면, 토픽 분포가 다시 계산되어 다른 결과를 제공할 수 있습니다.

버전 2.0.0의 새 기능입니다.

class pyspark.ml.clustering.GaussianMixture(*args, **kwargs)

가우시안 혼합 클러스터링. 이 클래스는 다변량 가우시안 혼합 모델(GMMs)에 대한 기대 최대화를 수행합니다. GMM은 하나의 복합체에 대한 각각의 기여를 지정하는 관련 "혼합" 가중치와 함께 독립적인 가우시안 분포들의 복합체 분포를 나타냅니다.

샘플 포인트의 한 집합이 주어지면, 이 클래스는 k개의 가우시안 혼합에 대한 로그 우도를 최대화하여 로그 우도가 `convergenceTol` 미만으로 변화하거나 최대 반복 횟수에 도달할 때까지 반복합니다. 이 프로세스는 일반적으로 수렴이 보장되지만 전역 최적값을 찾는 것은 보장되지 않습니다.

노트: (많은 특징을 가진) 고차원 데이터의 경우, 이 알고리즘은 성능이 떨어질 수 있습니다. 이는 고차원 데이터가 (a) 클러스터링하기 어렵기 때문입니다 (통계/이론적 주장에 기초하여). 그리고 (b) 가우스 분포의 수치 문제 때문입니다.

```
>>> from pyspark.ml.linalg import Vectors
```

```
>>> data = [(Vectors.dense([-0.1, -0.05]),),
...           (Vectors.dense([-0.01, -0.1]),),
...           (Vectors.dense([0.9, 0.8])),,
...           (Vectors.dense([0.75, 0.935])),,
...           (Vectors.dense([-0.83, -0.68])),,
...           (Vectors.dense([-0.91, -0.76]))]
>>> df = spark.createDataFrame(data, ["features"])
>>> gm = GaussianMixture(k=3, tol=0.0001,
...                         maxIter=10, seed=10)
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
>>> model = gm.fit(df)
>>> model.hasSummary
True
>>> summary = model.summary
>>> summary.k
3
>>> summary.clusterSizes
[2, 2, 2]
>>> summary.logLikelihood
8.14636...
>>> weights = model.weights
>>> len(weights)
3
>>> model.gaussiansDF.select("mean").head()
Row(mean=DenseVector([0.825, 0.8675]))
>>> model.gaussiansDF.select("cov").head()
Row(cov=DenseMatrix(2, 2, [0.0056, -0.0051, -0.0051, 0.0046], False))
>>> transformed = model.transform(df).select("features", "prediction")
>>> rows = transformed.collect()
>>> rows[4].prediction == rows[5].prediction
True
>>> rows[2].prediction == rows[3].prediction
True
>>> gmm_path = temp_path + "/gmm"
>>> gm.save(gmm_path)
>>> gm2 = GaussianMixture.load(gmm_path)
>>> gm2.getK()
3
>>> model_path = temp_path + "/gmm_model"
>>> model.save(model_path)
>>> model2 = GaussianMixtureModel.load(model_path)
>>> model2.hasSummary
False
>>> model2.weights == model.weights
True
>>> model2.gaussiansDF.select("mean").head()
Row(mean=DenseVector([0.825, 0.8675]))
>>> model2.gaussiansDF.select("cov").head()
Row(cov=DenseMatrix(2, 2, [0.0056, -0.0051, -0.0051, 0.0046], False))
```

버전 2.0.0의 새 기능입니다.

getK()*k* 값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

setK(*value*)*k*의 값을 설정합니다.

버전 2.0.0의 새 기능입니다.

setParams (*self*, *featuresCol='features'*, *predictionCol='prediction'*, *k=2*, *probabilityCol='probability'*, *tol=0.01*, *maxIter=100*, *seed=None*)

가우시안 혼합 모형을 위한 파라미터를 설정합니다.
버전 2.0.0의 새 기능입니다.

class pyspark.ml.clustering.**GaussianMixtureModel** (*java_model=None*)
가우시안 혼합에 의해 적합된 모형

버전 2.0.0의 새 기능입니다.

property gaussiansDF

가우시안 분포를 데이터 프레임으로 돌이킵니다. 각 행은 가우스 분포를 나타냅니다.
DataFrame에는 평균(벡터)과 cov(행렬)의 두 개의 열이 있습니다.
버전 2.0.0의 새 버전입니다.

property hasSummary

모델 인스턴스에 대한 훈련 요약이 있는지 여부를 나타냅니다.

버전 2.1.0의 새 버전입니다.

property summary

훈련 셋에서 훈련된 모델의 요약(예: 클러스터 할당, 클러스터 크기)을 가져옵니다.
요약이 없는 경우 예외가 발생합니다.
버전 2.1.0의 새 버전입니다.

property weights

혼합에서 각 가우시안 분포에 대한 가중치입니다. k개의 가우시안에 대한 다행 확률 분포이며, 여기서 가중치[i]는 가우시안 i에 대한 가중치이고 가중치의 합은 1입니다. 버전 2.0.0의 새 버전입니다.

class pyspark.ml.clustering.**GaussianMixtureSummary** (*java_obj=None*)

노트: 실험적

주어진 모형에서 가우시안 혼합 군집 분류 결과입니다.
버전 2.1.0의 새 기능입니다.

property logLikelihood

주어진 데이터에서 모형에 대한 총 로그 우도입니다.

버전 2.2.0의 새 기능입니다.

property probability

각 훈련 데이터 포인트에 대한 개별 군집의 확률 데이터 프레임입니다.

버전 2.1.0의 새 기능입니다

property probabilityCol

*predictions*에서 각 군집의 예측 확률의 열 이름입니다.

버전 2.1.0의 새 기능입니다.

class pyspark.ml.clustering.KMeans (*args, **kwargs)
초기화 모드와 같은 k-means++ (Bahmani et al에 의한 k-means 알고리즘)를 동반하는 K-means 군집 분류.

```
>>> from pyspark.ml.linalg import Vectors
>>> data = [(Vectors.dense([0.0, 0.0]),), (Vectors.dense([1.0, 1.0]),),
...           (Vectors.dense([9.0, 8.0]),), (Vectors.dense([8.0, 9.0]),)]
>>> df = spark.createDataFrame(data, ["features"])
>>> kmeans = KMeans(k=2, seed=1)
>>> model = kmeans.fit(df)
>>> centers = model.clusterCenters()
>>> len(centers)
2
>>> model.computeCost(df)
2.000...
>>> transformed = model.transform(df).select("features", "prediction")
>>> rows = transformed.collect()
>>> rows[0].prediction == rows[1].prediction
True
>>> rows[2].prediction == rows[3].prediction
True
>>> model.hasSummary
True
>>> summary = model.summary
>>> summary.k
2
>>> summary.clusterSizes
[2, 2]
>>> summary.trainingCost
2.000...
>>> kmeans_path = temp_path + "/kmeans"
>>> kmeans.save(kmeans_path)
>>> kmeans2 = KMeans.load(kmeans_path)
>>> kmeans2.getK()
2
>>> model_path = temp_path + "/kmeans_model"
>>> model.save(model_path)
>>> model2 = KMeansModel.load(model_path)
>>> model2.hasSummary
False
>>> model.clusterCenters()[0] == model2.clusterCenters()[0]
array([ True,  True], dtype=bool)
>>> model.clusterCenters()[1] == model2.clusterCenters()[1]
array([ True,  True], dtype=bool)
```

버전 1.5.0의 새 기능입니다.

getDistanceMeasure()
distanceMeasure 값을 가져옵니다.

버전 2.4.0의 새 기능입니다.

getInitMode()
initMode 값을 가져옵니다.

버전 1.5.0의 새 기능입니다.

getInitSteps()

initSteps 값을 가져옵니다.

버전 1.5.0의 새 기능입니다.

getK()

k 값을 가져옵니다.

버전 1.5.0의 새 기능입니다.

setDistanceMeasure(value)

distanceMeasure의 값을 설정합니다.

버전 2.4.0의 새 기능입니다.

setInitMode(value)

initMode의 값을 설정합니다.

버전 1.5.0의 새 기능입니다.

setInitSteps(value)

initSteps의 값을 설정합니다.

버전 1.5.0의 새 기능입니다.

setK(value)

k의 값을 설정합니다.

버전 1.5.0의 새 기능입니다.

setParams(self, featuresCol='features', predictionCol='prediction', k=2, initMode='k-means||', initSteps=2, tol=0.0001, maxIter=20, seed=None, distanceMeasure='euclidean')

KMeans의 파라미터를 설정합니다.

버전 1.5.0의 새 기능입니다.

class pyspark.ml.clustering.KMeansModel(java_model=None)

KMeans에 의해 적합된 모형

버전 1.5.0의 새 기능입니다.

clusterCenters()

NumPy 배열의 목록으로 표시되는 군집의 중심을 가져옵니다.

버전 1.5.0의 새 기능입니다.

computeCost(dataset)

주어진 데이터에서 모형에 대한 K-means 비용(가장 가까운 중심에서 점들 간 거리 제곱의 합)을 반환합니다.

..노트:: D:: 버전 2.4.0에서 더 이상 사용되지 않습니다. 버전 3.0.0에서 제거됩니다. 대신 **ClusteringEvaluator**를 사용하십시오.

요약에 포함된 훈련 데이터 셋에서의 비용을 확인할 수도 있습니다.

버전 2.0.0의 새 기능입니다.

property hasSummary

모델 인스턴스에 대한 훈련 요약이 있는지 여부를 나타냅니다.

버전 2.1.0의 새 기능입니다.

property summary

훈련 셋에서 훈련된 모델의 요약(예: 클러스터 할당, 클러스터 크기)을 가져옵니다.

요약이 없는 경우 예외가 발생합니다.

버전 2.1.0의 새 기능입니다.

class pyspark.ml.clustering.LDA(*args, **kwargs)

LDA(Latent Dirichlet Allocation)는 텍스트 문서를 위해 설계된 토픽 모델입니다.

용어:

- “용어” = “단어”: 어휘 요소
- “토큰”: 문서에 나타나는 용어의 인스턴스
- “토픽”: 일부 개념을 나타내는 용어에 대한 다행 분포
- “문서”: 입력 데이터의 하나의 행에 해당하는 텍스트 한 조각

LDA 논문 원본(저널 버전): Blei, Ng, and Jordan. “Latent Dirichlet Allocation.” JMLR, 2003.

입력 데이터 (featuresCol): LDA는 입력 데이터로서 featuresCol 파라미터를 통해 문서 모음을 부여받습니다. 각 문서는 각 입력이 문서에서 대응되는 용어(단어)에 대한 카운트 인 vocaSize 길이 벡터로 명시됩니다. pyspark.ml.feature.Tokenizer 와 pyspark.ml.feature.CountVectorizer와 같은 특정 변환기는 텍스트를 단어 카운트 벡터들로 변환하는데 유용할 수 있습니다.

```
>>> from pyspark.ml.linalg import Vectors, SparseVector
>>> from pyspark.ml.clustering import LDA
>>> df = spark.createDataFrame([[1, Vectors.dense([0.0, 1.0])],
...     [2, SparseVector(2, {0: 1.0})]], ["id", "features"])
>>> lda = LDA(k=2, seed=1, optimizer="em")
>>> model = lda.fit(df)
>>> model.isDistributed()
True
>>> localModel = model.toLocal()
>>> localModel.isDistributed()
False
>>> model.vocabSize()
2
>>> model.describeTopics().show()
+-----+-----+
|topic|termIndices|      termWeights|
+-----+-----+
|    0|[1, 0]| [0.50401530077160...|
|    1|[0, 1]| [0.50401530077160...|
+-----+-----+
...
```

(다음 페이지에 계속)

(○] 전 페이지에서 계속)

```
>>> model.topicsMatrix()
DenseMatrix(2, 2, [0.496, 0.504, 0.504, 0.496], 0)
>>> lda_path = temp_path + "/lda"
>>> lda.save(lda_path)
>>> sameLDA = LDA.load(lda_path)
>>> distributed_model_path = temp_path + "/lda_distributed_model"
>>> model.save(distributed_model_path)
>>> sameModel = DistributedLDAModel.load(distributed_model_path)
>>> local_model_path = temp_path + "/lda_local_model"
>>> localModel.save(local_model_path)
>>> sameLocalModel = LocalLDAModel.load(local_model_path)
```

버전 2.0.0의 새 기능입니다.

getDocConcentration()

docConcentration 값 혹은 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

getK()

k 값 혹은 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

getKeepLastCheckpoint()

keepLastCheckpoint 값 혹은 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

getLearningDecay()

learningDecay 값 혹은 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

getLearningOffset()

learningOffset 값 혹은 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

getOptimizeDocConcentration()

optimizeDocConcentration 값 혹은 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

getOptimizer()

optimizer 값 혹은 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

getSubsamplingRate()

subSamplingRate 값 혹은 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

getTopicConcentration()

topicConcentration 값 혹은 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

getTopicDistributionCol()

topicDistributionCol 값 혹은 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

setDocConcentration (value)

docConcentration의 값을 설정합니다.

```
>>> algo = LDA().setDocConcentration([0.1, 0.2])
>>> algo.getDocConcentration()
[0.1..., 0.2...]
```

버전 2.0.0의 새 기능입니다.

setK (value)

k의 값을 설정합니다.

```
>>> algo = LDA().setK(10)
>>> algo.getK()
10
```

버전 2.0.0의 새 기능입니다.

setKeepLastCheckpoint (value)

keepLastCheckpoint의 값을 설정합니다.

```
>>> algo = LDA().setKeepLastCheckpoint(False)
>>> algo.getKeepLastCheckpoint()
False
```

버전 2.0.0의 새 기능입니다.

setLearningDecay (value)

learningDecay의 값을 설정합니다.

```
>>> algo = LDA().setLearningDecay(0.1)
>>> algo.getLearningDecay()
0.1...
```

버전 2.0.0의 새 기능입니다.

setLearningOffset (value)

learningOffset의 값을 설정합니다.

```
>>> algo = LDA().setLearningOffset(100)
>>> algo.getLearningOffset()
100.0
```

버전 2.0.0의 새 기능입니다.

setOptimizeDocConcentration (value)

optimizeDocConcentration의 값을 설정합니다.

```
>>> algo = LDA().setOptimizeDocConcentration(True)
>>> algo.getOptimizeDocConcentration()
True
```

버전 2.0.0의 새 기능입니다.

setOptimizer (value)

optimizer의 값을 설정합니다. 현재 ‘em’과 ‘online’만 지원합니다.

```
>>> algo = LDA().setOptimizer("em")
>>> algo.getOptimizer()
'em'
```

버전 2.0.0의 새 기능입니다.

setParams (self, featuresCol='features', maxIter=20, seed=None, checkpointInterval=10, k=10, optimizer='online', learningOffset=1024.0, learningDecay=0.51, subsamplingRate=0.05, optimizeDocConcentration=True, docConcentration=None, topicConcentration=None, topicDistributionCol='topicDistribution', keepLastCheckpoint=True)

LDA의 파라미터를 설정합니다.

버전 2.0.0의 새 기능입니다.

setSubsamplingRate (value)

subsamplingRate의 값을 설정합니다.

```
>>> algo = LDA().setSubsamplingRate(0.1)
>>> algo.getSubsamplingRate()
0.1...
```

버전 2.0.0의 새 기능입니다.

setTopicConcentration (value)

topicConcentration의 값을 설정합니다.

```
>>> algo = LDA().setTopicConcentration(0.5)
>>> algo.getTopicConcentration()
0.5...
```

버전 2.0.0의 새 기능입니다.

setTopicDistributionCol (value)

topicDistributionCol의 값을 설정합니다.

```
>>> algo = LDA().setTopicDistributionCol("topicDistributionCol")
>>> algo.getTopicDistributionCol()
'topicDistributionCol'
```

버전 2.0.0의 새 기능입니다.

class pyspark.ml.clustering.LDAModel (java_model=None)

Latent Dirichlet Allocation (LDA) 모델. 이 개념을 통해 로컬 및 분산 데이터 구조를 비롯한 다양한 기본 표현이 가능합니다.

버전 2.0.0의 새 기능입니다.

describeTopics (maxTermsPerTopic=10)

상위 가중치 용어로 설명된 토픽을 반환합니다.

버전 2.0.0의 새 기능입니다.

estimatedDocConcentration ()

데이터로부터 추정된 LDA.docConcentration에 대한 값. 온라인 LDA가 사용되거나 LDA.optimizeDocConcentration이 false로 설정된 경우, LDA.docConcentration 파라미터에 대한 고정(주어진) 값을 반환합니다.

버전 2.0.0의 새 기능입니다.

isDistributed()

인스턴스가 분산 LDA 모델 유형인지 여부를 나타냅니다.

버전 2.0.0의 새 기능입니다.

logLikelihood (dataset)

전체 코퍼스(corporus)의 로그 우도에 대한 하한값을 계산합니다. 온라인 LDA 논문의 식 (16) 참조 (Hoffman et al., 2010).

경고: 모델이 [DistributedLDAModel](#)의 인스턴스인 경우(optimizer가 "em"으로 설정됐을 때 생성된), 드라이버에 대한 대규모 [topicsMatrix \(\)](#) 를 수집하는 작업이 포함됩니다. 이 구현은 향후 변경될 수 있습니다.

버전 2.0.0의 새 기능입니다.

logPerplexity (dataset)

perplexity에 대한 상한을 계산합니다. (낮은 것이 더 좋습니다) 온라인 LDA 논문의 식 (16)을 참조하십시오(Hoffman et al., 2010).

경고: 모델이 [DistributedLDAModel](#)의 인스턴스인 경우(optimizer가 “em”으로 설정됐을 때 생성된), 드라이버에 대한 대규모 [topicsMatrix \(\)](#) 를 수집하는 작업이 포함됩니다. 이 구현은 향후 변경될 수 있습니다.

버전 2.0.0의 새 기능입니다.

topicsMatrix ()

추론된 토픽들, 여기서 각 토픽은 용어에 대한 분포로 표시됩니다. 이것은 각 열이 토픽 인 크기 vocabSize x k의 행렬입니다. 토픽의 순서에 대한 보장은 없습니다.

경고: 모델이 실제로 Expectation-Maximization (“em”) optimizer에 의해 생성된 [DistributedLDAModel](#) 모델 인스턴스인 경우, 이 방법에는 vocabSize x k 순서로 드라이버에 대량의 데이터를 수집하는 것이 포함될 수 있습니다.

버전 2.0.0의 새 기능입니다.

vocabSize ()

어휘 크기(어휘 안의 단어들 혹은 용어들의 개수)

버전 2.0.0의 새 기능입니다.

```
class pyspark.ml.clustering.LocalLDAModel (java_model=None)
    LDA에 의해 적합된 로컬(비분산) 모델. 이 모델은 추론된 토픽들만 저장하며, 훈련 데이터
    셋에 대한 정보는 저장하지 않습니다.
    버전 2.0.0의 새 기능입니다.
```

```
class pyspark.ml.clustering.PowerIterationClustering(*args, **kwargs)
```

노트: 실험적

Lin과 Cohen이 개발한 확장 가능한 그래프 클러스터링 알고리즘인 PIC(Power Eatition Clustering). 초록에서: PIC는 데이터의 정규화된 쌍별 유사성 행렬에서 truncated power 반복을 사용하여 데이터 세트의 매우 낮은 차원 임베딩을 찾습니다.

이 클래스는 아직 추정량/트랜스포머가 아닙니다. PowerIterationClustering 알고리즘을 실행하기 위해 *assignClusters()* 메서드를 사용합니다.

참고 항목:

스펙트럼 군집화에 관한 위키피디아

```
>>> data = [(1, 0, 0.5),
...           (2, 0, 0.5), (2, 1, 0.7),
...           (3, 0, 0.5), (3, 1, 0.7), (3, 2, 0.9),
...           (4, 0, 0.5), (4, 1, 0.7), (4, 2, 0.9), (4, 3, 1.1),
...           (5, 0, 0.5), (5, 1, 0.7), (5, 2, 0.9), (5, 3, 1.1), (5, 4, 1.
...           ↪3)]
>>> df = spark.createDataFrame(data).toDF("src", "dst", "weight")
>>> pic = PowerIterationClustering(k=2, maxIter=40, weightCol="weight")
>>> assignments = pic.assignClusters(df)
>>> assignments.sort(assignments.id).show(truncate=False)
+---+-----+
|id |cluster|
+---+-----+
|0  |1      |
|1  |1      |
|2  |1      |
|3  |1      |
|4  |1      |
|5  |0      |
+---+-----+
...
>>> pic_path = temp_path + "/pic"
>>> pic.save(pic_path)
>>> pic2 = PowerIterationClustering.load(pic_path)
>>> pic2.getK()
2
>>> pic2.getMaxIter()
40
```

버전 2.4.0의 새 기능입니다.

assignClusters (dataset)

PIC 알고리즘을 실행하고 각 입력 버텍스에 대한 클러스터 할당을 반환합니다.

파라미터 **dataset** – PIC 논문의 행렬 A인 유사도 행렬를 나타내는 src, dst, weight 열이 있는 데이터 셋입니다. src 열의 값은 i, dst 열의 값은 j, t가 중치 열 값은 음이 아닌 유사도 s_{ij} 입니다. 이는 대칭행렬이므로 $s_{ij} = s_{ji}, 0 \leq s_{ij} \leq 1$ 아닌 유사성을 가진 임의의 (i, j) 에 대하여, 입력에 (i, j, s_{ij}) 혹은 (j, i, s_{ji}) 가 있어야 합니다. $i = j$ 을 가정하므로, $s_{ii} = 1.0$ 인 행은 무시됩니다.

반환 버텍스 ID의 열과 ID에 해당하는 클러스터를 포함하는 데이터 집합을 반환합니다. 스키마는 다음과 같습니다: - id: Long - cluster: Int

버전 2.4.0의 새 기능입니다.

getDstCol ()

dstCol 값 혹은 기본값을 가져옵니다.

버전 2.4.0의 새 기능입니다.

getInitMode ()

initMode 값 혹은 기본값을 가져옵니다.

버전 2.4.0의 새 기능입니다.

getK ()

k 값 혹은 기본값을 가져옵니다.

버전 2.4.0의 새 기능입니다.

getSrcCol ()

srcCol 값 혹은 기본값을 가져옵니다.

버전 2.4.0의 새 기능입니다.

setDstCol (value)

dstCol의 값을 설정합니다.

버전 2.4.0의 새 기능입니다.

setInitMode (value)

initMode의 값을 설정합니다.

버전 2.4.0의 새 기능입니다.

setK (value)

k의 값을 설정합니다.

버전 2.4.0의 새 기능입니다.

setParams (self, k=2, maxIter=20, initMode='random', srcCol='src', dstCol='dst', weightCol=None)

PowerIterationClustering의 파라미터를 설정합니다.

버전 2.4.0의 새 기능입니다.

setSrcCol (value)
srcCol의 값을 설정합니다.
버전 2.4.0의 새 버전입니다.

27.5 추천 API

class pyspark.ml.recommendation.**ALS** (*args, **kwargs)

교대 최소 제곱 행렬 인수 분해 (Alternating Least Squares, ALS).

ALS 는 등급 행렬 R 를 두 하위 등급 행렬인 X 와 Y 의 곱으로 추정하려고 합니다. 즉, $X * Y^T = R$. 일반적으로 이러한 근사치를 '인자' 행렬이라고 합니다. 일반적인 접근 방식은 반복입니다. 각 반복 동안, 인자 행렬 둘 하나는 상수로 유지되고 다른 하나는 최소 제곱을 사용하기 위해 풀이됩니다. 새로 풀이된 인자 행렬은 다른 인자 행렬에 대해 풀이하는 동안 상수로 유지됩니다.

이것은 두 개의 요소 집합("사용자" 및 "제품"이라고 함)을 블록으로 그룹화하는 ALS 인수분해 알고리즘의 블록화된 구현입니다. 그리고 각 반복에서 각 사용자 벡터의 복사본을 각 제품 블록에 하나씩만 그리고 해당 사용자의 특징 벡터가 필요한 제품 블록에 대해서만 전송함으로써 커뮤니케이션을 줄입니다. 이는 각 사용자(제품 블록에 기여할)의 "아웃링크"와 각 제품(각 사용자 블록으로부터 받을 특정 벡터에 의존할)의 "인링크" 정보를 결정하기 위해 등급 매트릭스에 대한 일부 정보를 미리 계산함으로써 달성됩니다.

이를 통해 각 사용자 블록과 제품 블록 간의 특징 벡터 배열만 전송할 수 있으며, 제품 블록에서 사용자의 등급을 찾아 이러한 메시지를 기반으로 제품을 업데이트할 수 있습니다.

암시적 선호 데이터의 경우, 사용되는 알고리즘은 여기서 사용되는 블록화된 접근 방식에 맞게 조정된 “암시적 피드백 데이터셋을 위한 협업 필터링”을 기반으로 합니다.

기본적으로 등급 행렬 R 에 대한 낮은 등급 근사치를 찾는 대신 평점 $r > 0$ 이면 선호 행렬 P 의 원소가 1, $r \leq 0$ 이면 0인 선호 행렬 P 에 대한 근사치를 찾습니다. 그런 다음 등급은 아이템에 부여된 명시적인 등급이 아닌 표시된 사용자 선호도의 강도와 관련된 '신뢰도' 값으로 작용합니다.

```
>>> df = spark.createDataFrame(
...     [(0, 0, 4.0), (0, 1, 2.0), (1, 1, 3.0), (1, 2, 4.0), (2, 1, 1.0),
...     (2, 2, 5.0)],
...     ["user", "item", "rating"])
>>> als = ALS(rank=10, maxIter=5, seed=0)
>>> model = als.fit(df)
>>> model.rank
10
>>> model.userFactors.orderBy("id").collect()
[Row(id=0, features=[...]), Row(id=1, ...), Row(id=2, ...)]
>>> test = spark.createDataFrame([(0, 2), (1, 0), (2, 0)], ["user", "item"])
>>> predictions = sorted(model.transform(test).collect(), key=lambda r:r[0])
>>> predictions[0]
Row(user=0, item=2, prediction=-0.13807615637779236)
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
>>> predictions[1]
Row(user=1, item=0, prediction=2.6258413791656494)
>>> predictions[2]
Row(user=2, item=0, prediction=-1.5018409490585327)
>>> user_recs = model.recommendForAllUsers(3)
>>> user_recs.where(user_recs.user == 0)           .select("recommendations.
˓→item", "recommendations.rating").collect()
[Row(item=[0, 1, 2], rating=[3.910..., 1.992..., -0.138...])]
>>> item_recs = model.recommendForAllItems(3)
>>> item_recs.where(item_recs.item == 2)           .select("recommendations.
˓→user", "recommendations.rating").collect()
[Row(user=[2, 1, 0], rating=[4.901..., 3.981..., -0.138...])]
>>> user_subset = df.where(df.user == 2)
>>> user_subset_recs = model.recommendForUserSubset(user_subset, 3)
>>> user_subset_recs.select("recommendations.item", "recommendations.
˓→rating").first()
Row(item=[2, 1, 0], rating=[4.901..., 1.056..., -1.501...])
>>> item_subset = df.where(df.item == 0)
>>> item_subset_recs = model.recommendForItemSubset(item_subset, 3)
>>> item_subset_recs.select("recommendations.user", "recommendations.
˓→rating").first()
Row(user=[0, 1, 2], rating=[3.910..., 2.625..., -1.501...])
>>> als_path = temp_path + "/als"
>>> als.save(als_path)
>>> als2 = ALS.load(als_path)
>>> als.getMaxIter()
5
>>> model_path = temp_path + "/als_model"
>>> model.save(model_path)
>>> model2 = ALSModel.load(model_path)
>>> model.rank == model2.rank
True
>>> sorted(model.userFactors.collect()) == sorted(model2.userFactors.
˓→collect())
True
>>> sorted(model.itemFactors.collect()) == sorted(model2.itemFactors.
˓→collect())
True
```

버전 1.4.0의 새 기능입니다.

getAlpha()

알파 값 혹은 기본값을 가져옵니다.

버전 1.4.0의 새 기능입니다.

getColdStartStrategy()

coldStartStrategy 값 혹은 기본값을 가져옵니다.

버전 2.2.0의 새 기능입니다.

getFinalStorageLevel()

finalStorageLevel 값 혹은 기본값을 가져옵니다.

버전 2.2.0의 새 기능입니다.

getImplicitPrefs()

implicitPrefs (잠재된 선호도) 값 혹은 기본값을 가져옵니다.

버전 1.4.0의 새 기능입니다.

getIntermediateStorageLevel()

intermediateStorageLevel 값 혹은 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

getItemCol()

itemCol 값 혹은 기본값을 가져옵니다.

버전 1.4.0의 새 기능입니다.

getNonnegative()

nonnegative(0보다 큰) 값 혹은 기본값을 가져옵니다.

버전 1.4.0의 새 기능입니다.

getNumItemBlocks()

numItemBlocks 값 혹은 기본값을 가져옵니다.

버전 1.4.0의 새 기능입니다.

getNumUserBlocks()

numUserBlocks 값 혹은 기본값을 가져옵니다.

버전 1.4.0의 새 기능입니다.

getRank()

순위 값 혹은 기본값을 가져옵니다.

버전 1.4.0의 새 기능입니다.

getRatingCol()

ratingCol 값 혹은 기본값을 가져옵니다.

버전 1.4.0의 새 기능입니다.

getUserCol()

userCol 값 혹은 기본값을 가져옵니다.

버전 1.4.0의 새 기능입니다.

setAlpha(value)

alpha의 값을 설정합니다.

버전 1.4.0의 새 기능입니다.

setColdStartStrategy(value)

coldStartStrategy의 값을 설정합니다.

버전 2.2.0의 새 기능입니다.

setFinalStorageLevel(value)

finalStorageLevel의 값을 설정합니다.

버전 2.2.0의 새 기능입니다.

setImplicitPrefs (value)

implicitPrefs의 값을 설정합니다.

버전 1.4.0의 새 기능입니다.

setIntermediateStorageLevel (value)

intermediateStorageLevel의 값을 설정합니다.

버전 2.0.0의 새 기능입니다.

setItemCol (value)

itemCol의 값을 설정합니다.

버전 1.4.0의 새 기능입니다.

setNonnegative (value)

nonnegative의 값을 설정합니다.

버전 1.4.0의 새 기능입니다.

setNumBlocks (value)

numUserBlocks 와 numItemBlocks 모두 특정 값으로 설정합니다.

버전 1.4.0의 새 기능입니다.

setNumItemBlocks (value)

numItemBlocks의 값을 설정합니다.

버전 1.4.0의 새 기능입니다.

setNumUserBlocks (value)

numUserBlocks의 값을 설정합니다.

버전 1.4.0의 새 기능입니다.

setParams (self, rank=10, maxIter=10, regParam=0.1, numUserBlocks=10, numItemBlocks=10, implicitPrefs=False, alpha=1.0, userCol='user', itemCol='item', seed=None, ratingCol='rating', nonnegative=False, checkpointInterval=10, intermediateStorageLevel='MEMORY_AND_DISK', finalStorageLevel='MEMORY_AND_DISK', coldStartStrategy='nan')

ALS의 파라미터를 설정합니다.

버전 1.4.0의 새 기능입니다.

setRank (value)

rank의 값을 설정합니다.

버전 1.4.0의 새 기능입니다.

setRatingCol (value)

ratingCol의 값을 설정합니다.

버전 1.4.0의 새 기능입니다.

setUserCol (value)

userCol의 값을 설정합니다.

버전 1.4.0의 새 기능입니다.

class pyspark.ml.recommendation.**ALSModel** (*java_model=None*)
ALS에 의해 적합된 모형.

버전 1.4.0의 새 기능입니다.

property itemFactors
id 와 *features*

버전 1.4.0의 새 기능입니다.

타입 아이템 요인들을 두 개의 열로 저장한 데이터 프레임

property rank
행렬 인수분해 모델의 순위

버전 1.4.0의 새 기능입니다.

recommendForAllItems (*numUsers*)

모든 아이템에 대해 각 아이템별 추천되는 상위 *numUsers* 사용자를 반환합니다.

파라미터 **numUsers** – 각 아이템에 대한 최대 추천 수

반환 (*itemCol*, *recommendations*(추천들)) 열을 갖는 데이터 프레임으로
*recommendations*은 (*userCol*, *rating*) 행들의 배열로 저장됩니다.

버전 2.2.0의 새 기능입니다.

recommendForAllUsers (*numItems*)

모든 사용자들에 대해 사용자별 추천되는 상위 *numItems* 아이템을 반환합니다.

파라미터 **numItems** – 각 사용자에 대한 최대 추천 수

반환 (*userCol*, *recommendations*) 열을 갖는 데이터 프레임으로
*recommendations*은 (*itemCol*, *rating*) 행들의 배열로 저장됩니다.

버전 2.2.0의 새 기능입니다.

recommendForItemSubset (*dataset*, *numUsers*)

입력 데이터 세트의 각 아이템 ID에 대해 추천된 상위 *numUsers* 사용자를 반환합니다.

입력 데이터 세트에 중복된 ID가 있는 경우 고유 ID당 추천 한 세트만 반환됩니다.

파라미터

- **dataset** – 아이템 ids 열을 포함하는 데이터 셋. 열 이름이 *itemCol*과 일치해야 합니다.
- **numUsers** – 각 아이템의 최대 추천 수

반환 결과 (*itemCol*, *recommendations*) 열을 갖는 데이터 프레임으로
*recommendations*은 (*userCol*, *rating*) 행들의 배열로 저장됩니다.

버전 2.3.0의 새 기능입니다.

recommendForUserSubset (*dataset*, *numItems*)

입력 데이터 세트의 각 사용자 ID에 대해 추천된 상위 *numItems* 사용자를 반환합니다.

입력 데이터 세트에 중복된 ID가 있는 경우 고유 ID당 추천 한 세트만 반환됩니다.

파라미터

- **dataset** – 사용자 ID들의 열을 포함하는 데이터 셋. 열 이름이 `userCol` 와 일치해야 합니다.
- **numItems** – 각 사용자의 최대 추천 수

반환 (`userCol, recommendations`) 열을 갖는 데이터 프레임으로 `recommendations`은 (`itemCol, rating`) 행들의 배열로 저장됩니다.

버전 2.3.0의 새 기능입니다.

property userFactors

`id` 와 `features`

버전 1.4.0의 새 기능입니다.

타입 사용자 요인들을 두 개의 열로 저장한 데이터 프레임

27.6 파이프라인 API

class pyspark.ml.pipeline.Pipeline(*args, **kwargs)

단순한 파이프라인으로, 추정량 역할을 합니다. 파이프라인은 각각이 추정량 또는 트랜스포머인 스테이지들의 일련으로 구성됩니다. `Pipeline.fit()` 이 호출되면 스테이지는 순서대로 실행됩니다. 스테이지가 추정량인 경우, 모델을 적합시키기 위해 입력 데이터 셋에 `Estimator.fit()` 메서드가 호출됩니다. 그런 다음, 트랜스포머인 모델은 다음 스테이지의 입력으로 데이터셋을 변환하는 데 사용됩니다. 스테이지가 트랜스포머인 경우, 트랜스포머입니다. 다음 스테이지의 데이터 셋을 생성하기 위해 `transform()` 메서드를 호출합니다. `Pipeline`의 적합 모델은 파이프라인 단계에 해당하는 적합 모델과 트랜스포머로 구성된 `PipelineModel`입니다. 스테이지가 빈 목록인 경우 파이프라인은 `identity` 트랜스포머 역할을 합니다.

버전 1.3.0의 새 기능입니다.

copy(extra=None)

인스턴스의 복사본을 생성합니다.

파라미터 **extra** – 추가 파라미터

반환 신규 인스턴스

버전 1.4.0의 새 기능입니다.

getStages()

파이프라인 스테이지를 얻습니다.

버전 1.3.0의 새 기능입니다.

classmethod read()

해당 클래스의 MLReader 인스턴스를 반환합니다.

버전 2.0.0의 새 기능입니다.

setParams (self, stages=None)
파이프라인을 위한 파라미터들을 설정합니다.

버전 1.3.0의 새 기능입니다.

setStages (value)
파이프라인 스테이지를 설정합니다.

파라미터 **value** – 트랜스포머 혹은 추정량의 목록

반환 파이프라인 인스턴스

버전 1.3.0의 새 기능입니다.

write()
해당 ML 인스턴스를 위한 MLWriter 인스턴스를 반환합니다.

버전 2.0.0의 새 기능입니다.

class pyspark.ml.pipeline.PipelineModel (stages)
트랜스포머 및 적합 모형들과 함께 컴파일된 파이프라인을 나타냅니다.

버전 1.3.0의 새 기능입니다.

copy (extra=None)
인스턴스의 복사본을 생성합니다.

파라미터 **extra** – 추가 파라미터

반환 신규 인스턴스

버전 1.4.0의 새 기능입니다.

classmethod read()
해당 클래스를 위한 MLReader 인스턴스를 반환합니다.

버전 2.0.0의 새 기능입니다.

write()
해당 ML 인스턴스를 위한 MLWriter 인스턴스를 반환합니다.

버전 2.0.0의 새 기능입니다.

class pyspark.ml.pipeline.PipelineModelReader (cls)
(비공식) *PipelineModel* 타입의 MLReader 특수화

types load (path)
입력 경로의 ML 인스턴스를 로드합니다.

class pyspark.ml.pipeline.PipelineModelWriter (instance)
(비공식) *PipelineModel* 타입의 MLWriter 특수화

saveImpl (path)
save() 는 덮어쓰기를 처리한 다음 메서드를 호출합니다. 하위 클래스는 인스턴스의 실제 저장을 구현하려면 메서드를 중단시켜야 합니다.

class pyspark.ml.pipeline.PipelineReader (cls)
(비공식) *Pipeline* 타입의 MLReader 특수화

load(path)

입력 경로경로로부터 ML 인스턴스 로드합니다.

```
class pyspark.ml.pipeline.PipelineSharedReadWrite
```

노트: DeveloperApi

Pipeline 와 *PipelineModel*에서 공유하는 MLReader 및 MLWriter를 위한 함수 버전 2.3.0의 새 기능입니다.

static getStagePath(stageUid, stageIdx, numStages, stagesDir)

지정된 스테이지를 저장하기 위한 경로 가져옵니다.

static load(metadata, sc, path)

Pipeline 혹은 *PipelineModel*을 위한 메타데이터와 스테이지들을 로드합니다.

반환 (UID, list of stages)

static saveImpl(instance, stages, sc, path)

Pipeline 혹은 *PipelineModel*의 메타데이터와 스테이지들을 저장합니다. - 메타데이터를 path/metadata에 저장합니다. - stages를 stages/IDX_UID에 저장합니다.

static validateStages(stages)

모든 단계가 쓰기 가능한지 (Writable) 검사합니다.

```
class pyspark.ml.pipeline.PipelineWriter(instance)
```

(비공식) *Pipeline* 타입의 MLWriter 특수화

saveImpl(path)

save()는 덮어쓰기를 처리한 다음 메서드를 호출합니다. 하위 클래스는 인스턴스의 실제 저장을 구현하려면 메서드를 중단시켜야 합니다.

27.7 튜닝 API

```
class pyspark.ml.tuning.CrossValidator(*args, **kwargs)
```

K-폴드 교차 검증은 데이터 셋을 별도의 훈련 및 테스트 데이터 셋들로 사용되는 비-중복 무작위 분할 폴드 세트로 분할하여 모델 선택을 수행합니다. 예를 들어, k=3 폴드의 경우, K-폴드 교차 검증은 3개의 (훈련, 테스트) 데이터 세트 쌍을 생성하며, 각 데이터는 2/3의 데이터를 학습에 사용하고 1/3은 테스트에 사용합니다. 각 폴드는 테스트 세트로 정확히 한 번 사용됩니다.

```
>>> from pyspark.ml.classification import LogisticRegression
>>> from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>> from pyspark.ml.linalg import Vectors
>>> dataset = spark.createDataFrame(
...     [ (Vectors.dense([0.0]), 0.0),
...      (Vectors.dense([0.4]), 1.0),
...      (Vectors.dense([0.5]), 0.0),
...      (Vectors.dense([0.6]), 1.0),
```

(다음 페이지에 계속)

(continued from previous page)

```

...      (Vectors.dense([1.0]), 1.0)] * 10,
...      ["features", "label"])
>>> lr = LogisticRegression()
>>> grid = ParamGridBuilder().addGrid(lr.maxIter, [0, 1]).build()
>>> evaluator = BinaryClassificationEvaluator()
>>> cv = CrossValidator(estimator=lr, estimatorParamMaps=grid,_
    ↪evaluator=evaluator,
...      parallelism=2)
>>> cvModel = cv.fit(dataset)
>>> cvModel.avgMetrics[0]
0.5
>>> evaluator.evaluate(cvModel.transform(dataset))
0.8333...

```

버전 1.4.0의 새 기능입니다.

copy (extra=None)

랜덤하게 생성된 uid와 일부 추가 파라미터를 사용하여 이 인스턴스의 복사본을 만듭니다. 이 복사본들은 내장된 paramMap의 깊은 복사를 생성하고 내장된 추가 파라미터들을 복사합니다.

파라미터 **extra** – 신규 인스턴스를 복사하기 위한 추가 파라미터

반환 인스턴스 복사본

버전 1.4.0의 새 기능입니다.

getNumFolds ()

numFolds 값 혹은 기본값을 가져옵니다.

버전 1.4.0의 새 기능입니다.

classmethod read()

해당 클래스의 MLReader 인스턴스를 반환합니다.

버전 2.3.0의 새 기능입니다.

setNumFolds (value)

numFolds의 값을 설정합니다.

버전 1.4.0의 새 기능입니다.

setParams (estimator=None, estimatorParamMaps=None, evaluator=None, numFolds=3,

seed=None, parallelism=1, collectSubModels=False)

setParams(self, estimator=None, estimatorParamMaps=None, evaluator=None, numFolds=3, seed=None, parallelism=1, collectSubModels=False): 교차 검증자의 파라미터를 설정합니다.

버전 1.4.0의 새 기능입니다.

write ()

ML 인스턴스를 위한 MLWriter 인스턴스를 반환합니다.

버전 2.3.0의 새 기능입니다.

class pyspark.ml.tuning.**CrossValidatorModel** (*bestModel*, *avgMetrics*=[], *subModels*=None)

CrossValidatorModel 은 폴드 간 평균 교차 검증 메트릭이 가장 높은 모델을 포함하고 있으며 해당 모델을 사용하여 입력 데이터를 변환합니다. **CrossValidatorModel**은 평가된 각 파라미터 맵에 대한 메트릭도 추적합니다.

버전 1.4.0의 새로운 기능입니다.

avgMetrics

CrossValidator.estimatorParamMaps의 각 *paramMap*에 대한 해당 순서의 평균 교차 검증 메트릭입니다.

bestModel

교차 검증을 통한 최적 모형

copy (*extra*=None)

임의로 생성된 uid와 일부 추가 파라미터를 사용하여 인스턴스의 복사본을 만듭니다. 이렇게 하면 기본 *bestModel*이 복사되고 내장된 *paramMap*의 깊은 복사본이 생성되며 내장된 파라미터와 추가 파라미터가 복사됩니다. 추가 파라미터를 *subModels*로 복사하지 않습니다.

파라미터 **extra** – 신규 인스턴스를 복사하기 위한 추가 파라미터

반환 인스턴스 복사본

버전 1.4.0의 새 기능입니다.

classmethod read()

해당 클래스의 MLReader 인스턴스를 반환합니다.

버전 2.3.0의 새 기능입니다.

subModels

교차 검증의 하위 모델 목록

write()

ML 인스턴스를 위한 MLWriter 인스턴스를 반환합니다.

버전 2.3.0의 새 기능입니다.

class pyspark.ml.tuning.**ParamGridBuilder**

grid search 기반 모델 선택에 사용되는 파라미터 그리드를 위한 Builder

```
>>> from pyspark.ml.classification import LogisticRegression
>>> lr = LogisticRegression()
>>> output = ParamGridBuilder() \
...     .baseOn({lr.labelCol: 'l'}) \
...     .baseOn([lr.predictionCol, 'p']) \
...     .addGrid(lr.regParam, [1.0, 2.0]) \
...     .addGrid(lr.maxIter, [1, 5]) \
...     .build()
>>> expected = [
...     {lr.regParam: 1.0, lr.maxIter: 1, lr.labelCol: 'l', lr.
...     predictionCol: 'p'},
...     {lr.regParam: 2.0, lr.maxIter: 1, lr.labelCol: 'l', lr.
...     predictionCol: 'p'},
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```

...     {lr.regParam: 1.0, lr.maxIter: 5, lr.labelCol: 'l', lr.
→predictionCol: 'p'},
...     {lr.regParam: 2.0, lr.maxIter: 5, lr.labelCol: 'l', lr.
→predictionCol: 'p'}]
>>> len(output) == len(expected)
True
>>> all([m in expected for m in output])
True

```

버전 1.4.0의 새 기능입니다.

addGrid(param, values)

그리드의 지정된 파라미터들을 고정된 값들로 설정합니다.

버전 1.4.0의 새 기능입니다.

baseOn(*args)

그리드의 지정된 파라미터들을 고정된 값들로 설정합니다. 파라미터 사전 (dictionary) 또는 (파라미터, 값) 쌍의 목록을 허용합니다.

버전 1.4.0의 새 기능입니다.

build()

파라미터 그리드에서 지정한 파라미터들의 모든 조합들을 빌드하고 반환합니다.

버전 1.4.0의 새 기능입니다.

class pyspark.ml.tuning.TrainValidationSplit(*args, **kwargs)

노트: 실험적

하이퍼 파라미터 튜닝을 위한 검증. 입력 데이터셋을 훈련과 검증 셋들로 랜덤하게 분할하고 검증 셋에 대한 평가 메트릭을 사용하여 최상의 모델을 선택합니다. [CrossValidator](#), 와 유사하지만 데이터 세트를 한 번만 분할합니다.

```

>>> from pyspark.ml.classification import LogisticRegression
>>> from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>> from pyspark.ml.linalg import Vectors
>>> dataset = spark.createDataFrame(
...     [ (Vectors.dense([0.0]), 0.0),
...      (Vectors.dense([0.4]), 1.0),
...      (Vectors.dense([0.5]), 0.0),
...      (Vectors.dense([0.6]), 1.0),
...      (Vectors.dense([1.0]), 1.0) ] * 10,
...     ["features", "label"])
>>> lr = LogisticRegression()
>>> grid = ParamGridBuilder().addGrid(lr.maxIter, [0, 1]).build()
>>> evaluator = BinaryClassificationEvaluator()
>>> tvs = TrainValidationSplit(estimator=lr, estimatorParamMaps=grid,
→evaluator=evaluator,

```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
...     parallelism=2)
>>> tvsModel = tvs.fit(dataset)
>>> evaluator.evaluate(tvsModel.transform(dataset))
0.8333...
```

버전 2.0.0의 새 기능입니다.

copy (extra=None)

랜덤하게 생성된 uid와 일부 추가 파라미터를 사용하여 이 인스턴스의 복사본을 만듭니다. 이 복사본들은 내장된 paramMap의 깊은 복사를 생성하고 내장된 추가 파라미터들을 복사합니다.

파라미터 **extra** – 신규 인스턴스를 복사하기 위한 추가 파라미터

반환 인스턴스 복사본

버전 2.0.0의 새 기능입니다.

getTrainRatio()

trainRatio(훈련 비율) 값 혹은 기본값을 가져옵니다.

버전 2.0.0의 새 기능입니다.

classmethod read()

해당 클래스의 MLReader 인스턴스를 반환합니다.

버전 2.3.0의 새 기능입니다.

setParams (estimator=None, estimatorParamMaps=None, evaluator=None, trainRatio=0.75, parallelism=1, collectSubModels=False, seed=None)

setParams(self, estimator=None, estimatorParamMaps=None, evaluator=None, trainRatio=0.75, parallelism=1, collectSubModels=False, seed=None): 훈련과 검증 데이터 셋 분할을 위한 파라미터를 설정합니다.

버전 2.0.0의 새 기능입니다.

setTrainRatio (value)

trainRatio의 값을 설정합니다.

버전 2.0.0의 새 기능입니다.

write()

해당 ML 인스턴스의 MLWriter 인스턴스를 반환합니다.

버전 2.3.0의 새 기능입니다.

class pyspark.ml.tuning.TrainValidationSplitModel (bestModel, validationMetrics=[], subModels=None)

노트: 실험적

훈련과 검증 셋 분할을 위한 모델

버전 2.0.0의 새 기능입니다.

bestModel

훈련과 검증 셋 분할을 통한 최적 모델

copy(extra=None)

임의로 생성된 uid와 일부 추가 파라미터를 사용하여 이 인스턴스의 복사본을 만듭니다. 기본 bestModel을 복사하고 내장된 paramMap의 깊은 복사를 생성하고 내장된 추가 파라미터들을 복사합니다. 그리고 검증 메트릭스의 얕은 복사를 생성합니다. 추가 파라미터를 subModels에 복사하지 않습니다.

파라미터 **extra** – 신규 인스턴스를 복사하기 위한 추가 파라미터

반환 인스턴스 복사본

버전 2.0.0의 새 기능입니다.

classmethod read()

해당 클래스의 MLReader 인스턴스를 반환합니다.

버전 2.3.0의 새 기능입니다.

subModels

훈련과 검증 셋 분할로 인한 서브 모델들

validationMetrics

평가된 검증 메트릭스

write()

ML 인스턴스에 대한 MLWriter 인스턴스를 반환합니다.

버전 2.3.0의 새 기능입니다.

27.8 평가 API

```
class pyspark.ml.evaluation.BinaryClassificationEvaluator(*args,
                                                       **kwargs)
```

노트: 실험적

두 개의 입력 열(원시 예측 및 라벨)을 기대하는 이진 분류를 위한 평가자입니다. 원시 예측 열은 double 형식(이진 0/1 예측 또는 라벨 1의 확률) 또는 유형 벡터(원시 예측, 점수 또는 라벨 확률의 길이-2 벡터)일 수 있습니다.

```
>>> from pyspark.ml.linalg import Vectors
>>> scoreAndLabels = map(lambda x: (Vectors.dense([1.0 - x[0], x[0]]), x[1]),
...                         [(0.1, 0.0), (0.1, 1.0), (0.4, 0.0), (0.6, 0.0), (0.6, 1.0), (0.6,
...                         1.0), (0.8, 1.0)])
>>> dataset = spark.createDataFrame(scoreAndLabels, ["raw", "label"])
... 
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
>>> evaluator = BinaryClassificationEvaluator(rawPredictionCol="raw")
>>> evaluator.evaluate(dataset)
0.70...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "areaUnderPR"})
0.83...
>>> bce_path = temp_path + "/bce"
>>> evaluator.save(bce_path)
>>> evaluator2 = BinaryClassificationEvaluator.load(bce_path)
>>> str(evaluator2.getRawPredictionCol())
'raw'
```

버전 1.4.0의 새 기능입니다.

getMetricName()

metricName 값 혹은 기본값을 가져옵니다.

버전 1.4.0의 새 기능입니다.

setMetricName(value)

metricName의 값을 설정합니다.

버전 1.4.0의 새 기능입니다.

setParams(self, rawPredictionCol='rawPrediction', labelCol='label', metricName='areaUnderROC')

이진 분류 평가자를 위한 파라미터를 설정합니다.

버전 1.4.0의 새 기능입니다.

```
class pyspark.ml.evaluation.ClusteringEvaluator(*args, **kwargs)
```

노트: 실험적

예측과 특징의 두 가지 입력 열을 예상하는 군집 분석 결과 평가자. 메트릭은 유클리드 거리 제곱을 사용하여 실루엣 측도를 계산합니다.

실루엣은 군집 내의 일관성을 검증하기 위한 측도입니다. 1에서 -1 사이의 범위를 가지며, 여기서 1에 가까운 값은 군집 내의 점들이 동일한 군집 내의 다른 점들에 가깝고 다른 군집의 점들로부터 멀리 떨어져 있음을 의미합니다.

```
>>> from pyspark.ml.linalg import Vectors
>>> featureAndPredictions = map(lambda x: (Vectors.dense(x[0]), x[1]),
...     [[[0.0, 0.5], 0.0), ([0.5, 0.0], 0.0), ([10.0, 11.0], 1.0),
...     ([10.5, 11.5], 1.0), ([1.0, 1.0], 0.0), ([8.0, 6.0], 1.0)])
>>> dataset = spark.createDataFrame(featureAndPredictions, ["features",
...     "prediction"])
...
>>> evaluator = ClusteringEvaluator(predictionCol="prediction")
>>> evaluator.evaluate(dataset)
0.9079...
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
>>> ce_path = temp_path + "/ce"
>>> evaluator.save(ce_path)
>>> evaluator2 = ClusteringEvaluator.load(ce_path)
>>> str(evaluator2.getPredictionCol())
'prediction'
```

버전 2.3.0의 새 기능입니다.

getDistanceMeasure()

distanceMeasure 값을 가져옵니다.

버전 2.4.0의 새 기능입니다.

getMetricName()

metricName 값 혹은 기본값을 가져옵니다.

버전 2.3.0의 새 기능입니다.

setDistanceMeasure(distanceMeasure) (version 2.3.0)

distanceMeasure의 값을 설정합니다.

버전 2.4.0의 새 기능입니다.

setMetricName(metricName)

metricName의 값을 설정합니다.

버전 2.3.0의 새 기능입니다.

setParams(version 2.3.0) (*predictionCol='prediction', featuresCol='features', metricName='silhouette', distanceMeasure='squaredEuclidean'*)

군집 분류 평가자를 위한 파라미터를 설정합니다.

버전 2.3.0의 새 기능입니다.

class pyspark.ml.evaluation.Evaluator

예측에서 메트릭을 계산하는 평가자의 기본 클래스입니다.

버전 1.4.0의 새 기능입니다.

evaluate(dataset, params=None)

옵션 파라미터를 사용하여 출력을 평가합니다.

파라미터

- **dataset** – 라벨/관측치, 예측값을 포함하는 데이터 셋

- **params** – 내장된 params를 중단하는 선택적인 파라미터 맵

반환 메트릭

버전 1.4.0의 새 기능입니다.

isLargerBetter()

`evaluate()`에서 반환하는 메트릭을 최대화(True, default) 또는 최소화(False)해야 하는지 나타냅니다. 주어진 평가자는 최대화 또는 최소화할 수 있는 여러 메트릭을 지원할 수 있습니다.

버전 1.5.0의 새 기능입니다.

```
class pyspark.ml.evaluation.MulticlassClassificationEvaluator(*args,  
                                                               **kwargs)
```

노트: 실험적

예측 및 라벨의 두 가지 입력 열이 예상되는 다중 클래스 분류를 위한 평가자.

```
>>> scoreAndLabels = [(0.0, 0.0), (0.0, 1.0), (0.0, 0.0),  
...      (1.0, 0.0), (1.0, 1.0), (1.0, 1.0), (1.0, 1.0), (2.0, 2.0), (2.0,  
↪ 0.0)]  
>>> dataset = spark.createDataFrame(scoreAndLabels, ["prediction", "label  
↪"])  
...  
>>> evaluator = MulticlassClassificationEvaluator(predictionCol=  
↪ "prediction")  
>>> evaluator.evaluate(dataset)  
0.66...  
>>> evaluator.evaluate(dataset, {evaluator.metricName: "accuracy"})  
0.66...  
>>> mce_path = temp_path + "/mce"  
>>> evaluator.save(mce_path)  
>>> evaluator2 = MulticlassClassificationEvaluator.load(mce_path)  
>>> str(evaluator2.getPredictionCol())  
'prediction'
```

버전 1.5.0의 새 기능입니다.

getMetricName()

metricName 값 혹은 기본값을 가져옵니다.

버전 1.5.0의 새 기능입니다.

setMetricName(value)

metricName의 값을 설정합니다.

버전 1.5.0의 새 기능입니다.

setParams(self, predictionCol='prediction', labelCol='label', metricName='f1')

다중 클래스 분류 평가자를 위한 파라미터들을 설정합니다.

버전 1.5.0의 새 기능입니다.

```
class pyspark.ml.evaluation.RegressionEvaluator(*args,  
                                               **kwargs)
```

노트: 실험적

예측 및 라벨 두 가지 입력 열이 예상되는 회귀 분석을 위한 평가자.

```

>>> scoreAndLabels = [(-28.98343821, -27.0), (20.21491975, 21.5),
...      (-25.98418959, -22.0), (30.69731842, 33.0), (74.69283752, 71.0)]
>>> dataset = spark.createDataFrame(scoreAndLabels, ["raw", "label"])
...
>>> evaluator = RegressionEvaluator(predictionCol="raw")
>>> evaluator.evaluate(dataset)
2.842...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "r2"})
0.993...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "mae"})
2.649...
>>> re_path = temp_path + "/re"
>>> evaluator.save(re_path)
>>> evaluator2 = RegressionEvaluator.load(re_path)
>>> str(evaluator2.getPredictionCol())
'raw'

```

버전 1.4.0의 새 기능입니다.

getMetricName()

metricName 값 혹은 기본값을 가져옵니다.

버전 1.4.0의 새 기능입니다.

setMetricName(value)

metricName의 값을 설정합니다.

버전 1.4.0의 새 기능입니다.

setParams(self, predictionCol='prediction', labelCol='label', metricName='rmse')

회귀분석 평가자를 위한 파라미터들을 설정합니다.

버전 1.4.0의 새 기능입니다.

28 장

주요 참고

문헌 정보

[Feng2017] W. Feng 과 M. Chen. [아파치 스파크 학습하기](#), Github 2017.

[Feng2016PSD] W. Feng, A. J. Salgado, C. Wang, S. M. Wise. p-라플라시안 항을 포함하는 일부 비선형 타원방정식의 사전조건화된 가장 가파른 하강법. *J. Comput. Phys.*, 334:45–67, 2016.

[Feng2014] W. Feng. [수치해석을 위한 선행 노트](#), 뉴스빌 테네시 대학교.

[Karau2015] H. Karau, A. Konwinski, P. Wendell and M. Zaharia. [스파크 학습하기: 가볍고 빠른 빅 데이터 분석](#). O'Reilly Media, Inc., 2015

[Kirillov2016] Anton Kirillov. [아파치 스파크: 핵심 개념, 아키텍쳐와 내부 구조](#). <http://datastrophic.io/core-concepts-architecture-and-internals-of-apache-spark/>

[zeppelin2nb] Wenqiang Feng and Ryan Blue. [주피터 노트북 라이브러리 API에서 제플린 노트북](#), 2019.

[jupyter-zeppelin] Ryan Blue. [주피터/제플린 전환](#), 2017.

파이썬 모듈 인덱스

p

`pyspark.ml.classification`, 430
`pyspark.ml.clustering`, 450
`pyspark.ml.evaluation`, 477
`pyspark.ml.pipeline`, 470
`pyspark.ml.recommendation`, 465
`pyspark.ml.regression`, 411
`pyspark.ml.stat`, 405
`pyspark.ml.tuning`, 472

인덱스

A

accuracy() (pyspark.ml.classification.LogisticRegressionSummary property), 441
addGrid() (pyspark.ml.tuning.ParamGridBuilder method), 475
AFTSurvivalRegression (class in pyspark.ml.regression), 411
AFTSurvivalRegressionModel (class in pyspark.ml.regression), 412
aic() (pyspark.ml.regression.GeneralizedLinearRegression property), 420
ALS (class in pyspark.ml.recommendation), 465
ALSModel (class in pyspark.ml.recommendation), 469
areaUnderROC() (pyspark.ml.classification.BinaryLogisticRegressionSummary property), 430
assignClusters() (pyspark.ml.clustering.PowerIterationClustering method), 463
avgMetrics (pyspark.ml.tuning.CrossValidatorModel attribute), 474

B

baseOn() (pyspark.ml.tuning.ParamGridBuilder method), 475
bestModel (pyspark.ml.tuning.CrossValidatorModel attribute), 474
bestModel (pyspark.ml.tuning.TrainValidationSplitModel attribute), 477
BinaryClassificationEvaluator (class in pyspark.ml.evaluation), 477

BinaryLogisticRegressionSummary (class in pyspark.ml.classification), 430
BinaryLogisticRegressionTrainingSummary (class in pyspark.ml.classification), 431
BisectingKMeans (class in pyspark.ml.clustering), 450
BisectingKMeansModel (class in pyspark.ml.clustering), 451
BisectingKMeansSummary (class in pyspark.ml.clustering), 452

boundaries() (pyspark.ml.regression.IsotonicRegressionModel property), 423
build() (pyspark.ml.tuning.ParamGridBuilder method), 475

C

ChiSquareTest (class in pyspark.ml.stat), 405
clusterCenters() (pyspark.ml.clustering.BisectingKMeansModel method), 451
clusterCenters() (pyspark.ml.clustering.KMeansModel method), 457
ClusteringEvaluator (class in pyspark.ml.evaluation), 478
coefficientMatrix() (pyspark.ml.classification.LogisticRegressionModel property), 440
coefficients() (pyspark.ml.classification.LinearSVCModel property), 437
coefficients() (pyspark.ml.classification.LogisticRegressionModel property), 440
coefficients() (pyspark.ml.regression.AFTSurvivalRegressionModel

D

| | |
|---|---|
| <i>property), 413</i> | |
| <code>coefficients()</code> | (pys- |
| <i>park.ml.regression.GeneralizedLinearRegressionModel(class in pyspark.ml.classification), 431</i> | <i>DecisionTreeClassificationModel</i> |
| <i>property), 419</i> | |
| <code>coefficients()</code> | (pys- |
| <i>park.ml.regression.LinearRegressionModel(property), 424</i> | <i>DecisionTreeClassifier (class in pyspark.ml.classification), 432</i> |
| <code>coefficientStandardErrors()</code> | (pys- |
| <i>park.ml.regression.GeneralizedLinearRegressionTrainingSummary(property), 421</i> | <i>DecisionTreeRegressionModel (class in pyspark.ml.regression), 413</i> |
| <code>coefficientStandardErrors()</code> | (pys- |
| <i>park.ml.regression.LinearRegressionSummary(property), 425</i> | <i>DecisionTreeRegressor (class in pyspark.ml.regression), 413</i> |
| <code>computeCost()</code> | (pys- |
| <i>park.ml.clustering.BisectingKMeansModel method), 452</i> | <i>degreesOfFreedom()</i> |
| <code>computeCost()</code> | (pys- |
| <i>park.ml.clustering.KMeansModel method), 457</i> | (pys- |
| Configure Spark on Mac and Ubuntu,
17 | <i>park.ml.regression.GeneralizedLinearRegressionSummary property), 420</i> |
| <code>copy()</code> | (pys- |
| <i>pyspark.ml.classification.OneVsRest method), 447</i> | <i>park.ml.regression.LinearRegressionSummary property), 425</i> |
| <code>copy()</code> | (pys- |
| <i>pyspark.ml.classification.OneVsRestModel method), 448</i> | <i>describeTopics()</i> |
| <code>copy()</code> | (pys- |
| <i>pyspark.ml.pipeline.Pipeline method), 470</i> | <i>park.ml.clustering.LDAModel method), 462</i> |
| <code>copy()</code> | (pys- |
| <i>pyspark.ml.pipeline.PipelineModel method), 471</i> | <i>deviance()</i> |
| <code>copy()</code> | (pys- |
| <i>pyspark.ml.tuning.CrossValidator method), 473</i> | <i>park.ml.regression.GeneralizedLinearRegressionSummary property), 420</i> |
| <code>copy()</code> | (pys- |
| <i>pyspark.ml.tuning.CrossValidatorModel method), 474</i> | <i>devianceResiduals()</i> |
| <code>copy()</code> | (pys- |
| <i>pyspark.ml.tuning.TrainValidationSplit method), 476</i> | <i>park.ml.regression.LinearRegressionSummary property), 425</i> |
| <code>copy()</code> | (pys- |
| <i>pyspark.ml.tuning.TrainValidationSplitModel method), 477</i> | <i>dispersion()</i> |
| <code>corr()</code> | (pys- |
| <i>pyspark.ml.stat.Correlation static method), 406</i> | <i>park.ml.regression.GeneralizedLinearRegressionSummary property), 420</i> |
| <code>Correlation(class in pyspark.ml.stat), 406</code> | <i>DistributedLDAModel (class in pyspark.ml.clustering), 452</i> |
| <code>count()</code> | (pys- |
| <i>pyspark.ml.stat.Summarizer static method), 409</i> | E |
| <code>CrossValidator(class in pyspark.ml.tuning), 472</code> | |
| <code>CrossValidatorModel(class in pyspark.ml.tuning), 473</code> | |
| | <code>estimatedDocConcentration()</code> |
| | (pys- |
| | <i>park.ml.clustering.LDAModel method), 462</i> |
| | <code>evaluate()</code> |
| | (pys- |
| | <i>park.ml.classification.LogisticRegressionModel method), 440</i> |
| | <code>evaluate()</code> |
| | (pys- |
| | <i>park.ml.regression.GeneralizedLinearRegressionModel method), 419</i> |
| | <code>evaluate()</code> |
| | (pys- |
| | <i>park.ml.regression.LinearRegressionModel method), 424</i> |
| | <code>evaluateEachIteration()</code> |
| | (pys- |
| | <i>park.ml.classification.GBTClassificationModel method), 433</i> |

evaluateEachIteration() (pyspark.ml.regression.GBTRegressionModel method), 414

Evaluator (class in pyspark.ml.evaluation), 479

explainedVariance() (pyspark.ml.regression.LinearRegressionSummary property), 425

F

falsePositiveRateByLabel() (pyspark.ml.classification.LogisticRegressionSummary property), 441

featureImportances() (pyspark.ml.classification.DecisionTreeClassificationModel property), 432

featureImportances() (pyspark.ml.classificationGBTClassificationModel property), 433

featureImportances() (pyspark.ml.classification.RandomForestClassificationModel property), 448

featureImportances() (pyspark.ml.regression.DecisionTreeRegressionModel property), 413

featureImportances() (pyspark.ml.regression.GBTRegressionModel property), 415

featureImportances() (pyspark.ml.regression.RandomForestRegressionModel property), 428

featuresCol() (pyspark.ml.classification.LogisticRegressionSummary property), 441

featuresCol() (pyspark.ml.regression.LinearRegressionSummary property), 426

fMeasureByLabel() (pyspark.ml.classification.LogisticRegressionSummary method), 441

fMeasureByThreshold() (pyspark.ml.classification.BinaryLogisticRegressionSummary property), 430

G

GaussianMixture (class in pyspark.ml.clustering), 453

GaussianMixtureModel (class in pyspark.ml.clustering), 455

GaussianMixtureSummary (class in pyspark.ml.clustering), 455

gaussiansDF() (pyspark.ml.clustering.GaussianMixtureModel property), 455

GBTClassificationModel (class in pyspark.ml.classification), 433

GBTClassifier (class in pyspark.ml.classification), 434

GBTRegressionModel (class in pyspark.ml.regression), 414

GBTRegressor (class in pyspark.ml.regression), 415

GeneralizedLinearRegression (class in pyspark.ml.regression), 416

GeneralizedLinearRegressionModel (class in pyspark.ml.regression), 419

GeneralizedLinearRegressionSummary (class in pyspark.ml.regression), 419

GeneralizedLinearRegressionTrainingSummary (class in pyspark.ml.regression), 421

getAlpha() (pyspark.ml.recommendation.ALS method), 466

getBlockSize() (pyspark.ml.classification.MultilayerPerceptronClassifier method), 444

getCensorCol() (pyspark.ml.regression.AFTSurvivalRegression method), 412

getCheckpointFiles() (pyspark.ml.clustering.DistributedLDAModel method), 452

getColdStartStrategy() (pyspark.ml.recommendation.ALS method), 466

getDistanceMeasure() (pyspark.ml.clustering.BisectingKMeans method), 451

getDistanceMeasure() (pyspark.ml.clustering.KMeans method), 456

getDistanceMeasure() (pyspark.ml.evaluation.ClusteringEvaluator method), 479

getDocConcentration() (pyspark.ml.clustering.LDA method), 459

getDstCol() (pyspark.ml.clustering.PowerIterationClustering

```

        method), 464
getEpsilon ()                               (pys-
    park.ml.regression.LinearRegression
    method), 424
getFamily ()                                (pys-
    park.ml.classification.LogisticRegression
    method), 438
getFamily ()                                (pys-
    park.ml.regression.GeneralizedLinearRegression
    method), 418
getFeatureIndex ()                           (pys-
    park.ml.regression.IsotonicRegression
    method), 422
getFinalStorageLevel ()                     (pys-
    park.ml.recommendation.ALS
    method), 466
getImplicitPrefs ()                          (pys-
    park.ml.recommendation.ALS
    method), 467
getInitialWeights ()                         (pys-
    park.ml.classification.MultilayerPerceptronClassifier
    method), 444
getInitMode () (pyspark.ml.clustering.KMeans
    method), 456
getInitMode ()                               (pys-
    park.ml.clustering.PowerIterationClustering
    method), 464
getInitSteps ()                              (pys-
    park.ml.clustering.KMeans
    method), 457
getIntermediateStorageLevel ()              (pys-
    park.ml.recommendation.ALS
    method), 467
getIsotonic ()                               (pys-
    park.ml.regression.IsotonicRegression
    method), 422
getItemCol () (pyspark.ml.recommendation.ALS
    method), 467
getK () (pyspark.ml.clustering.BisectingKMeans
    method), 451
getK () (pyspark.ml.clustering.GaussianMixture
    method), 454
getK () (pyspark.ml.clustering.KMeans
    method), 457
getK () (pyspark.ml.clustering.LDA
    method), 459
getK () (pyspark.ml.clustering.PowerIterationClustering
    method), 464
getKeepLastCheckpoint () (pys-
    park.ml.clustering.LDA
    method), 459
getLayers ()                                (pys-
    park.ml.classification.MultilayerPerceptronClassifier
    method), 444
getLearningDecay ()                          (pys-
    park.ml.clustering.LDA
    method), 459
getLearningOffset ()                         (pys-
    park.ml.clustering.LDA
    method), 459
getLink ()                                  (pys-
    park.ml.regression.GeneralizedLinearRegression
    method), 418
getLinkPower ()                             (pys-
    park.ml.regression.GeneralizedLinearRegression
    method), 418
getLinkPredictionCol () (pys-
    park.ml.regression.GeneralizedLinearRegression
    method), 418
getLossType ()                               (pys-
    park.ml.classification.GBTClassifier
    method), 435
getLossType ()                               (pys-
    park.ml.regression.GBTRegressor
    method), 416
getLowerBoundsOnCoefficients () (pys-
    park.ml.classification.LogisticRegression
    method), 438
getLowerBoundsOnIntercepts () (pys-
    park.ml.classification.LogisticRegression
    method), 438
getMetricName ()                            (pys-
    park.ml.evaluation.BinaryClassificationEvaluator
    method), 478
getMetricName ()                            (pys-
    park.ml.evaluation.ClusteringEvaluator
    method), 479
getMetricName ()                            (pys-
    park.ml.evaluation.MulticlassClassificationEvaluator
    method), 480
getMetricName ()                            (pys-
    park.ml.evaluation.RegressionEvaluator
    method), 481
getMinDivisibleClusterSize () (pys-
    park.ml.clustering.BisectingKMeans
    method), 451
getModelType ()                            (pys-
    park.ml.classification.NaiveBayes
    method), 446

```

| | | |
|---|-------------------|---|
| getNonnegative()
park.ml.recommendation.ALS
467 | (pys-
method), | method), 438 |
| getNumFolds()
park.ml.tuning.CrossValidator
473 | (pys-
method), | getThresholds() (pys-
park.ml.classification.LogisticRegression
method), 439 |
| getNumItemBlocks()
park.ml.recommendation.ALS
467 | (pys-
method), | getTopicConcentration() (pys-
park.ml.clustering.LDA method), 459 |
| getNumUserBlocks()
park.ml.recommendation.ALS
467 | (pys-
method), | getTopicDistributionCol() (pys-
park.ml.clustering.LDA method), 460 |
| getOffsetCol()
park.ml.regression.GeneralizedLinearRegression
method), 418 | (pys-
method), | getTrainRatio() (pys-
park.ml.tuning.TrainValidationSplit
method), 476 |
| getOptimizeDocConcentration()
park.ml.clustering.LDA method), 459 | (pys-
method), | getUpperBoundsOnCoefficients() (pys-
park.ml.classification.LogisticRegression
method), 439 |
| getOptimizer()
(pyspark.ml.clustering.LDA
method), 459 | (pys-
method), | getUpperBoundsOnIntercepts() (pys-
park.ml.classification.LogisticRegression
method), 439 |
| getQuantileProbabilities()
(pys-
park.ml.regression.AFTSurvivalRegression
method), 412 | (pys-
method), | getUserCol() (pyspark.ml.recommendation.ALS
method), 467 |
| getQuantilesCol()
(pys-
park.ml.regression.AFTSurvivalRegression
method), 412 | (pys-
method), | getVariancePower() (pys-
park.ml.regression.GeneralizedLinearRegression
method), 418 |
| getRank()
(pyspark.ml.recommendation.ALS
method), 467 | (pys-
method), | H |
| getRatingCol()
(pys-
park.ml.recommendation.ALS
method), 467 | (pys-
method), | hasSummary() (pys-
park.ml.classification.LogisticRegressionModel
property), 440 |
| getSmoothing()
(pys-
park.ml.classification.NaiveBayes method),
446 | (pys-
method), | hasSummary() (pys-
park.ml.clustering.BisectingKMeansModel
property), 452 |
| getSrcCol()
(pys-
park.ml.clustering.PowerIterationClustering
method), 464 | (pys-
method), | hasSummary() (pys-
park.ml.clustering.GaussianMixtureModel
property), 455 |
| getStagePath()
(pys-
park.ml.pipeline.PipelineSharedReadWrite
static method), 472 | (pys-
method), | hasSummary() (pys-
park.ml.clustering.KMeansModel prop-
erty), 457 |
| getStages()
(pyspark.ml.pipeline.Pipeline
method), 470 | (pys-
method), | hasSummary() (pys-
park.ml.regression.GeneralizedLinearRegressionModel
property), 419 |
| getStepSize()
(pys-
park.ml.classification.MultilayerPerceptronClassifier
method), 444 | (pys-
method), | hasSummary() (pys-
park.ml.regression.LinearRegressionModel
property), 425 |
| getSubsamplingRate()
(pys-
park.ml.clustering.LDA method), 459 | (pys-
method), | intercept() (pys-
park.ml.classification.LinearSVCModel
property), 437 |
| getThreshold()
(pys-
park.ml.classification.LogisticRegression | (pys-
method), | |

```

intercept()                               (pys- LDAModel (class in pyspark.ml.clustering), 461
    park.ml.classification.LogisticRegressionModel linearRegression (class in pyspark.ml.regression), 423
    property), 440

intercept()                               (pys- LinearRegressionModel (class in pyspark.ml.regression), 424
    park.ml.regression.AFTSurvivalRegressionModel property), 413
                                                LinearRegressionSummary (class in pyspark.ml.regression), 428

intercept()                               (pys- park.ml.regression), 425
    park.ml.regression.GeneralizedLinearRegressionModel RegressionTrainingSummary
    property), 419

intercept()                               (pys- LinearSVC (class in pyspark.ml.classification),
    park.ml.regression.LinearRegressionModel        435
    property), 425
                                                LinearSVCModel (class in pyspark.ml.classification), 437

interceptVector()                          (pys- park.ml.classification), 437
    park.ml.classification.LogisticRegressionModel load () (pyspark.ml.pipeline.PipelineModelReader
    property), 440
                                                method), 471

isDistributed()                           (pys- load () (pyspark.ml.pipeline.PipelineReader
    park.ml.clustering.LDAModel      method), 471
    462
                                                load () (pyspark.ml.pipeline.PipelineSharedReadWrite
                                                static method), 472

isLargerBetter()                          (pys- LocalLDAModel (class in pyspark.ml.clustering),
    park.ml.evaluation.Evaluator       method), 462
    479

IsotonicRegression (class in pyspark.ml.regression), 422
                                                LogisticRegression (class in pyspark.ml.classification), 437

IsotonicRegressionModel (class in pyspark.ml.regression), 422
                                                LogisticRegressionModel (class in pyspark.ml.classification), 440

itemFactors()                            (pys- LogisticRegressionSummary (class in pyspark.ml.classification),
    park.ml.recommendation(ALSModel      441
    property), 469
                                                LogisticRegressionTrainingSummary
                                                (class in pyspark.ml.classification), 442

logLikelihood()                           (pys- logLikelihood () (pyspark.ml.clustering.GaussianMixtureSummary
                                                property), 455

logLikelihood()                           (pys- park.ml.clustering.LDAModel      method), 462
                                                462
                                                logPerplexity () (pyspark.ml.clustering.LDAModel      method),
                                                462

labelCol()                                (pys- prior () (pyspark.ml.clustering.DistributedLDAModel
    park.ml.classification.LogisticRegressionSummary      method), 452
    property), 441

labelCol()                                (pys- M
    park.ml.regression.LinearRegressionSummary
    property), 426

labels() (pyspark.ml.classification.LogisticRegressionSummarySummarizer(pyspark.ml.stat.Summarizer static method),
    property), 441
                                                409

layers() (pyspark.ml.classification.MultilayerPerceptronClassifier(pyspark.ml.stat.Summarizer static
    property), 443
                                                method), 409

```

K

KMeans (class in pyspark.ml.clustering), 455
 KMeansModel (class in pyspark.ml.clustering), 457
 KolmogorovSmirnovTest (class in pyspark.ml.stat), 407

L

labelCol() (pyspark.ml.classification.LogisticRegressionSummary
 property), 441
 labelCol() (pyspark.ml.regression.LinearRegressionSummary
 property), 426

labels() (pyspark.ml.classification.LogisticRegressionSummarySummarizer(pyspark.ml.stat.Summarizer static method),
 property), 441
 layers() (pyspark.ml.classification.MultilayerPerceptronClassifier(pyspark.ml.stat.Summarizer static
 property), 443
 LDA (class in pyspark.ml.clustering), 458

M

prior () (pyspark.ml.clustering.DistributedLDAModel
 method), 452
 prior () (pyspark.ml.clustering.DistributedLDAModel
 method), 409

meanAbsoluteError () (pyspark.ml.regression.LinearRegressionSummary static method), 410
property, 426

meanSquaredError () (pyspark.ml.regression.LinearRegressionSummary static method), 426

metrics () (pyspark.ml.stat.Summarizer static method), 409

min () (pyspark.ml.stat.Summarizer static method), 410

module

- pyspark.ml.classification, 430
- pyspark.ml.clustering, 450
- pyspark.ml.evaluation, 477
- pyspark.ml.pipeline, 470
- pyspark.ml.recommendation, 465
- pyspark.ml.regression, 411
- pyspark.ml.stat, 405
- pyspark.ml.tuning, 472

MulticlassClassificationEvaluator (class in pyspark.ml.evaluation), 480

MultilayerPerceptronClassificationModel (class in pyspark.ml.classification), 443

MultilayerPerceptronClassifier (class in pyspark.ml.classification), 443

N

NaiveBayes (class in pyspark.ml.classification), 445

NaiveBayesModel (class in pyspark.ml.classification), 446

normL1 () (pyspark.ml.stat.Summarizer static method), 410

normL2 () (pyspark.ml.stat.Summarizer static method), 410

nullDeviance () (pyspark.ml.regression.GeneralizedLinearRegressionSummary static method), 420

numInstances () (pyspark.ml.regression.GeneralizedLinearRegressionSummary static method), 420

numInstances () (pyspark.ml.regression.LinearRegressionSummary static method), 426

numIterations () (pyspark.ml.regression.GeneralizedLinearRegressionTrainingSummary static method), 421

o

objectiveHistory () (pyspark.ml.classification.LogisticRegressionTrainingSummary static method), 442

objectiveHistory () (pyspark.ml.regression.LinearRegressionTrainingSummary static method), 428

OneVsRest (class in pyspark.ml.classification), 446

OneVsRestModel (class in pyspark.ml.classification), 448

P

ParamGridBuilder (class in pyspark.ml.tuning), 474

pi () (pyspark.ml.classification.NaiveBayesModel static method), 446

Pipeline (class in pyspark.ml.pipeline), 470

PipelineModel (class in pyspark.ml.pipeline), 471

PipelineModelReader (class in pyspark.ml.pipeline), 471

PipelineModelWriter (class in pyspark.ml.pipeline), 471

PipelineReader (class in pyspark.ml.pipeline), 471

PipelineSharedReadWrite (class in pyspark.ml.pipeline), 472

PipelineWriter (class in pyspark.ml.pipeline), 472

PowerIterationClustering (class in pyspark.ml.clustering), 463

pr () (pyspark.ml.classification.BinaryLogisticRegressionSummary static method), 430

precisionByLabel () (pyspark.ml.classification.LogisticRegressionSummary static method), 441

precisionByThreshold () (pyspark.ml.classification.BinaryLogisticRegressionSummary static method), 431

predict () (pyspark.ml.regression.AFTSurvivalRegressionModel static method), 413

predictionCol () (pyspark.ml.classification.LogisticRegressionSummary static method), 443

```

    property), 441                               module, 411
predictionCol()                                (pys- pyspark.ml.stat
    park.ml.regression.GeneralizedLinearRegressionSummary, 405
    property), 420                               pyspark.ml.tuning
predictionCol()                                (pys- module, 472
    park.ml.regression.LinearRegressionSummaryR
    property), 427
predictions()                                 (pys- r2 () (pyspark.ml.regression.LinearRegressionSummary
    park.ml.classification.LogisticRegressionSummary   property), 427
    property), 441                               r2adj () (pyspark.ml.regression.LinearRegressionSummary
predictions()                                 (pys-   property), 427
    park.ml.regression.GeneralizedLinearRegressionRandomForestClassificationModel
    property), 420                               (class in pyspark.ml.classification), 448
predictions()                                 (pys- RandomForestClassifier (class in pys-
    park.ml.regression.IsotonicRegressionModel      park.ml.classification), 448
    property), 423                               RandomForestRegressionModel (class in
predictions()                                 (pys-   pyspark.ml.regression), 428
    park.ml.regression.LinearRegressionSummaryRandomForestRegressor (class in pys-
    property), 427                               park.ml.regression), 429
predictQuantiles()                            (pys- rank () (pyspark.ml.recommendation.ALSModel
    park.ml.regression.AFTSurvivalRegressionModel   property), 469
    method), 413                               rank () (pyspark.ml.regression.GeneralizedLinearRegressionSummary
probability()                                (pys-   property), 420
    park.ml.clustering.GaussianMixtureSummaryread () (pyspark.ml.pipeline.Pipeline   class
    property), 455                               method), 470
probabilityCol()                             (pys-   read () (pyspark.ml.pipeline.PipelineModel class
    park.ml.classification.LogisticRegressionSummary   method), 471
    property), 442                               read () (pyspark.ml.tuning.CrossValidator class
probabilityCol()                             (pys-   method), 473
    park.ml.clustering.GaussianMixtureSummaryread () (pyspark.ml.tuning.CrossValidatorModel
    property), 455                               class method), 474
pValues()                                    (pys-   read () (pyspark.ml.tuning.TrainValidationSplit
    park.ml.regression.GeneralizedLinearRegressionTrainingSummary), 476
    property), 421                               read () (pyspark.ml.tuning.TrainValidationSplitModel
pValues()                                    (pys-   class method), 477
    park.ml.regression.LinearRegressionSummaryrecallByLabel ()           (pys-
    property), 426                               park.ml.classification.LogisticRegressionSummary
pyspark.ml.classification                      module, 430                               property), 442
pyspark.ml.clustering                         module, 450                               recallByThreshold ()           (pys-
pyspark.ml.evaluation                        module, 477                               park.ml.classification.BinaryLogisticRegressionSummary
pyspark.ml.pipeline                          module, 470                               property), 431
pyspark.ml.recommendation                   module, 465                               recommendForAllItems ()           (pys-
pyspark.ml.regression

```

| | | | |
|---|---|--|-------|
| recommendForItemSubset () | (pys- | setCensorCol () | (pys- |
| park.ml.recommendation.ALSModel | | park.ml.regression.AFTSurvivalRegression | |
| method), 469 | | method), 412 | |
| recommendForUserSubset () | (pys- | setColdStartStrategy () | (pys- |
| park.ml.recommendation.ALSModel | | park.ml.recommendation.ALS | |
| method), 469 | | method), 467 | |
| RegressionEvaluator (class in | pys- | setDistanceMeasure () | (pys- |
| park.ml.evaluation), 480 | | park.ml.clustering.BisectingKMeans | |
| residualDegreeOfFreedom () | (pys- | method), 451 | |
| park.ml.regression.GeneralizedLinearRegressionSummary | | setDistanceMeasure () | (pys- |
| property), 420 | | park.ml.clustering.KMeans | |
| residualDegreeOfFreedomNull () | (pys- | method), 457 | |
| park.ml.regression.GeneralizedLinearRegressionSummary | | setDistanceMeasure () | (pys- |
| property), 421 | | park.ml.evaluation.ClusteringEvaluator | |
| residuals () | (pys- | method), 479 | |
| park.ml.regression.GeneralizedLinearRegressionSummary | | setDiameter () | (pys- |
| method), 421 | | park.ml.clustering.PowerIterationClustering | |
| residuals () | (pys- | method), 464 | |
| park.ml.regression.LinearRegressionSummary | | setDstCol () | (pys- |
| property), 427 | | park.ml.clustering.PowerIterationClustering | |
| roc () | (pyspark.ml.classification.BinaryLogisticRegressionSummary) | setFamily () | (pys- |
| property), 431 | | park.ml.regression.LinearRegression | |
| rootMeanSquaredError () | (pys- | method), 424 | |
| park.ml.regression.LinearRegressionSummary | | setFamily () | (pys- |
| property), 427 | | park.ml.classification.LogisticRegression | |
| Run on Databricks Community Cloud, | | method), 439 | |
| 11 | | setFamily () | (pys- |
| S | | park.ml.regression.GeneralizedLinearRegression | |
| saveImpl () | (pys- | method), 418 | |
| park.ml.pipeline.PipelineModelWriter | | setFeatureIndex () | (pys- |
| method), 471 | | park.ml.regression.IsotonicRegression | |
| saveImpl () | (pys- | method), 422 | |
| park.ml.pipeline.PipelineSharedReadWrite | | setFeatureSubsetStrategy () | (pys- |
| static method), 472 | | park.ml.classificationGBTClassifier | |
| saveImpl () | (pyspark.ml.pipeline.PipelineWriter | setFeatureSubsetStrategy () | (pys- |
| method), 472 | | park.ml.classification.RandomForestClassifier | |
| scale () | (pyspark.ml.regression.AFTSurvivalRegressionModel | setFeatureSubsetStrategy () | (pys- |
| property), 413 | | park.ml.classification.RandomForestClassifier | |
| scale () | (pyspark.ml.regression.LinearRegressionModel | setFeatureSubsetStrategy () | (pys- |
| property), 425 | | park.ml.regression.GBTRegressor | |
| Set up Spark on Cloud, 29 | | method), 416 | |
| setAlpha () | (pyspark.ml.recommendation.ALS | setFeatureSubsetStrategy () | (pys- |
| method), 467 | | park.ml.regression.RandomForestRegressor | |
| setBlockSize () | (pys- | method), 430 | |
| park.ml.classification.MultilayerPerceptronClassifier | | setFinalStorageLevel () | (pys- |
| method), 444 | | park.ml.recommendation.ALS | |
| | | method), 467 | |

```

setImplicitPrefs()           (pys-          method), 418
    park.ml.recommendation.ALS   method), setLossType()           (pys-
        468                      method), 468
setInitialWeights()          (pys-          method), 435
    park.ml.classification.MultilayerPerceptronClassifier
        method), 444
setInitMode()                (pys-          method), 416
    pyspark.ml.clustering.KMeans
        method), 457
setInitMode()                (pys-          method), 416
    park.ml.clustering.PowerIterationClustering
        method), 464
setInitSteps()               (pys-          method), 439
    park.ml.clustering.KMeans
        method), 457
setIntermediateStorageLevel() (pys-          method), 439
    park.ml.recommendation.ALS
        method), 468
setIsotonic()                (pys-          method), 478
    park.ml.regression.IsotonicRegression
        method), 422
setItemCol()                (pys-          method), 480
    pyspark.ml.recommendation.ALS
        method), 468
setK()                       (pys-          method), 481
    pyspark.ml.clustering.BisectingKMeans
        method), 451
setK()                       (pys-          method), 481
    pyspark.ml.clustering.GaussianMixture
        method), 454
setK()                       (pys-          method), 451
    pyspark.ml.clustering.KMeans
        method), 457
setK()                       (pys-          method), 451
    pyspark.ml.clustering.LDA
        method), 460
setK()                       (pys-          method), 446
    pyspark.ml.clustering.PowerIterationClustering
        method), 464
setKeepLastCheckpoint()       (pys-          method), 468
    park.ml.clustering.LDA
        method), 460
setLayers()                  (pys-          method), 468
    park.ml.classification.MultilayerPerceptronClassifier
        method), 444
setLearningDecay()           (pys-          method), 473
    park.ml.clustering.LDA
        method), 460
setLearningOffset()           (pys-          method), 473
    park.ml.clustering.LDA
        method), 460
setLink()                     (pys-          method), 468
    park.ml.regression.GeneralizedLinearRegression
        method), 418
setLinkPower()                (pys-          method), 468
    park.ml.regression.GeneralizedLinearRegression
        method), 418
setLinkPredictionCol()        (pys-          method), 498
    park.ml.regression.GeneralizedLinearRegression
        method), 498
setLossType()                 (pys-          method), 416
    park.ml.classification.GBTClassifier
        method), 435
setLowerBoundsOnCoefficients() (pys-          method), 416
    park.ml.classification.LogisticRegression
        method), 439
setLowerBoundsOnIntercepts()   (pys-          method), 439
    park.ml.classification.LogisticRegression
        method), 439
setMetricName()               (pys-          method), 479
    park.ml.evaluation.BinaryClassificationEvaluator
        method), 478
setMetricName()               (pys-          method), 479
    park.ml.evaluation.ClusteringEvaluator
        method), 479
setMetricName()               (pys-          method), 480
    park.ml.evaluation.MulticlassClassificationEvaluator
        method), 480
setMetricName()               (pys-          method), 481
    park.ml.evaluation.RegressionEvaluator
        method), 481
setMinDivisibleClusterSize()  (pys-          method), 451
    park.ml.clustering.BisectingKMeans
        method), 451
setModelType()                (pys-          method), 468
    park.ml.classification.NaiveBayes
        method), 446
setNonnegative()              (pys-          method), 468
    park.ml.recommendation.ALS
        method), 468
setNumBlocks()                (pys-          method), 468
    park.ml.recommendation.ALS
        method), 468
setNumFolds()                 (pys-          method), 473
    park.ml.tuning.CrossValidator
        method), 473
setNumItemBlocks()             (pys-          method), 468
    park.ml.recommendation.ALS
        method), 468
setNumUserBlocks()             (pys-          method), 468
    park.ml.recommendation.ALS
        method), 468
setOffsetCol()                (pys-          method), 498
    park.ml.regression.GeneralizedLinearRegression
        method), 498

```

```

    method), 418
setOptimizeDocConcentration() (pys- setParams()
park.ml.clustering.LDA method), 460 park.ml.evaluation.MulticlassClassificationEvaluator
setOptimizer() (pyspark.ml.clustering.LDA setParams()
method), 461 (pyspark.ml.evaluation.RegressionEvaluator
setParams() (pyspark.ml.classification.DecisionTreeClassifier setParams()
method), 433 (pyspark.ml.pipeline.Pipeline
method), 433 method), 470
setParams() (pyspark.ml.classification.GBTClassifier setParams() (pyspark.ml.recommendation.ALS
method), 435 (pyspark.ml.regression.AFTSurvivalRegression
method), 435 method), 468
setParams() (pyspark.ml.classification.LinearSVC method),
436 setParams() (pyspark.ml.regression.DecisionTreeRegressor
method), 412
setParams() (pyspark.ml.classification.LogisticRegression setParams() (pyspark.ml.regression.GBTRegressor
method), 439 method), 414
setParams() (pyspark.ml.classification.MultilayerPerceptronClassifier setParams()
method), 444 (pyspark.ml.regression.GeneralizedLinearRegression
method), 416 method), 419
setParams() (pyspark.ml.classification.NaiveBayes method),
446 setParams() (pyspark.ml.regression.IsotonicRegression
method), 422
setParams() (pyspark.ml.classification.OneVsRest method),
448 setParams() (pyspark.ml.regression.LinearRegression
method), 424
setParams() (pyspark.ml.classification.RandomForestClassifier setParams()
method), 449 (pyspark.ml.regression.RandomForestRegressor
method), 430
setParams() (pyspark.ml.clustering.BisectingKMeans setParams() (pyspark.ml.tuning.CrossValidator
method), 451 method), 473
setParams() (pyspark.ml.clustering.GaussianMixture setParams()
method), 454 (pyspark.ml.tuning.TrainValidationSplit
method), 454 method), 476
setParams() (pyspark.ml.clustering.KMeans setQuantileProbabilities()
method), 457 (pyspark.ml.regression.AFTSurvivalRegression
method), 412
setParams() (pyspark.ml.clustering.LDA setQuantilesCol()
method), 461 (pyspark.ml.regression.AFTSurvivalRegression
method), 412
setParams() (pyspark.ml.clustering.PowerIterationClustering setRank()
method), 464 (pyspark.ml.recommendation.ALS
method), 468
setParams() (pyspark.ml.evaluation.BinaryClassificationEvaluator setRatingCol()
method), 478 (pyspark.ml.recommendation.ALS
method), 468
setParams() (pyspark.ml.evaluation.ClusteringEvaluator
method), 479

```

```

setSmoothing()          (pys-           property), 440
    park.ml.classification.NaiveBayes method), 446
setSrcCol()             (pys-           summary()          (pys-
    park.ml.clustering.PowerIterationClustering method), 464
setStages()              (pyspark.ml.pipeline.Pipeline
    method), 471
setStepSize()             (pys-           property), 452
    park.ml.classification.MultilayerPerceptronClassifier
    method), 445
setSubsamplingRate()      (pys-           summary()          (pys-
    park.ml.clustering.LDA method), 461
setThreshold()            (pys-           park.ml.regression.GeneralizedLinearRegressionModel
    park.ml.classification.LogisticRegression
    method), 439
setThresholds()            (pys-           property), 419
    park.ml.classification.LogisticRegression
    method), 439
setTopicConcentration()   (pys-           summary()          (pys-
    park.ml.clustering.LDA method), 461
setTopicDistributionCol() (pys-           park.ml.regression.LinearRegressionModel
    park.ml.clustering.LDA method), 461
    property), 425
setTrainRatio()             (pys-           summary()          (pyspark.ml.stat.SummaryBuilder
    park.ml.tuning.TrainValidationSplit
    method), 476
setUpperBoundsOnCoefficients() (pys-           method), 410
    park.ml.classification.LogisticRegression
    method), 440
setUpperBoundsOnIntercepts() (pys-           SummaryBuilder (class in pyspark.ml.stat), 410
    park.ml.classification.LogisticRegression
    method), 440
setUserCol() (pyspark.ml.recommendation.ALS
    method), 468
setVariancePower()         (pys-           T
    park.ml.regression.GeneralizedLinearRegression
    method), 419
solver() (pyspark.ml.regression.GeneralizedLinearRegression
    property), 421
subModels                 (pys-           test()            (pyspark.ml.stat.ChiSquareTest
    park.ml.tuning.CrossValidatorModel
    attribute), 474
subModels                 (pys-           static
    park.ml.tuning.TrainValidationSplitModel
    attribute), 477
Summarizer (class in pyspark.ml.stat), 408
summary()                (pys-           method), 405
    park.ml.classification.LogisticRegressionModel

```

T

```

topicsMatrix()            (pys-           test()            (pyspark.ml.stat.KolmogorovSmirnovTest
    park.ml.clustering.LDAModel
    method), 462
toLocal()                 (pys-           static method), 407
    park.ml.clustering.DistributedLDAModel
method), 453
totalIterations()          (pys-           theta()           (pyspark.ml.classification.NaiveBayesModel
    park.ml.classification.LogisticRegressionTrainingSummary
    property), 446
property), 443
totalIterations()          (pys-           toLocal()          (pyspark.ml.clustering.DistributedLDAModel
    park.ml.regression.LinearRegressionTrainingSummary
    method), 462
method), 453
totalIterations()          (pys-           topicsMatrix()     (pyspark.ml.clustering.LDAModel
    park.ml.regression.LinearRegressionTrainingSummary
    method), 462
method), 453
totalIterations()          (pys-           totalIterations() (pyspark.ml.clustering.DistributedLDAModel
    park.ml.regression.LinearRegressionTrainingSummary
    method), 462
method), 453
totalIterations()          (pys-           trainingLogLikelihood() (pyspark.ml.clustering.DistributedLDAModel
    park.ml.regression.LinearRegressionTrainingSummary
    method), 462
method), 453
TrainValidationSplit (class in pys-           trees()           (pyspark.ml.classification.GBTClassificationModel
    park.ml.tuning), 475
TrainValidationSplitModel (class in pys-           property), 434
    park.ml.tuning), 476
trees() (pyspark.ml.classification.GBTClassificationModel
    property), 434

```

U

validateStages() (*pyspark.ml.pipeline.PipelineSharedReadWrite static method*), 472

validationMetrics (*pyspark.ml.tuning.TrainValidationSplitModel attribute*), 477

variance() (*pyspark.ml.stat.Summarizer static method*), 410

vocabSize() (*pyspark.ml.clustering.LDAModel method*), 462

W

```
weightedFalsePositiveRate()          (pys-
    park.ml.classification.LogisticRegressionSummary
    property), 442
weightedFMeasure()                  (pys-
    park.ml.classification.LogisticRegressionSummary
    method), 442
weightedPrecision()                (pys-
    park.ml.classification.LogisticRegressionSummary
    property), 442
weightedRecall()                   (pys-
    park.ml.classification.LogisticRegressionSummary
    property), 442
weightedTruePositiveRate()         (pys-
    park.ml.classification.LogisticRegressionSummary
```