7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

데이터 마이닝의 분류에 사용될 데이터셋은 “스마트카 상태 정보”로 하이브의 Manage 영역에 Managed_SmartCar_Status_Info라는 이름의 테이블에 약 200만 건의 데이터가 적재돼 있다.

	car_number	sex	age	marriage	region	job	car_capacity	car_year	car_model	tire_fl	tire_fr	tire_bl	tire_br	light_fl	light_fr	light_bl	light_br
0	U0001	여	43	미혼	전남	학생	2000	2002	B	82	81	92	91	1	1	1	1
1	U0001	여	43	미혼	전남	학생	2000	2002	B	93	86	78	88	1	1	1	1
2	U0001	여	43	미혼	전남	학생	2000	2002	B	95	99	95	71	1	1	1	1
3	U0001	여	43	미혼	전남	학생	2000	2002	B	79	72	84	94	1	1	1	1
4	U0001	여	43	미혼	전남	학생	2000	2002	B	71	86	91	93	1	1	1	1
5	U0001	여	43	미혼	전남	학생	2000	2002	B	83	91	91	80	1	1	1	1
6	U0001	여	43	미혼	전남	학생	2000	2002	B	83	76	96	95	1	1	1	1
7	U0001	여	43	미혼	전남	학생	2000	2002	B	78	99	99	72	1	1	1	1
8	U0001	여	43	미혼	전남	학생	2000	2002	B	90	76	99	87	1	1	1	1
9	U0001	여	43	미혼	전남	학생	2000	2002	B	93	83	88	88	1	1	1	1

Tip _ 랜덤 포레스트(Random Forest) 알고리즘

랜덤 포레스트는 학습을 통해 각각의 특징을 가지는 여러 개의 의사결정 트리를 앙상블로 구성하는 알고리즘이다. 단일 의사결정 트리와 달리 모델의 오버피팅을 최소화하면서 일반화 성능을 향상시킨 머신러닝 기법이다.

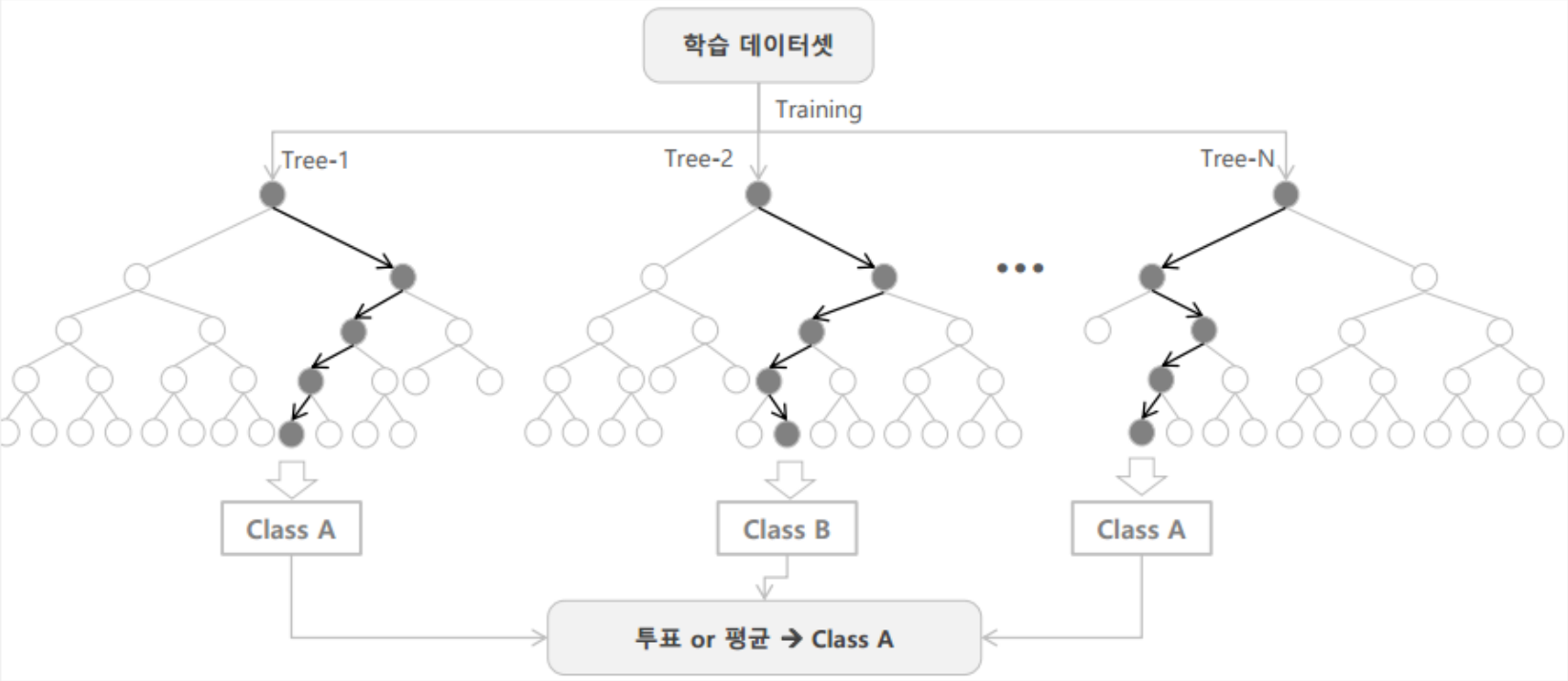
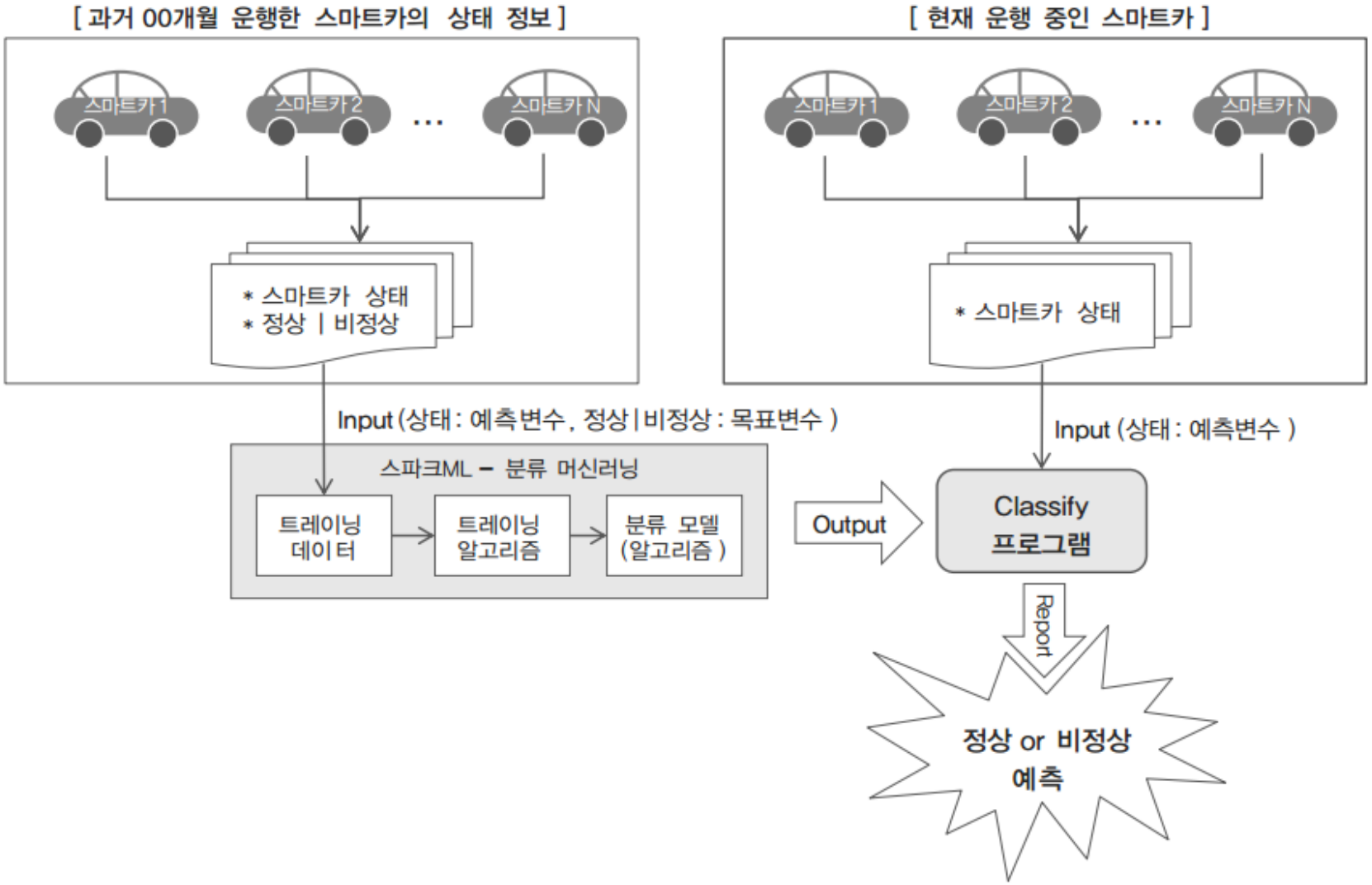


그림 7.58 랜덤 포레스트의 트리 구조

그림 7.58에서 보면 여러 의사결정 트리로부터 얻어진 결과를 모아 최종 분류값을 결정하는데 이때 평균을 이용하거나, 과반수 투표 방식 등을 이용한다.

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)



7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

01. 하이브를 이용해 트레이닝 데이터셋을 만드는 작업을 한다. 먼저 “스마트카 상태 정보” 데이터를 머하웃의 분류기의 입력 데이터로 사용하기 위해 하이브로 재구성한다. 휴의 Hive Editor에서 다음의 QL을 실행한다. C://예제소스/bigdata2nd-master/CH07/HiveQL/그림-7.60.hql에 실행할 하이브 QL이 있으니 복사해서 활용하도록 한다.

```
insert overwrite local directory '/home/pilot-pjt/spark-data/classification/input'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
select
  sex, age, marriage, region, job, car_capacity, car_year, car_model,
  tire_fl, tire_fr, tire_bl, tire_br, light_fl, light_fr, light_bl, light_br,
  engine, break, battery,
  case when ((tire_fl_s  + tire_fr_s  + tire_bl_s  + tire_br_s  +
              light_fl_s + light_fr_s + light_bl_s + light_br_s +
              engine_s   + break_s    + battery_s  +
              car_capacity_s + car_year_s + car_model_s) < 6)
      then '비정상' else '정상'
  end as status
from (
  select
    sex, age, marriage, region, job, car_capacity, car_year, car_model,
    tire_fl, tire_fr, tire_bl, tire_br, light_fl, light_fr, light_bl, light_br,
    engine, break, battery,

    case
     when (1500 > cast(car_capacity as int)) then -0.3
        when (2000 > cast(car_capacity as int)) then -0.2
        else -0.1
    end as car_capacity_s ,

    case
    when (2005 > cast(car_year as int)) then -0.3
       when (2010 > cast(car_year as int)) then -0.2
       else -0.1
    end as car_year_s ,

    case
    when ('B' = car_model) then -0.3
       when ('D' = car_model) then -0.3
       when ('F' = car_model) then -0.3
       when ('H' = car_model) then -0.3
       else 0.0
    end as car_model_s ,

    case
       when (10 > cast(tire_fl as int)) then 0.1
       when (20 > cast(tire_fl as int)) then 0.2
       when (40 > cast(tire_fl as int)) then 0.4
```

```
      else 0.5
    end as tire_fl_s ,

    case
      when (10 > cast(tire_fr as int)) then 0.1
      when (20 > cast(tire_fr as int)) then 0.2
      when (40 > cast(tire_fr as int)) then 0.4
      else 0.5
    end as tire_fr_s ,

    case
      when (10 > cast(tire_bl as int)) then 0.1
      when (20 > cast(tire_bl as int)) then 0.2
      when (40 > cast(tire_bl as int)) then 0.4
      else 0.5
    end as tire_bl_s ,

    case
      when (10 > cast(tire_br as int)) then 0.1
      when (20 > cast(tire_br as int)) then 0.2
      when (40 > cast(tire_br as int)) then 0.4
      else 0.5
    end as tire_br_s ,

    case when (cast(light_fl as int) = 2) then 0.0 else 0.5 end as light_fl_s ,
    case when (cast(light_fr as int) = 2) then 0.0 else 0.5 end as light_fr_s ,
    case when (cast(light_bl as int) = 2) then 0.0 else 0.5 end as light_bl_s ,
    case when (cast(light_br as int) = 2) then 0.0 else 0.5 end as light_br_s ,
```

```
72
73      case
74         when (engine = 'A') then 1.0
75         when (engine = 'B') then 0.5
76         when (engine = 'C') then 0.0
77      end as engine_s ,
78
79      case
80         when (break = 'A') then 1.0
81         when (break = 'B') then 0.5
82         when (break = 'C') then 0.0
83      end as break_s ,
84
85      case
86         when (20 > cast(battery as int)) then 0.2
87         when (40 > cast(battery as int)) then 0.4
88         when (60 > cast(battery as int)) then 0.6
89         else 1.0
90      end as battery_s
91
92   from managed_smartcar_status_info ) T1
```

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

02. 예측변수와 목표변수 값이 들어간 "스마트카 상태 정보" 입력 데이터셋이 정상적으로 만들어졌는지 확인한다. Server02에 SSH를 통해 접속하고 다음 명령을 실행해 내용을 확인한다.

```
$ more /home/pilot-pjt/spark-data/classification/input/*
```

```
남,27,미혼,충남,학생,3000,2010,B,80,91,95,84,1,1,1,1,B,A,84,정상
남,27,미혼,충남,학생,3000,2010,B,94,96,85,81,1,1,1,1,B,A,96,정상
남,27,미혼,충남,학생,3000,2010,B,97,80,99,89,1,1,1,1,A,A,87,정상
남,27,미혼,충남,학생,3000,2010,B,74,99,98,92,1,1,1,1,A,A,89,정상
남,30,미혼,대전,자영업,1500,2012,A,99,70,89,86,1,1,1,1,A,A,82,정상
남,30,미혼,대전,자영업,1500,2012,A,91,85,75,86,1,1,1,1,A,A,97,정상
남,30,미혼,대전,자영업,1500,2012,A,90,91,86,91,1,1,1,1,A,A,81,정상
남,30,미혼,대전,자영업,1500,2012,A,85,87,83,95,1,1,1,1,B,B,86,비정상
남,30,미혼,대전,자영업,1500,2012,A,97,92,95,93,1,1,1,1,A,A,85,정상
남,30,미혼,대전,자영업,1500,2012,A,82,76,73,81,1,1,1,1,A,A,96,정상
남,30,미혼,대전,자영업,1500,2012,A,92,80,87,90,1,1,1,1,A,A,86,정상
남,30,미혼,대전,자영업,1500,2012,A,96,90,96,86,1,1,1,1,A,A,80,정상
남,27,미혼,충남,학생,3000,2010,B,93,71,85,95,1,1,1,1,B,B,99,비정상
```

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

03. 분류기의 트레이닝 데이터셋을 만들기 위해 우선 두 개의 파일을 리눅스의 cat 명령을 이용해 하나의 파일로 합쳐 classification_dataset.txt라는 이름의 파일을 만든다.

```
$ cd /home/pilot-pjt/spark-data/classification/input
$ cat 000000_0 000001_0 > classification_dataset.txt
```

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

04. 스파크의 입력 데이터로 사용하기 위해 HDFS의 /pilot-pjt/spark-data/classification/input/ 경로를 생성하고 classification_dataset.txt 파일을 저장한다.

```
$ hdfs dfs -mkdir -p /pilot-pjt/spark-data/classification/input
$ hdfs dfs -put /home/pilot-pjt/spark-data/classification/input/classification_dataset.txt
/pilot-pjt/spark-data/classification/input
```

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

05. 스파크 머신러닝에 사용할 학습 데이터가 준비됐다. 파일럿 프로젝트에서는 스파크ML을 실행하기 위해 제플린을 활용한다. 제플린이 종료됐으면 다음 명령어를 통해 제플린 서버를 실행하고, 크롬 브라우저를 통해 제플린 웹IDE에 접속한다.

```
$ zeppelin-daemon.sh restart
```

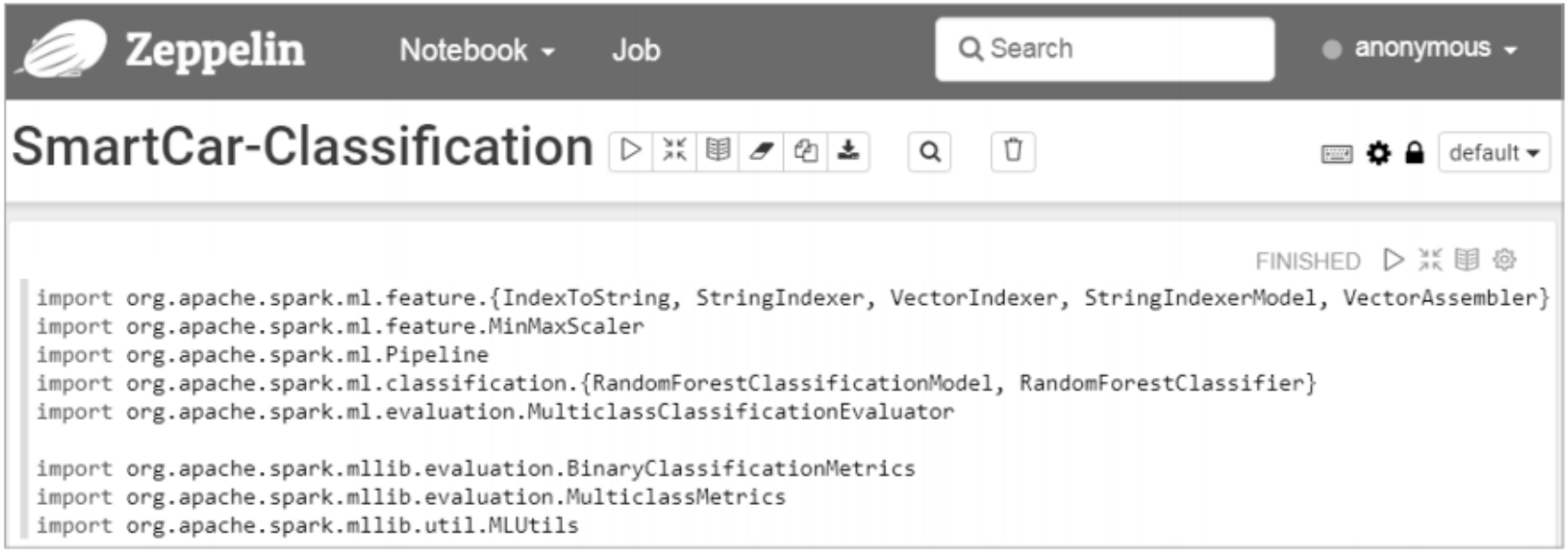
- 제플린 웹IDE URL: http://server02.hadoop.com:8081/

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

06. 제플린 상단 메뉴의 [Notebook] → [Create new note]를 선택하고 [Note Name]으로 "SmartCar-Classification"을 입력하고, [Default Interpreter]는 "spark"를 선택한 후 [Create] 버튼을 클릭한다.

07. 이제 스파크ML 프로그래밍을 시작한다. 스파크ML은 크게 자바, 파이썬, 스칼라로 개발할 수 있다. 파일럿 프로젝트에서는 스칼라를 이용하겠다. 첫 번째 입력창(Paragraph)에 사용할 라이브러리를 다음과 같이 입력하고 Shift+Enter 또는 우측 상단의 [Run] 버튼으로 실행한다. 관련 전체 소스코드는 C://예제소스/bigdata2nd-master/CH07/SparkML/SmartCar-Classification.scala를 참고한다.



7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

08. 하이브에서 생성해 둔 학습 데이터를 로드하고, 결과를 확인하기 위해 다음과 같이 코드를 입력한 후 실행 버튼을 클릭한다. 5개의 데이터가 조회된다.

```
val ds = spark.read.csv("/pilot-pjt/spark-data/classification/input/classification_dataset.txt")
ds.show(5)
```

```
+---+---+----+----+--------+----+----+---+---+---+----+----+----+----+----+----+----+----+----+
|_c0|_c1| _c2| _c3|     _c4| _c5| _c6|_c7|_c8|_c9|_c10|_c11|_c12|_c13|_c14|_c15|_c16|_c17|_c18|_c19|
+---+---+----+----+--------+----+----+---+---+---+----+----+----+----+----+----+----+----+----+----+
| 여| 62|미혼|충남|프리랜서|2500|2000|  A| 95| 76|  88|  95|   1|   1|   1|   1|   A|   A|  96|정상|
| 여| 62|미혼|충남|프리랜서|2500|2000|  A| 92| 95|  99|  82|   1|   1|   1|   1|   A|   A|  96|정상|
| 여| 62|미혼|충남|프리랜서|2500|2000|  A| 91| 85|  90|  93|   1|   1|   1|   1|   A|   B|  80|정상|
| 여| 62|미혼|충남|프리랜서|2500|2000|  A| 85| 72|  92|  90|   1|   1|   1|   1|   B|   A|  85|정상|
| 여| 62|미혼|충남|프리랜서|2500|2000|  A| 98| 85|  90|  88|   1|   1|   1|   1|   A|   A| 100|정상|
+---+---+----+----+--------+----+----+---+---+---+----+----+----+----+----+----+----+----+----+----+
only showing top 5 rows

ds: org.apache.spark.sql.DataFrame = [_c0: string, _c1: string ... 18 more fields]
```

Took 1 sec. Last updated by anonymous at April 14 2020, 5:47:52 PM. (outdated)

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

09. 다음 코드를 통해 이상징후 탐지 모델에 사용할 칼럼만 선택해 스파크 데이터셋을 새로 만든다.

```
val dsSmartCar = ds.selectExpr("cast(_c5 as long) car_capacity",
                               "cast(_c6 as long) car_year",
                               "cast(_c7 as string) car_model",
                               "cast(_c8 as int) tire_fl",
                               "cast(_c9 as long) tire_fr",
                               "cast(_c10 as long) tire_bl",
                               "cast(_c11 as long) tire_br",
                               "cast(_c12 as long) light_fl",
                               "cast(_c13 as long) light_fr",
                               "cast(_c14 as long) light_bl",
                               "cast(_c15 as long) light_br",
                               "cast(_c16 as string) engine",
                               "cast(_c17 as string) break",
                               "cast(_c18 as long) battery",
                               "cast(_c19 as string) status"
                              )
```

```
dsSmartCar: org.apache.spark.sql.DataFrame = [car_capacity: bigint, car_year: bigint ... 13 more fields]
```

Took 1 sec. Last updated by anonymous at April 14 2020, 5:56:19 PM.

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

10. 다음 작업을 통해 문자형 카테고리 칼럼을 숫자형 칼럼으로 생성하고, 기존 칼럼은 삭제한다.

FINISHED

```
val dsSmartCar_1 = new StringIndexer().setInputCol("car_model").setOutputCol("car_model_n")
                                      .fit(dsSmartCar).transform(dsSmartCar)
val dsSmartCar_2 = new StringIndexer().setInputCol("engine").setOutputCol("engine_n")
                                      .fit(dsSmartCar_1).transform(dsSmartCar_1)
val dsSmartCar_3 = new StringIndexer().setInputCol("break").setOutputCol("break_n")
                                      .fit(dsSmartCar_2).transform(dsSmartCar_2)
val dsSmartCar_4 = new StringIndexer().setInputCol("status").setOutputCol("label")
                                      .fit(dsSmartCar_3).transform(dsSmartCar_3)
val dsSmartCar_5 = dsSmartCar_4.drop("car_model").drop("engine").drop("break").drop("status")

dsSmartCar_5.show()
```

car_capacity	car_year	tire_fl	tire_fr	tire_bl	tire_br	light_fl	light_fr	light_bl	light_br	battery	car_model_n	engine_n	break_n	label
2500	2001	89	70	86	95	1	1	1	1	86	1.0	1.0	0.0	0.0
2500	2001	88	88	97	85	1	1	1	1	52	1.0	0.0	0.0	0.0
2500	2001	72	87	98	97	1	1	1	1	86	1.0	0.0	1.0	0.0
2500	2001	98	88	92	89	1	1	1	1	99	1.0	1.0	0.0	0.0
2500	2001	95	97	71	75	1	1	1	1	82	1.0	0.0	1.0	0.0
2500	2001	86	70	87	72	1	1	1	1	86	1.0	0.0	0.0	0.0
2500	2001	93	86	86	87	1	1	1	1	82	1.0	1.0	0.0	0.0
2500	2001	84	99	83	73	1	1	1	1	93	1.0	0.0	0.0	0.0
2500	2001	87	71	91	98	1	1	1	1	97	1.0	0.0	0.0	0.0
2500	2001	97	85	84	96	1	1	1	1	93	1.0	0.0	1.0	0.0
2500	2001	96	100	74	74	1	1	1	1	99	1.0	0.0	0.0	0.0
2500	2001	98	97	89	85	1	1	1	1	89	1.0	0.0	0.0	0.0
2500	2001	85	71	94	93	1	1	1	1	86	1.0	0.0	1.0	0.0
2500	2001	90	98	74	83	1	1	1	1	86	1.0	0.0	0.0	0.0

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

11. 머신러닝에 사용할 변수를 벡터화해서 feature라는 필드에 새로 생성하고 해당 값들에 대해 스케일링 작업도 진행하는 코드를 실행한다.

FINISHED

```
val cols = Array("car_capacity", "car_year", "car_model_n", "tire_fl",
                 "tire_fr", "tire_bl", "tire_br", "light_fl", "light_fr",
                 "light_bl", "light_br", "engine_n", "break_n", "battery")

val dsSmartCar_6 = new VectorAssembler().setInputCols(cols)
                                        .setOutputCol("features")
                                        .transform(dsSmartCar_5)
val dsSmartCar_7 = new MinMaxScaler().setInputCol("features")
                                     .setOutputCol("scaledFeatures")
                                     .fit(dsSmartCar_6)
                                     .transform(dsSmartCar_6)
val dsSmartCar_8 = dsSmartCar_7.drop("features")
                               .withColumnRenamed("scaledfeatures", "features")
dsSmartCar_8.show()
```

car_capacity	car_year	tire_fl	tire_fr	tire_bl	tire_br	light_fl	light_fr	light_bl	light_br	battery	car_model_n	engine_n	break_n	label	features
2500	2001	89	70	86	95	1	1	1	1	86	1.0	1.0	0.0	0.0	[0.6,0.0625,0.142...
2500	2001	88	88	97	85	1	1	1	1	52	1.0	0.0	0.0	0.0	[0.6,0.0625,0.142...
2500	2001	72	87	98	97	1	1	1	1	86	1.0	0.0	1.0	0.0	[0.6,0.0625,0.142...
2500	2001	98	88	92	89	1	1	1	1	99	1.0	1.0	0.0	0.0	[0.6,0.0625,0.142...
2500	2001	95	97	71	75	1	1	1	1	82	1.0	0.0	1.0	0.0	[0.6,0.0625,0.142...
2500	2001	86	70	87	72	1	1	1	1	86	1.0	0.0	0.0	0.0	[0.6,0.0625,0.142...
2500	2001	93	86	86	87	1	1	1	1	82	1.0	1.0	0.0	0.0	[0.6,0.0625,0.142...
2500	2001	84	99	83	73	1	1	1	1	93	1.0	0.0	0.0	0.0	[0.6,0.0625,0.142...
2500	2001	87	71	91	98	1	1	1	1	97	1.0	0.0	0.0	0.0	[0.6,0.0625,0.142...
2500	2001	97	85	84	96	1	1	1	1	93	1.0	0.0	1.0	0.0	[0.6,0.0625,0.142...
2500	2001	96	100	74	74	1	1	1	1	99	1.0	0.0	0.0	0.0	[0.6,0.0625,0.142...
2500	2001	98	97	89	85	1	1	1	1	89	1.0	0.0	0.0	0.0	[0.6,0.0625,0.142...
2500	2001	85	71	94	93	1	1	1	1	86	1.0	0.0	1.0	0.0	[0.6,0.0625,0.142...
2500	2001	90	98	74	83	1	1	1	1	86	1.0	0.0	0.0	0.0	[0.6,0.0625,0.142...
2500	2001	71	93	76	94	1	1	1	1	94	1.0	1.0	1.0	1.0	[0.6,0.0625,0.142...

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

12. 전처리 작업이 끝난 스파크 학습 데이터셋을 LibSVM 형식의 파일로 HDFS의 "/pilot-pjt/spark-data/classification/smartCarLibSvm" 경로에 저장한다.

FINISHED

```
var dsSmartCar_9 = dsSmartCar_8.select("label", "features")
dsSmartCar_9.write.format("libsvm").save("/pilot-pjt/spark-data/classification/smartCarLibSVM")
```

```
dsSmartCar_9: org.apache.spark.sql.DataFrame = [label: double, features: vector]
```

Took 2 min 3 sec. Last updated by anonymous at April 14 2020, 7:02:58 PM.

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

13. LibSVM 형식으로 학습 데이터가 잘 저장됐는지 확인하기 위해 휴의 좌측 상단에 있는 드롭다운 메뉴를 클릭하고 [브라우저] → [파일]을 선택해 파일 브라우저에서 "/pilot-pjt/spark-data/classification/smartCarLibSvm" 경로에 확장자가 .libsvm인 파일을 열어 본다. 그림 7.68에서 볼 수 있듯이 각 행마다 첫 번째 값이 레이블(0.0 – 정상, 1.0 – 비정상)이고, 그다음은 피처값으로 1~14번의 인덱스 번호가 붙은 스마트카 상태 데이터로서 1번의 car_capacity부터 14번의 battery 상태값으로 구성돼 있다.

/ pilot-pjt / spark-data / classification / smartCarLibSVM /
part-00000-8448f87f-a41c-4921-aaab-038ba7471d88-c000.libsvm

```
0.0 1:0.6 2:0.0 3:0.14285714285714285 4:0.946236559139785 5:0.7241379310344828 6:0.8356164383561644 7:0.9074074074074074 8:0.0 9:0.0
10:0.0 11:0.0 12:0.0 13:0.0 14:0.9565217391304348
0.0 1:0.6 2:0.0 3:0.14285714285714285 4:0.9139784946236559 5:0.9425287356321839 6:0.9863013698630136 7:0.6666666666666666 8:0.0 9:0.0
10:0.0 11:0.0 12:0.0 13:0.0 14:0.9565217391304348
0.0 1:0.6 2:0.0 3:0.14285714285714285 4:0.9032258064516129 5:0.8275862068965517 6:0.863013698630137 7:0.8703703703703703 8:0.0 9:0.0
10:0.0 11:0.0 12:0.0 13:0.5 14:0.782608695652174
0.0 1:0.6 2:0.0 3:0.14285714285714285 4:0.8387096774193549 5:0.6781609195402298 6:0.8904109589041096 7:0.8148148148148148 8:0.0 9:0.0
10:0.0 11:0.0 12:0.5 13:0.0 14:0.8369565217391305
0.0 1:0.6 2:0.0 3:0.14285714285714285 4:0.978494623655914 5:0.8275862068965517 6:0.863013698630137 7:0.7777777777777778 8:0.0 9:0.0 1
0:0.0 11:0.0 12:0.0 13:0.0 14:1.0
```

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

14. 제플린의 스파크ML 컨텍스트로 해당 파일을 다시 로드한다.

FINISHED

```
val dsSmartCar_10 = spark.read.format("libsvm").load("/pilot-pjt/spark-data/classification/smartCarLibSVM")
dsSmartCar_10.show(5)
```

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|(14,[0,2,3,4,5,6,...|
|  0.0|(14,[0,2,3,4,5,6,...|
|  0.0|(14,[0,2,3,4,5,6,...|
|  0.0|(14,[0,2,3,4,5,6,...|
|  0.0|(14,[0,2,3,4,5,6,...|
+-----+--------------------+
only showing top 5 rows

dsSmartCar_10: org.apache.spark.sql.DataFrame = [label: double, features: vector]
```

Took 36 sec. Last updated by anonymous at April 14 2020, 7:26:18 PM.

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

15. 레이블과 피처의 인덱서를 만들고, 전체 데이터셋을 학습(Training)과 테스트(Test) 데이터로 나누는 코드를 실행한다.

FINISHED

```
val labelIndexer = new StringIndexer().setInputCol("label").setOutputCol("indexedLabel").fit(dsSmartCar_10)
val featureIndexer = new VectorIndexer().setInputCol("features").setOutputCol("indexedFeatures").fit(dsSmartCar_10)

val Array(trainingData, testData) = dsSmartCar_10.randomSplit(Array(0.7, 0.3))
```

```
labelIndexer: org.apache.spark.ml.feature.StringIndexerModel = strIdx_4cf53f7fc598
featureIndexer: org.apache.spark.ml.feature.VectorIndexerModel = vecIdx_220067b0d461
trainingData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: double, features: vector]
testData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: double, features: vector]
```

Took 1 min 20 sec. Last updated by anonymous at April 14 2020, 7:32:32 PM.

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

16. 이제 랜덤 포레스트 머신러닝을 위한 파라미터를 설정한 후, 스파크ML 파이프라인을 만들고 Training 데이터셋으로 모델을 학습시킨다. 랜덤 포레스트의 모델은 파일럿 프로젝트 특성상 5개의 트리로만 만든다. 모델 학습에는 약 3~5분 정도의 시간이 걸린다.

저사양 파일럿 환경: 트리의 개수를 "5" → "3"으로 조정한다.

- setNumTrees(3)

FINISHED

```
val rf = new RandomForestClassifier()
  .setLabelCol("indexedLabel")
  .setFeaturesCol("indexedFeatures")
  .setNumTrees(5)

val labelConverter = new IndexToString()
  .setInputCol("prediction")
  .setOutputCol("predictedLabel")
  .setLabels(labelIndexer.labels)

val pipeline = new Pipeline().setStages(Array(labelIndexer, featureIndexer, rf, labelConverter))

val model = pipeline.fit(trainingData)
```

```
rf: org.apache.spark.ml.classification.RandomForestClassifier = rfc_bc903f27cac2
labelConverter: org.apache.spark.ml.feature.IndexToString = idxToStr_2aa94831fb2c
pipeline: org.apache.spark.ml.Pipeline = pipeline_d08041bd5dea
model: org.apache.spark.ml.PipelineModel = pipeline_d08041bd5dea
```

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

17. 모델 학습이 성공적으로 끝나면 다음 코드로 랜덤 포레스트 모델의 설명력을 확인해 본다. 총 5개의 트리가 만들어졌고 각 변수에 정의된 디시전 값을 확인할 수 있다.

FINISHED

```
val rfModel = model.stages(2).asInstanceOf[RandomForestClassificationModel]
println(s"RandomForest Model Description :\n ${rfModel.toDebugString}")
```

```
RandomForest Model Description :
 RandomForestClassificationModel (uid=rfc_bc903f27cac2) with 5 trees
  Tree 0 (weight 1.0):
    If (feature 12 in {1.0,2.0})
     If (feature 10 in {1.0})
      Predict: 1.0
     Else (feature 10 not in {1.0})
      If (feature 5 <= 0.5)
       If (feature 1 in {13.0})
        Predict: 0.0
       Else (feature 1 not in {13.0})
        Predict: 1.0
      Else (feature 5 > 0.5)
       If (feature 0 in {4.0,6.0,7.0})
        If (feature 2 in {0.0,1.0,3.0,4.0,5.0,7.0})
         Predict: 0.0
        Else (feature 2 not in {0.0,1.0,3.0,4.0,5.0,7.0})
         Predict: 1.0
```

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

18. 모델 학습이 성공적으로 끝나면 테스트 데이터로 모델의 정확도를 확인해 보기 위한 평가기를 실행한다.

FINISHED

```
val predictions = model.transform(testData)
predictions.select("predictedLabel", "label", "features").show(5)

val evaluator = new MulticlassClassificationEvaluator()
  .setLabelCol("indexedLabel")
  .setPredictionCol("prediction")
  .setMetricName("accuracy")

val accuracy = evaluator.evaluate(predictions)
```

```
+--------------+-----+--------------------+
|predictedLabel|label|            features|
+--------------+-----+--------------------+
|           0.0|  0.0|(14,[0,1,2,3,4,5,...|
|           0.0|  0.0|(14,[0,1,2,3,4,5,...|
|           0.0|  0.0|(14,[0,1,2,3,4,5,...|
|           0.0|  0.0|(14,[0,1,2,3,4,5,...|
|           0.0|  0.0|(14,[0,1,2,3,4,5,...|
+--------------+-----+--------------------+
only showing top 5 rows

predictions: org.apache.spark.sql.DataFrame = [label: double, features: vector ... 6 more fields]
evaluator: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = mcEval_494dac056ff4
accuracy: Double = 0.9216033707716176
```

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

19. 테스트 데이터로 예측 정확도를 확인해 본다. 필자의 모델은 정확도 92%로, 비교적 뛰어난 스마트카 상태 예측 모델이 만들어졌다.

```
println(s"@ Accuracy Rate = ${(accuracy)}")
println(s"@ Error Rate = ${(1.0 - accuracy)}")
```

FINISHED

```
@ Accuracy Rate = 0.9216033707716176
@ Error Rate = 0.07839662922838242
```

Took 1 sec. Last updated by anonymous at April 14 2020, 8:03:01 PM.

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

20. 다음 코드로 스마트카의 정상/비정상 예측에 대한 Confusion Matrix를 확인할 수 있다.

```
val results = model.transform(testData).select("features", "label", "prediction")
val predictionAndLabels = results.select($"prediction",$"label").as[(Double, Double)].rdd

val bMetrics = new BinaryClassificationMetrics(predictionAndLabels)
val mMetrics = new MulticlassMetrics(predictionAndLabels)
val labels = mMetrics.labels

println("Confusion Matrix:")
println(mMetrics.confusionMatrix)
```

FINISHED

```
Confusion Matrix:
441493.0  6241.0
41056.0   114514.0
```

- 정상을 정상으로 판단: 441,493건(정답 – True Negative)
- 정상을 비정상으로 판단: 6,241건(1종 오류 – False Positive)
- 비정상을 정상으로 판단: 41,056건(2종 오류 – False Negative)
- 비정상을 비정상으로 판단: 114,514건(정답 – True Positive)

		예측결과	
		정상 (Negative)	비정상 (Positive)
실제결과	정상 (Negative)	True Negative (441,493건)	False Positive (6,241건)
	비정상 (Positive)	False Negative (41,056건)	**True Positive** (114,514건)

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

- Precision(정밀도) – 모델이 비정상으로 분류한 스마트카 중에서 실제 비정상인 스마트카비율
- Recall(재현율) – 실제 비정상인 스마트카 중에서 모델이 비정상 스마트카로 분류한 비율
- F1-Score – Precision과 Recall의 조화 평균

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

21. 다음 코드로 Precision, Recall, F1-Score에 대한 평가 결과를 확인할 수 있다.

```
labels.foreach { rate =>
   println(s"@ Precision Rate($rate) = " + mMetrics.precision(rate))
}
```

FINISHED

```
@ Precision(0.0) = 0.9167611975147169
@ Precision(1.0) = 0.9648622822752343
```

Took 55 sec. Last updated by anonymous at April 14 2020, 4:52:41 PM. (outdated)

그림 7.77 스마트카 상태 예측 모델 평가 – Precision(정밀도)

```
labels.foreach { rate =>
   println(s"Recall Rate($rate) = " + mMetrics.recall(rate))
}
```

FINISHED

```
Recall Rate(0.0) = 0.9906402665023355
FPR(0.0) = 0.259242575372438
Recall Rate(1.0) = 0.740757424627562
FPR(1.0) = 0.009359733497664531
```

Took 1 sec. Last updated by anonymous at April 14 2020, 4:58:41 PM. (outdated)

그림 7.78 스마트카 상태 예측 모델 평가 – Recall(재현율)

7.7 분석 파일럿 실행 5단계 – 스파크 ML

4 스마트카 상태정보 예측(분류)

```
labels.foreach { rate =>
  println(s"F1-Score($rate) = " + mMetrics.fMeasure(rate))
}
```

FINISHED

```
F1-Score(0.0) = 0.9522699590597149
F1-Score(1.0) = 0.8380870559197953
```

Took 1 sec. Last updated by anonymous at April 14 2020, 5:05:52 PM.

그림 7.79 스마트카 상태 예측 모델 평가 – F1-Score

7.7 분석 파일럿 실행 5단계 – 스파크 ML

스마트카 상태정보 예측(분류)

실습