20220503

분석 파일럿 실행 5단계 - 머하웃과 스파크ML을 이용한 머신러닝

▼ 머하웃과 스파크ML 같은 머신러닝 기술은 복잡도가 높은 비즈니스 로직을 자동으로 생성 및 관리 하거나, 대규모 단순 반복 작업에서 패턴들을 찾아 효율화하는 데 사용 된다.

▼ 자동으로 만들어진 프로그램을 모델이라고 하며, 모델은 대규모 데이터에서 과거의 패턴을 찾아 정의하는 학습과정을 통해 만들어진다.

▼ 세 가지 마이닝 기법인 추천, 분류, 군집 기능을 머하웃(추천)과 스파크ML(분류, 군집)을 이용해 좀 더 활용성 있는 분석을 진행한다.

머하웃 추천

- 데이터 마이닝의 추천에 사용될 데이터셋 설정
- 이 가운데 추천에 필요한 항목을 선정한다.
- 유사 패턴을 보이는 사용자 간의 유사성을 계산하고, 그 결과로부터 유사 사용자 간의 선호하는 아이템을 예측해서 추천하는 것이 사용자 기반 협업 필터링 모델이다.
    - 예를 들면, 사용자 A와 B가 공통으로 구매했던 물건들의 평가 점수로 유사도를 측정, 유사한 사용자군으로 관명되면 A는 구매했으나 B는 구매하지 않은 물건을 B에게 추천하는 것이다.
- **머하웃의 추천기 모델을 만들고 실행**

1. 머하웃의 추천기에 사용 가능한 형식으로 재구성한 파일을 만들기
    a. 휴의 Hive Editor에서 다음 QL을 실행
    b. 경로 이동

```
insert overwrite local directory '/home/pilot-pit/mahout-data/recommendation/input'
Row format delimited
FIELDS TERMINATED BY ','
select hash(car_number),hash(item),score from managed_smartcar_item_buylist_info
```

1. hash() : 고유값으로 넘기기 위한 단방향 함수
2. xshell로 이동하고 명령어 입력
    a. 경로 이동

```
cd /home/pilot-pjt/mahout-data/recommendation/input
```

1. 확인(q누르면 종료)

```
more /home/pilot-pjt/mahout-data/recommendation/input/*
```

3. HDFS에 파일저장
    a. 파일 생성

```
hdfs dfs -mkdir -p /pilot-pjt/mahout/recommendation/input
```

1. 파일 적제

```
 hdfs dfs -put /home/pilot-pjt/mahout-data/recommendation/input/* /pilot-pjt/mahout/recommendation/input/item_buylist.txt
```

1. 추천기 실행

```
mahout recommenditembased -i /pilot-pjt/mahout/recommendation/input/item_buylist.txt -o /pilot-pjt/mahout/recommendation/
```

1. 분석 결과가 저장된 HDFS의 pilot-pit/mahout/recommendation/output/에 있는 파일을 휴의 파일 브라우저로 열어서 part-r-000000 생성된 것을 확인

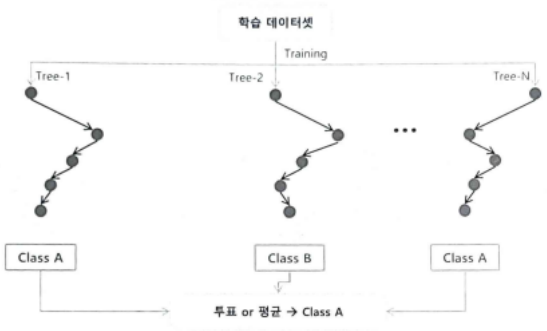
- 추천 분석을 재실행할 때는 기존 결과 파일을 삭제한 후 재실행해야 한다. 분류, 군집 분석에서도 같은 명령을 중복 실행할 때 이미 존재하는 파일(경로)이라는 에러가 발생할 수 있다. 그때는 해당 경로의 파일을 삭제한 후 재실행한다.
  - 삭제 명령

```
hdfs dfs -rm -R -skipTrash /pilot-pjt/mahout/recommendation/output
hdfs dfs -rm -R -skipTrash /user/root/temp
```

- **스파크ML 분류 - 정보 예측/분류**
  - 스파크ML은 제플린을 이용한다.
  - 분류 모델에 사용하는 알고리즘은 나이브 베이지안, 랜덤 포레스트, 로지스틱 회귀 등이 있다.

> 💡 랜덤 포레스트(Random Forest) 알고리즘

  - 학습을 통해 각각의 특징을 가진 여러 개의 의사결정 트리를 앙상블로 구성하는 알고리즘.
  - 단일 의사결정 트리와 달리 모델의 오버피팅을 최소화하면서 일반화 성능을 향상시킨 머신러닝 기법



  - 하이브를 이용해 트레이닝 데이터셋을 만드는 작업
  - Hive Editor에서 다음의 QL을 실행

```
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,

        case
            when (engine = 'A') then 1.0
            when (engine = 'B') then 0.5
            when (engine = 'C') then 0.0
        end as engine_s,

        case
            when (break = 'A') then 1.0
            when (break = 'B') then 0.5
            when (break = 'C') then 0.0
        end as break_s,

        case
            when (20 > cast(battery as int)) then 0.2
            when (40 > cast(battery as int)) then 0.4
            when (60 > cast(battery as int)) then 0.6
        end as battery_s

    from managed_smartcar_status_info) T1
```

- 이 하이브 QL의 실행 결과는 Server02의 /home/pilot-pjt/spark-data/calssification/input 로컬 디렉터리에 생성하게 된다.
- 데이터셋이 정상적으로 만들어졌는지 xshell을 이용하여 확인

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

```
ls /home/pilot-pjt/spark-data/classification/input/*
```

- 위 명령어를 이용해 파일 개수를 확인

- 데이터셋 만들기
  - 경로 이동

```
cd /home/pilot-pjt/spark-data/classification/input
```

  - 파일을 하나로 합치는 과정(> : 해당 파일을 읽어서 저장, >> : append의 역할)

```
cat 000000_0 000001_0 > classification_dataset.txt
```

- 스파크의 입력 데이터로 사용하기 위해 HDFS의 경로 생성 후 파일 저장
  - 디렉터리 생성

```
hdfs dfs -mkdir -p /pilot-pjt/spark-data/classification/input
```

  - 파일을 저장

```
hdfs dfs -put /home/pilot-pjt/spark-data/classifition_dataset.txt /pilot-pjt/spark-data/classification/input
```

- 제플린 재시작 및 웹DE 접속

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

  - 제플린 웹DE : server02.hadoop.com:8081
    - 새로운 노트북을 선택하고 “SmartCar-Classification”을 입력

```
import org.apache.spark.ml.feature.{IndexToString, StringIndexer, VectorIndexer, StringIndexerModel, VectorAssembler}
import org.apache.spark.ml.feature.MinMaxScaler
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.{RandomForestClassificationModel, RandomForestClassifier}
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.mllib.util.MLUtils
```

    - csv 파일로 데이터 로드

```
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|
+---+---+----+----+------+----+----+---+---+---+----+----+----+----+----+----+----+----+----+------+
| 남| 63|미혼|충남|전문직|3000|2004|  F| 96| 80|  88|  97|   1|   1|   1|   1|   A|   A|  87|  정상|
| 남| 63|미혼|충남|전문직|3000|2004|  F| 98| 99|  91|  88|   1|   1|   1|   1|   A|   B|  81|비정상|
| 남| 63|미혼|충남|전문직|3000|2004|  F| 88| 90|  78|  88|   1|   1|   1|   1|   B|   B|  92|비정상|
| 남| 63|미혼|충남|전문직|3000|2004|  F| 91| 81|  85|  92|   1|   1|   1|   1|   A|   A|  90|  정상|
| 남| 63|미혼|충남|전문직|3000|2004|  F| 92| 80|  92|  98|   1|   1|   1|   1|   B|   A|  65|비정상|
+---+---+----+----+------+----+----+---+---+---+----+----+----+----+----+----+----+----+----+------+
only showing top 5 rows

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

  - 이상징후 탐지 모델에 사용할 칼럼만 선택해 스파크 데이터셋을 새로 생성

```
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_f1",
                    "cast(_c13 as long) light_fr",
                    "cast(_c14 as long) light_b1",
                    "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"
                    )
```

- 문자형 카테고리 칼럼을 숫자형 칼럼으로 생성, 기존 칼럼 삭제

```
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() //기본값은 20
```

- 머신러닝에 사용할 변수를 백터화해서 feature라는 필드에 새로 생성, 해당 값들에 대해 스케일링 작업 진행

```
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(dsSma
val dsSmartCar_8 = dsSmartCar_7.drop("features").withColumnRenamed("scaledfeatures", "features")
dsSmartCar_8.show()
```

- 전처리 작업이 끝난 스파크 학습 데이터셋을 LibSVM 형식의 파일로 HDFS의 “pilot-pjt/spark-data/classification/smartCarLibSVM” 경로에 저장

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

- LibSVM형식으로 학습 데이터가 잘 저장됐는지 확인
  - 휴UI → 브라우저 → 파일
  - “/pliot-pjt/spark-data/calssification/smartCarLivSvm”경로에 확장자가 .libsvm인 파일 열기
- LibSVM 형식의 머신러닝 학습용 데이터 확인 및 로드

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

- 레이블과 피처의 인덱서를 만들고, 전체 데이터셋을 학습(Training)과 테스트(Test) 데이터로 나누는 코드 실행

```
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))
```

- 상태 정보 예측을 위한 랜덤 포레스트 모델 학습

```
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)
```

- 모델 설명력 확인

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

- 랜덤 포레스트 모델 평가기 실행

```
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)
```

- 테스트 데이터로 예측 정확도를 확인

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

- 스마트카의 정상/비정상 예측에 대한 Confusion Matrix를 확인

```
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)
```

```
Confusion Matrix:
464867.0  4435.0
68067.0   85349.0
```

  - 정상을 정상으로 판단 : 464,867건 (정답 - True Positive)
  - 비정상을 정상으로 판단 : 4,435건 (1종 오류 - False Positive) 오탐
  - 정상을 비정상으로 판단 : 68,067건 (2종 오류 - False Negative) 미탐
  - 비정상을 비정상으로 판단 : 85,349건 (정답 - True Negative)
- 스마트카 상태 예측 모델 평가 – Precision(정밀도)
  - Precision(정밀도) - 모델이 True라고 예측한 결과 중에서 실제 True인 결과 비율

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

- 스마트카 상태 예측 모델 평가 - Recall(재현율)
  - Recall(재현율) - 실제 True인 결과 중에서 모델이 True라고 예측한 결과 비율

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

- 스마트카 상태 예측 모델 평가 - F1-Score
  - F1-Score - Precision과 Recall의 조화 평균

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

마하웃과 스파크ML을 이용한 군집 - 스마트카 고객 분석 정보

- 군집분석은 이러한 속성 정보를 백터화하고 유사도 및 거리를 계산해 데이터의 새로운 군집을 발견하는 마이닝 기법이다.
- 군집분석은 RDBMS로는 분석이 어려울 만큼 대규모이면서 사람이 직관적으로 파악하기 어려운 데이터셋을 초기에 분석하는 데 자주 사용된다.
- 군집분석도 다양한 알고리즘을 선택적으로 적용할 수 있는데 Canopy, K-Means, Puzzy K-Means 등을 데이터의 특성에 맞게 적용한다.
- 머하웃을 이용한 Canopy 분석으로 대략적인 군집의 개수를 파악하고 스파크ML의 K-Means를 이용해 군집 분석을 진행

1. 휴의하이브 에디터로 데이터셋을 조회해서 로컬 디스크에 저장.
   a. 휴의 하이브 에디터에서 다음의 QL을 실행한다.

```
insert overwrite local directory '/home/pilot-pjt/mahout-data/clustering/input'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
select
  car_number,
  case
      when (car_capacity < 2000) then '소형'
      when (car_capacity < 3000) then '중형'
      when (car_capacity < 4000) then '대형'
  end as car_capacity,
  case
      when ((2016-car_year) <= 4)  then 'NEW'
      when ((2016-car_year) <= 8)  then 'NORMAL'
      else 'OLD'
  end as car_year ,
  car_model,
  sex as owner_sex,
  floor (cast(age as int) * 0.1 ) * 10 as owner_age,
  marriage as owner_marriage,
  job as owner_job,
  region as owner_region
from smartcar_master
```

   - 수행결과 : Server02의 /home/pilot-pjt/mahout-data/clustering/input 로컬 디렉터리 생성

2. 군집분석을 하기 위한 데이터셋이 정상적으로 만들어졌는지 xshell을 이용하여 확인

```
more /home/pilot=pjt/mahout-data/clustering/input/*
```

3. 파일 만들어져있는지 확인

```
ls -l /home/pilot-pjt/mahout-data/clustring/input
```

4. HDFS에 저장

```
hdfs dfs -mkdir -p /pilot-pjt/mahout/clustering/input
```

```
cd /home/pilot-pjt/mahout-data/clustering/input
```

```
mv 000000_0 smartcar_master.txt
```

```
hdfs dfs -put smartcar_master.txt /pilot-pjt/mahout/clustering/input
```

- “스마트카 사용자 마스터” 파일을 시퀀스 파일로 변환

```
hadoop jar /home/pilot-pjt/mahout-data/bigdata.smartcar.mahout-1.0.jar com.wikibook.bigdata.smartcar.mahout.TextToSequence /pilot-pjt/
```

- 시퀀스 파일의 내용을 확인하기 위해 다음의 HDFS 명령을 이용

```
hdfs dfs -text /pilot-pjt/mathout/clustering/output/seq/part-n-00000
```

- 데이터를 다차원의 공간 백터로 변환

```
mahout seq2sparse -i /pilot-pjt/mahout/clustering/output/seq -o /pilot-pjt/mahout/clustering/output/vec -wf tf -s 5 -md 3 -ng 2 -x 85
```

- Canopy 군집분석 실행

```
mahout canopy -i /pilot-pjt/mahout/clustering/output/vec/tf-vectors/ -o /pilot-pjt/mahout/clustering/canopy/out -dm org.apache.mahout.
```

- Camopy 군집분석 결과를 명령어로 확인

```
mahout clusterdump -i /pilot-pjt/mahout/clustering/canopy/out/clusters-*-final
```