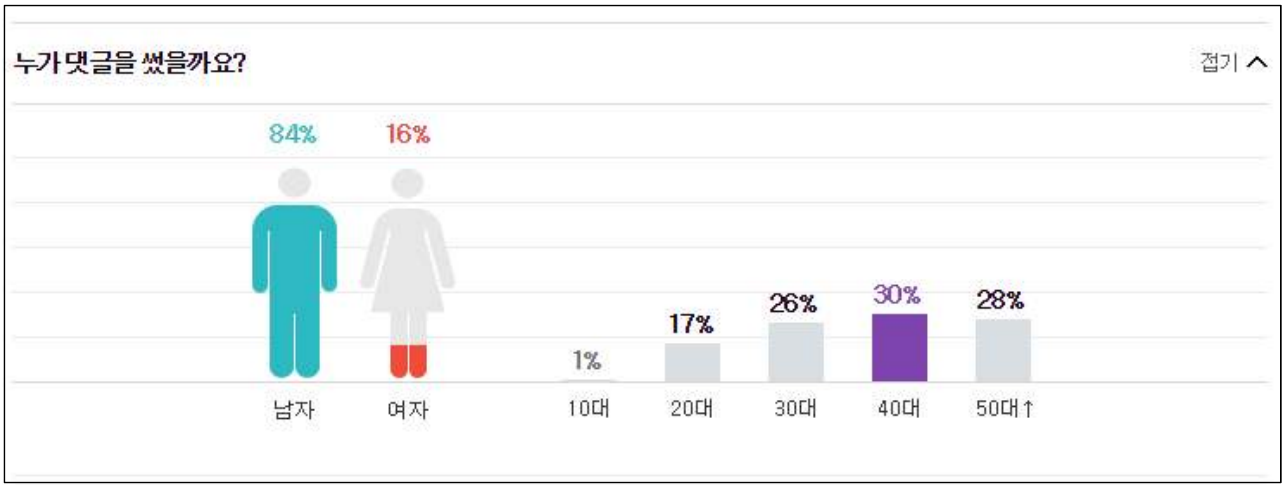
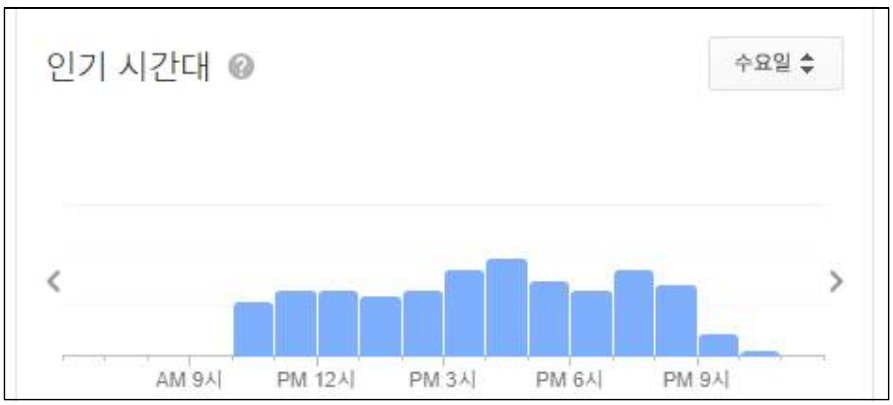
■ 머신러닝의 정의

1) 불과 몇 년 전까지만 해도 하둡으로 대표되는 빅데이터 작업은 데이터의 총합이나 평균, 분산과 같은 단순 통계량의 측정만을 목표로 삼는 경우가 많음(현재도 많이 사용함)





※ 참고 : 왓챠의 별점분포에 따른 키워드 모음

세상 영화들에 불만이 많으신 '개혁파' (1.1)
웬만해선 영화에 만족하지 않는 '헝그리파' (1.9)
별점을 대단히 짜게 주는 한줌의 '소금' 같은 분 :) (2.3~2.4)
웬만해서는 호평을 하지 않는 매서운 '독수리파' (2.5~2.7)
영화를 대단히 냉정하게 평가하는 '냉장고파' (2.9)
영화를 남들보다 진지하고 비판적으로 보는 '지성파' (3.1~3.2)
영화 평가에 상대적으로 깐깐한 '깐새우파' (3.3)
대체로 영화를 즐기지만 때론 혹평도 마다치 않는 '이성파' (3.4~3.5)
영화 평가에 있어 주관이 뚜렷한 '소나무파' (3.6)
대중의 평가에 잘 휘둘리지 않는 '지조파' (3.7)
편식 없이 영화를 골고루 보는 '균형파' (3.8)
영화를 정말로 즐길 줄 아는 '현명파' (3.9)
남들보다 별점을 조금 후하게 주는 '인심파' (4.0)
별점에 다소 관대한 경향이 있는 넉살 좋은 '이모' st. (4.1~4.2)
남 영화에 욕 잘 못하는 착한 품성의 '돌고래파'(4.3)
영화면 마냥 다 좋은 '천사급' 착한 사람♥ (4.4~4.5)
5점 뿌리는 '부처님급' 아량의 소유자 (4.7)

2) 머신러닝은 데이터를 학습하고, 데이터를 예측하는 알고리즘을 구성하고 연구하는 과학분야

3) 문제 영역 지식을 프로그램에 명시적으로 작성하는 대신, 확률 통계와 정보 이론 분야의 여러 기법을 사용해 데이터에 내재된 지식을 찾아냄

■ 머신러닝 개요

- 데이터를 이용해서 명시적으로 정의되지 않은 패턴을 컴퓨터로 학습하여 결과를 만들어 내는 학문 분야

1) 데이터

- 고전적인 인공지능 시스템에서는 여러 규칙을 단순 조합했지만 머신러닝은 항상 데이터가 기반

- 컴퓨터 알고리즘은 사용자가 어떻게 동작할지 정의를 내리지만 머신러닝은 알고리즘이 아닌, 데이터 학습을 통해 실행 동작이 바뀜
- 데이터를 기반으로 하는 점은 통계학과 비슷함

2) 패턴인식

- 통계학을 비롯해 딥러닝을 이용하여 데이터의 패턴을 유추
- 데이터를 보고 패턴을 유추하는 것이 머신러닝의 핵심
- 통계학이야말로 데이터에서 패턴을 찾아내는 학문으로 머신러닝의 기본이자 핵심적인 개념
- 딥러닝은 통계학의 전통적인 방법과는 다르지만 '패턴을 찾아내려고 학습한다'는 점에서 목표는 동일함

3) 컴퓨터를 이용한 계산

- 머신러닝은 데이터를 처리하고 패턴을 학습하고 계산하는데 컴퓨터를 사용
- 컴퓨터는 계산 속도를 높이고 많은 데이터를 효율적으로 다뤄야 함
- 예) 분산처리 시스템
- 머신러닝은 단순하게 수학적 모델 구축이나 증명에만 그치는 것이 아니라 실제 데이터를 계산해 결과를 만들어야 함

■ 머신러닝을 위한 수학지식

1) 선형대수

- 행렬, 행렬곱, 역행렬, 행렬 분해

2) 미분

- 최솟값/최댓값 개념
- 1차 미분
- 미적분학과 최적화

3) 통계학

- 분포
- 정상분포, 가우스 분포
- 상관관계
- 회귀

4) 확률

- 확률의 정의
- 조건부 확률

■ 머신러닝 역사

1) 고전적 인공지능 시대

- 엘런 튜링의 튜링 테스트
- 기계가 인간과 얼마나 비슷하게 대화하는 지를 기준으로 기계의 지능을 테스트

2) 신경망 시대

- 퍼센트론(perceptron)이라는 기초적인 신경망 개발(1957)
- 복잡한 신경망을 구성하면 입력과 출력을 유연하게 연결할 수 있음
- 하지만 구할 수 있는 데이터 양이 한정적이라 신경망 성능이 좋지 못함
- 기초 이론의 부족으로 한정적인 패턴만 학습이 가능
- 인공지능의 겨울

3) 통계학적 머신러닝 시대

- 통계학을 전산학과 접목시켜 대규모 데이터에서 패턴을 찾음(1990)
- 이전의 방법과 큰 차이점은 데이터에 비중을 준 것
- 기존의 방법들보다 성능이 뛰어남
- 이 시기에 머신러닝이라는 용어 등장

- 딥러닝이 나온 이후에는 통계학적 머신러닝이라고 명명

4) 빅데이터 시대

- 통계학적 머신러닝은 웹에서 나오는 데이터와 대용량 저장장치, 분산 처리 기술과 결합하여 큰 시너지를 냄
- 2010년부터 빅데이터 시대에는 기존보다 더 큰 데이터를 분석하기 위해 큰 규모의 머신러닝 시스템을 만들어 더 좋은 성능을 냄

5) 딥러닝 시대

- 데이터가 많아지고 GPU의 발전으로 컴퓨터의 연산 능력이 증가
- 이전의 신경망 시대보다 더 많은 데이터와 새로 개발된 이론을 합쳐 통계학적 머신러닝만 사용하는 모델을 넘어서는 결과가 나옴
- 딥러닝은 기존의 신경망보다 훨씬 더 복잡하고 깊이가 있음

■ 머신러닝의 작업 유형

1) 예측 모델링 : 주어진 예제(학습 데이터셋)로 예측 함수를 학습함

- 예측하려는 목표변수가 범주형 변수일 때는 분류라고 하고, 목표변수가 숫자의 형태일 때에는 회귀라고 함

2) 군집화 : 관측값 중 유사한 특징 변숫값을 가진 그룹, 즉 군집(cluster)을 도출함

3) 이상 탐지 : 정상이라고 생각하는 관측값과 현저하게 다른 관측값을 식별

4) 추천 시스템 : 다른 사용자들의 제품(아이템) 선호 이력 데이터를 기반으로 특정 사용자가 선호할 제품을 예측

5) 장바구니 분석 : 단일 관측값에서 동시에 자주 발견되는 아이템이나 변수

간의 연관관계를 도출함

■ 데이터 과학의 어려움

1) 성공적인 분석을 위한 작업의 대부분은 데이터 전처리 과정에서 이뤄짐

데이터를 활용하기 전에 정재, 개조, 합치기, 섞기 같은 여러가지 전처리 과정이 필요함

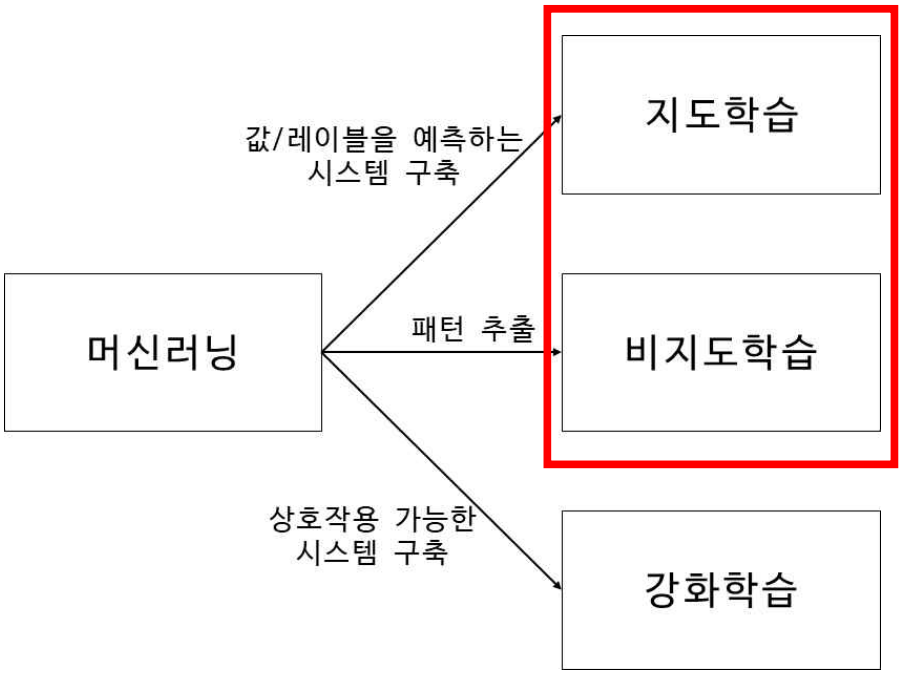
- 알고리즘을 선택해서 구현하는 일보다 '특징'을 추출하고 선택하는 데 더 많은 노력이 들어감

2) 데이터 과학에서 반복은 기본적인 과정

- 확률적 구배법, 기댓값 최대화법 등 널리 사용되는 최적화 기법들은 전부 반복에 의함. 데이터 과학자 입장에서 '모델'을 만들때 단 한 번의 시도로 모델이 완성되리라 기대해서는 안됨

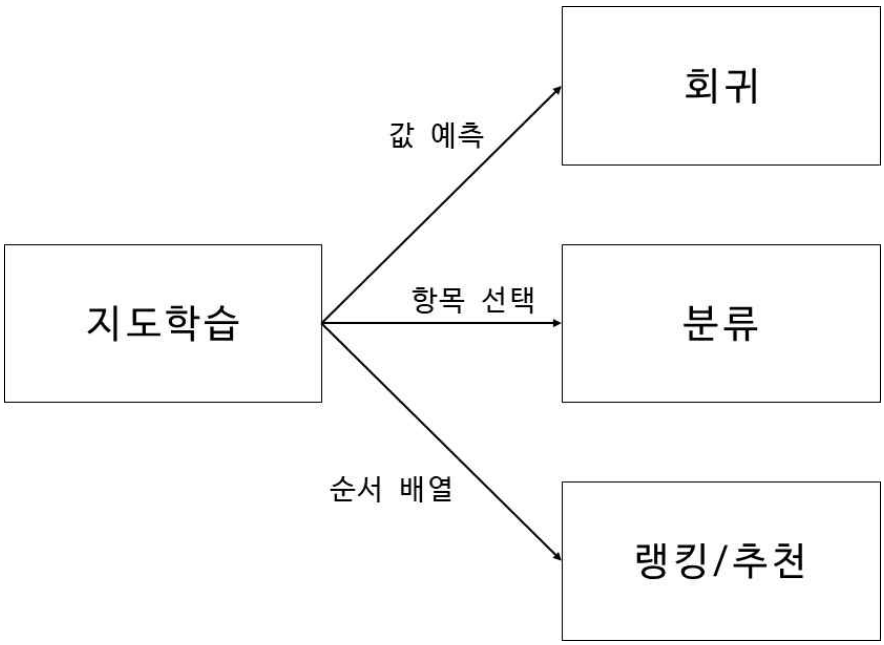
3) 잘 돌아가는 모델이 완성되었다고 하여 일이 끝나는 것이 아님. 추천등으로 활용하는 방안을 만들고, 주기적으로 혹은 실시간으로 모델을 다시 생성해야 할 수도 있음

■ 머신러닝 알고리즘의 유형



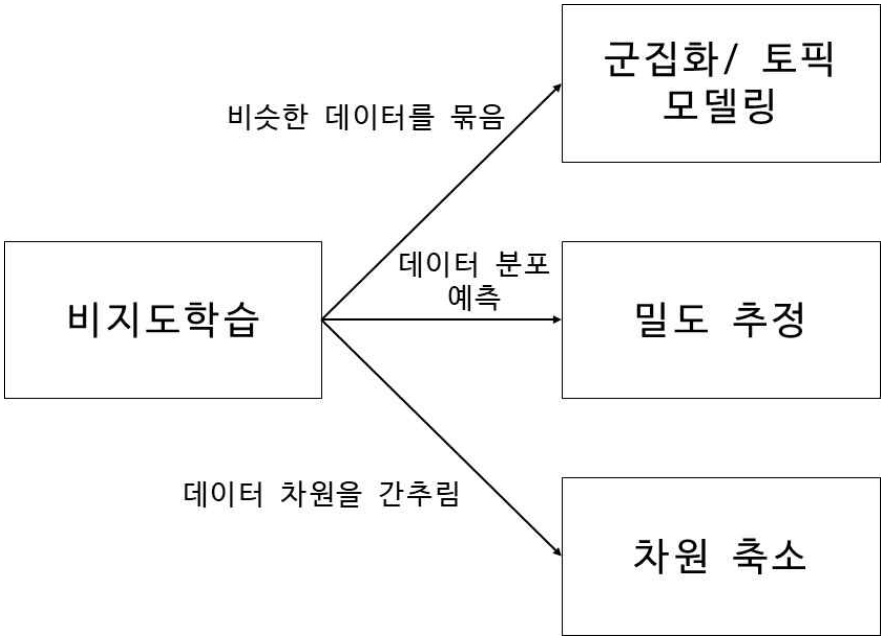
1) 지도 학습

- 지도 학습 : 예측 결과의 예상 정보, 즉 레이블이 포함된 데이텃 셋을 이용
- 스탬메일 탐지, 음성 및 필기 인식, 이미지 처리 등
- 크게 회귀와 분류, 랭킹/추천으로 나뉨



2) 비지도 학습

- 데이터의 숨겨진 구조를 탐색
- 군집화와 이상탐지, 이미지 세분화 등



■ 머신러닝의 기본 개념들

1) 관측과 특성

- 관측데이터의 속성을 나타내는 것 : 특성(feature)
- 동일한 데이터라도 데이터를 바라보는 관점이나 머신러닝의 목표에 따라서 최종적으로 사용되는 '특성'은 달라질 수 있음
- 원본 데이터를 변환 / 가공해서 머신러닝에 사용할 특성을 추출해내는 과정은 머신러닝을 수행하는 첫 단계이자 전체 작업의 성패를 결정할 수 있는 중요한 핵심 활동

2) 레이블

- 머신러닝은 학습 데이터셋을 이용해 학습 후 답이 알려지지 않은 새로운 입력값에 대한 출력값을 찾게 하는 방법에서 학습데이터의 정답이 '레이블'
- 레이블의 유형은 관측데이터셋과 수행하는 머신러닝의 종류에 따라 달라질 수 있음

3) 연속형 데이터와 이산형 데이터

- 연속형 데이터 : 무게 / 온도 / 습도등 연속적인 값을 가지는 데이터
- 이산형 데이터 : 나이 / 성별 / 갯수 등 불연속값의 데이터

4) 알고리즘과 모델

- 알고리즘과 모델을 혼용해서 쓰는 경우도 있으나 엄밀히 말하면 다름
- 학습은 알고리즘에 데이터를 적용하는 과정
- 모델은 학습의 결과물
- 모델은 클래스 또는 객체로 구현됨

5) 파라메트릭 알고리즘 / 넌파라메트릭 알고리즘

- 파라메트릭 알고리즘은 고정된 개수의 파라미터, 즉 계수를 사용하는 것으로 입력과 출력사이의 관계를 특성값에 관한 수학적 함수 또는 수식으로 가정하고, 이 수식의 결과가 실제 결괏값에 가깝도록 계수를 조정하는 방법을 사용
(선형 회귀, 로지스틱회귀 등)
- 넌파라메트릭 알고리즘은 입력과 출력 사이의 가설(함수)을 세우지 않고, 수행결과를 그대로 사용하는 방식
(서포티드 백터, 나이브 베이즈 등)

6) 지도 학습 / 비지도 학습

- 지도학습의 경우 훈련데이터에 '레이블', 즉 정답에 관한 정보가 포함됨
- 비지도학습의 경우 특성과 레이블간의 인과관계를 모르거나 지정하지 않고 컴퓨터의 처리에 맡기는 것

7) 훈련 데이터와 테스트 데이터

- 학습에 사용한 데이터와 테스트에 사용하는 데이터를 명확하게 구분하여 사용해야 함
- 교차검증 등 고도화된 기법으로 정확성 및 효율을 높일 수 있음

■ 벡터

- 벡터는 프로그램 상에서 double 타입의 값들을 포함하는 컬렉션으로 구현
- 벡터에 포함된 각 데이터는 정의된 순서에 따라 0부터 시작하는 정수형 인덱스를 부여받음
- 벡터 생성 예제

```
package org.jbm.spark_0627;

import org.apache.spark.mllib.regression.LabeledPoint;
import org.apache.spark.mllib.linalg.Vector;
import org.apache.spark.mllib.linalg.Vectors;
import org.apache.spark.mllib.util.MLUtils;
import org.apache.spark.rdd.RDD;
import org.apache.spark.sql.SparkSession;
import scala.Tuple2;

import java.util.Arrays;
import java.util.stream.Collectors;

public class VectorCreateApp1 {

  public static void main(String[] args) throws Exception {

    SparkSession spark = SparkSession.builder()
            .appName("Vector")
            .master("local[*]")
            .getOrCreate();

    //여러가지 방법으로 Vector 생성
    Vector v1 = Vectors.dense(0.1, 0.0, 0.2, 0.3);
    Vector v2 = Vectors.dense(new double[]{0.1, 0.0, 0.2, 0.3});
    Vector v3 = Vectors.sparse(4, Arrays.asList(new Tuple2(0, 0.1), new Tuple2(2, 0.2), new Tuple2(3,
0.3)));
    Vector v4 = Vectors.sparse(4, new int[]{0, 2, 3}, new double[]{0.1, 0.2, 0.3});

    System.out.println(Arrays.stream(v1.toArray())
            .mapToObj(String::valueOf).collect(Collectors.joining(", ")));

    System.out.println(Arrays.stream(v3.toArray())
            .mapToObj(String::valueOf).collect(Collectors.joining(", ")));

    //라벨(정답)이 붙은 Vector
    LabeledPoint v5 = new LabeledPoint(1.0, v1);
    System.out.println("label:" + v5.label() + ", features:" + v5.features());
```

```
    //파일에서 생성
    String path = "src/assets/sample_libsvm_data.txt";

    RDD<LabeledPoint> v6
            = MLUtils.loadLibSVMFile(spark.sparkContext(), path);
    LabeledPoint lp1 = v6.first();
    System.out.println("label:" + lp1.label() + ", features:" + lp1.features());

    spark.stop();
  }
}
```

- 결과

```
0.1, 0.0, 0.2, 0.3
0.1, 0.0, 0.2, 0.3
label:1.0, features:[0.1,0.0,0.2,0.3]
```

■ 파이프라인

- 머신러닝은 데이터 수집부터 가공, 특성 추출, 알고리즘 적용 및 모델 생성, 평가, 배포 및 활용에 이르는 일련의 작업을 반복하며 수행
- 이 방식은 머신러닝만의 특징은 아니고 하둡이나 스파크를 활용한 일반적인 빅데이터 처리 과정에서 볼 수 있는 처리 방법
- 파이프라인은 여러 종류의 알고리즘을 순차적으로 실행할 수 있게 지원하는 고차원 API
- 파이프 API를 이용해 머신러닝을 위한 워크플로우 생성할 수 있음
- 예제: 키와 몸무게, 나이 정보를 이용해 성별을 구분하는 예제를 파이프라인 API로 구현

```
package org.jbm.spark_0627;


import org.apache.spark.ml.Pipeline;
import org.apache.spark.ml.PipelineModel;
import org.apache.spark.ml.PipelineStage;
import org.apache.spark.ml.classification.LogisticRegression;
```

```
import org.apache.spark.ml.classification.LogisticRegressionModel;
import org.apache.spark.ml.feature.VectorAssembler;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.RowFactory;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.types.DataTypes;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;

import java.util.Arrays;
import java.util.List;

public class PipelineApp2 {

  public static void main(String[] args) throws Exception {

    SparkSession spark = SparkSession.builder()
            .appName("PipelineSample")
            .master("local[*]")
            .getOrCreate();

    StructField sf1 = DataTypes.createStructField("height", DataTypes.DoubleType, true);
    StructField sf2 = DataTypes.createStructField("weight", DataTypes.DoubleType, true);
    StructField sf3 = DataTypes.createStructField("age", DataTypes.IntegerType, true);
    StructField sf4 = DataTypes.createStructField("label", DataTypes.DoubleType, true);
    StructType schema1 = DataTypes.createStructType(Arrays.asList(sf1, sf2, sf3, sf4));

    List<Row> rows1 = Arrays.asList(RowFactory.create(161.0, 69.87, 29, 1.0),
            RowFactory.create(176.78, 74.35, 34, 1.0),
            RowFactory.create(159.23, 58.32, 29, 0.0));

    // 훈련용 데이터 (키, 몸무게, 나이, 성별)
    Dataset<Row> training = spark.createDataFrame(rows1, schema1);

    training.cache();

   // training.show();

   // test.show();

    VectorAssembler assembler = new VectorAssembler();
    assembler.setInputCols(new String[]{"height", "weight", "age"});
    assembler.setOutputCol("features");

    Dataset<Row> assembled_training = assembler.transform(training);

    assembled_training.show(false);
```

```
    // 모델 생성 알고리즘 (로지스틱 회귀 평가자)
    LogisticRegression lr = new LogisticRegression();
    lr.setMaxIter(10).setRegParam(0.01);

    // 모델 생성
    LogisticRegressionModel model = lr.fit(assembled_training);

    // 예측값 생성
    model.transform(assembled_training).show();

    // 파이프라인
    Pipeline pipeline = new Pipeline();
    pipeline.setStages(new PipelineStage[]{assembler, lr});

    // 파이프라인 모델 생성
    PipelineModel pipelineModel = pipeline.fit(training);

    // 파이프라인 모델을 이용한 예측값 생성
    pipelineModel.transform(training).show();

    String path1 = "src/assets/models/regression_model";
    String path2 = "src/assets/models/pipeline_model";

    // 모델 저장
    model.write().overwrite().save(path1);
    pipelineModel.write().overwrite().save(path2);

    spark.stop();
  }
}
```

- 예제: 모델 불러오기

```
package org.jbm.spark_0627;

import java.util.Arrays;
import java.util.List;

import org.apache.spark.ml.PipelineModel;
import org.apache.spark.ml.classification.LogisticRegressionModel;
import org.apache.spark.ml.feature.VectorAssembler;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.RowFactory;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.types.DataTypes;
```

```
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;

public class ModelLoadApp3 {

	public static void main(String[] args) {

		SparkSession spark = SparkSession.builder().appName("PipelineSample").master("local[*]").getOrCreate();

		// 모델 불러올 경로
		String path1 = "src/assets/models/regression_model";
		String path2 = "src/assets/models/pipeline_model";

		//로지스틱회귀모델
		LogisticRegressionModel loadedModel = LogisticRegressionModel.load(path1);
		//파이프라인모델
		PipelineModel loadedPipelineModel = PipelineModel.load(path2);

		StructField sf1 = DataTypes.createStructField("height", DataTypes.DoubleType, true);
		StructField sf2 = DataTypes.createStructField("weight", DataTypes.DoubleType, true);
		StructField sf3 = DataTypes.createStructField("age", DataTypes.IntegerType, true);

		List<Row> rows2 = Arrays.asList(RowFactory.create(169.4, 75.3, 42), RowFactory.create(185.1, 85.0, 37),
				RowFactory.create(161.6, 61.2, 28));

		StructType schema2 = DataTypes.createStructType(Arrays.asList(sf1, sf2, sf3));

		// 테스트용 데이터
		Dataset<Row> test = spark.createDataFrame(rows2, schema2);

		VectorAssembler assembler = new VectorAssembler();
		assembler.setInputCols(new String[] { "height", "weight", "age" });
		assembler.setOutputCol("features");

		Dataset<Row> assembled_training = assembler.transform(test);

		//예측값 출력
		loadedModel.transform(assembled_training).show();
		loadedPipelineModel.transform(test).show();

	}// main() end

}
```

- 결과

```
+------+------+---+----------------+--------------------+--------------------+----------+
|height|weight|age|        features|       rawPrediction|         probability|prediction|
+------+------+---+----------------+--------------------+--------------------+----------+
| 169.4|  75.3| 42|[169.4,75.3,42.0]|[-3.0855217899659...|[0.04370843319542...|       1.0|
| 185.1|  85.0| 37|[185.1,85.0,37.0]|[-4.8418050943384...|[0.00783098559170...|       1.0|
| 161.6|  61.2| 28|[161.6,61.2,28.0]|[1.56028508363870...|[0.82639425682108...|       0.0|
+------+------+---+----------------+--------------------+--------------------+----------+

+------+------+---+----------------+--------------------+--------------------+----------+
|height|weight|age|        features|       rawPrediction|         probability|prediction|
+------+------+---+----------------+--------------------+--------------------+----------+
| 169.4|  75.3| 42|[169.4,75.3,42.0]|[-3.0855217899660...|[0.04370843319542...|       1.0|
| 185.1|  85.0| 37|[185.1,85.0,37.0]|[-4.8418050943384...|[0.00783098559169...|       1.0|
| 161.6|  61.2| 28|[161.6,61.2,28.0]|[1.56028508363867...|[0.82639425682107...|       0.0|
+------+------+---+----------------+--------------------+--------------------+----------+
```

▲ 예측을 수행한 후 데이터프레임

▲ 1.0은 남자, 0.0은 여자

■ 알고리즘

- 스파크는 특성 추출, 변환, 선택을 위한 다양한 알고리즘 제공

1) Tokenizer

- 공백 문자를 기준으로 입력 문자열을 개별 단어의 배열로 변환하고 이 배열을 값으로 하는 새로운 칼럼을 생성하는 트랜스포머
- 문자열을 기반으로 하는 특성 처리에 자주 사용
- 문자열 구분자로 공백이 아닌 정규식을 사용하고자 할 경우 RegexTokenizer를 대신 사용할 수 있음
- 예제

```
package org.jbm.spark_0627;

import org.apache.spark.ml.feature.Tokenizer;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.RowFactory;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.types.DataTypes;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;
```

```
import java.util.Arrays;

public class TokenizerApp4 {

  public static void main(String[] args) throws Exception {

    SparkSession spark = SparkSession.builder()
            .appName("TokenizerSample")
            .master("local[*]")
            .getOrCreate();

    StructField sf1 = DataTypes.createStructField("input", DataTypes.StringType, true);
    StructType st1 = DataTypes.createStructType(Arrays.asList(sf1));
    Row r1 = RowFactory.create("나는 관대하다");
    Row r2 = RowFactory.create("리오넬 메시와 마르코스 로호의 골에 힘입은 아르헨티나가 극적으로 16강 진출에 성공했다");

    Dataset<Row> inputDF = spark.createDataFrame(Arrays.asList(r1, r2), st1);
    inputDF.printSchema();
    inputDF.show();

    Tokenizer tokenizer = new Tokenizer().setInputCol("input").setOutputCol("output");
    Dataset<Row> outputDF = tokenizer.transform(inputDF);
    outputDF.printSchema();
    outputDF.show(false);

    spark.stop();
  }
}
```

- 결과

```
root
 |-- input: string (nullable = true)
 |-- output: array (nullable = true)
 |    |-- element: string (containsNull = true)
```

```
+----------------------------------------+------------------------------------------------------------------+
|input                                   |output                                                            |
+----------------------------------------+------------------------------------------------------------------+
|나는 관대하다                          |[나는, 관대하다]                                                  |
|리오넬 메시와 마르코스 로호의 골에 힘입은 아르헨티나가 극적으로 16강 진출에 성공했다|[리오넬, 메시와, 마르코스, 로호의, 골에, 힘입은, 아르헨티나가, 극적으로, 16강, 진출에, 성공했다]|
+----------------------------------------+------------------------------------------------------------------+
```

▲ input 칼럼의 문자열을 분리한 결과에 해당하는 output 칼럼 확인 가능

2) TF-IDF

- TF-IDF(Term Frequency-Inverse Document Frequency)는 여러 문서 집합에서 특정 단어가 특정 문서 내에서 가지는 중요도를 수치화한 통계적 수치
- 문서 내에서 단어의 출현 빈도를 나타내는 TF(단어 빈도)와 문서군 내에서 출현 빈도를 나타내는 IDF(문서빈도, 빈도가 높을수록 점수가 낮아짐)의 조합으로 결정
- 문서 내에서 출현 빈도가 높은 단어일수록 높은 점수를 부여하되 특정 문서가 아닌 모든 문서에서 동일한 현상이 나타나면 흔하게 사용되는 중요하지 않은 단어로 간주해서 가중치를 낮춰줌
- 스파크 MLlib에서 TF-IDF 알고리즘은 TF 처리를 담당하는 부분과 IDF 처리를 담당하는 부분을 각각 따로 구현
- TF 처리에 해당하는 부분은 트랜스포머 클래스
- IDF 처리에 해당하는 부분은 평가자 클래스
- 예제: TF 트랜스 포머로 HashingTF 클래스를 사용

```
package org.jbm.spark_0627;

import org.apache.spark.ml.feature.IDF;
import org.apache.spark.ml.feature.IDFModel;
import org.apache.spark.ml.feature.Tokenizer;
import org.apache.spark.ml.feature.HashingTF;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.RowFactory;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.types.DataTypes;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;

import java.util.Arrays;

public class TfIDFApp5 {

  public static void main(String[] args) throws Exception {

    SparkSession spark = SparkSession.builder()
            .appName("TfIDFSample")
            .master("local[*]")
            .getOrCreate();
```

```
    StructField sf1 = DataTypes.createStructField("label", DataTypes.IntegerType, true);
    StructField sf2 = DataTypes.createStructField("sentence", DataTypes.StringType, true);
    StructType st1 = DataTypes.createStructType(Arrays.asList(sf1, sf2));

    Row r1 = RowFactory.create(0, "a a a b b c");
    Row r2 = RowFactory.create(0, "a b c");
    Row r3 = RowFactory.create(1, "a c a a d");

    Dataset<Row> df1 = spark.createDataFrame(Arrays.asList(r1, r2, r3), st1);

    Tokenizer tokenizer = new Tokenizer().setInputCol("sentence").setOutputCol("words");
    // 각 문장을 단어로 분리
    Dataset<Row> df2 = tokenizer.transform(df1);

    HashingTF hashingTF = new HashingTF()
            .setInputCol("words").setOutputCol("TF-Features").setNumFeatures(20);

    Dataset<Row> df3 = hashingTF.transform(df2);

    df3.cache();

    IDF idf = new IDF().setInputCol("TF-Features").setOutputCol("Final-Features");
    IDFModel idfModel = idf.fit(df3);

    Dataset<Row> rescaledData = idfModel.transform(df3);
    rescaledData.select("words", "TF-Features", "Final-Features").show(false);

    spark.stop();
  }
}
```

- 결과

```
+------------------+-------------------------+----------------------------------------+
|words             |TF-Features              |Final-Features                          |
+------------------+-------------------------+----------------------------------------+
|[a, a, a, b, b, c]|(20,[1,2,10],[2.0,1.0,3.0])|(20,[1,2,10],[0.5753641449035617,0.0,0.0])|
|[a, b, c]         |(20,[1,2,10],[1.0,1.0,1.0])|(20,[1,2,10],[0.28768207245178085,0.0,0.0])|
|[a, c, a, a, d]   |(20,[2,10,14],[1.0,3.0,1.0])|(20,[2,10,14],[0.0,0.0,0.6931471805599453])|
+------------------+-------------------------+----------------------------------------+
```

▲ "a a a b b c", "a b c", "a c a a d" 이 세 개의 문장을 이용해 각 문장 내에서 단어의 TF-IDF 값을 계산

▲ 최종 결과를 보면 가장 높은 빈도수를 가진 “a” 문자가 TF-Features에서 높은 점수를 받았다가 Final-Features에서 재조정된 것을 확인 가능

▲ 결과에서 맨 앞 숫자는 벡터의 크기, 두 번째 나오는 배열은 값의 위치(index),

마지막 배열은 값을 의미

■ 회귀

- 통계학이나 머신러닝 분야에서 회구분석의 목적은 변수 간의 관계를 찾는 것
- 먼저 변수 간의 관계에 대한 가설을 세운 뒤 이미 확보한 데이터와 최적화 알고리즘을 사용해 데이터셋에서 변수 간의 관계를 설명할 수 있는 최적화된 모델을 만듦
- 스파크에서 제공하는 최적화 알고리즘에는 선형 회귀(Linear regression), 일반화 선형 회귀(Generalized regression), 의사결정 트리 회귀(Decision tree regression), 랜덤 포레스트 회귀(Random forest regression), 그레디언트 부스티드 트리 회귀(Gradient-boosted tree regression), 생존 회귀(Survival regression), 등위 회귀(Isotonic regression) 등이 있음
- 예제: 스파크 선형회귀 알고리즘을 사용해 서울시 학생들의 키, 나이, 성별에 따른 몸무게를 예측

```
package org.jbm.spark_0627;

import org.apache.spark.ml.Pipeline;
import org.apache.spark.ml.PipelineModel;
import org.apache.spark.ml.PipelineStage;
import org.apache.spark.ml.clustering.KMeans;
import org.apache.spark.ml.clustering.KMeansModel;
import org.apache.spark.ml.evaluation.RegressionEvaluator;
import org.apache.spark.ml.feature.StringIndexer;
import org.apache.spark.ml.feature.VectorAssembler;
import org.apache.spark.ml.linalg.Vector;
import org.apache.spark.ml.regression.LinearRegression;
import org.apache.spark.ml.regression.LinearRegressionModel;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.api.java.UDF1;
import org.apache.spark.sql.types.DataTypes;

import static org.apache.spark.sql.functions.*;

import static org.apache.spark.sql.functions.collect_set;

public class RegressionApp7 {
```

```
public static void main(String[] args) throws Exception {

  SparkSession spark = SparkSession.builder()
          .appName("RegressionSample")
          .master("local[*]")
          .getOrCreate();

  // 데이터 제공처 : 서울시 학생 체격 현황 (키, 몸무게) 통계
  // (https://data.seoul.go.kr/dataList/datasetView.do?infId=10648&srvType=S&serviceKind=2)
  Dataset<Row> df1 = spark.read().option("header", "false")
          .option("sep", "\t")
          .option("timestampFormat", "yyyy")
          .option("inferSchema", true)
          .csv("src/assets/report.txt");



  df1.printSchema();
  df1.show(5, false);

  // Header 제거
  Dataset<Row> df2 = df1.where(df1.col("_c0").$eq$bang$eq("기간"));
  df2.show(3, false);

  spark.udf().register("toDouble", new UDF1<String, Double>() {
    @Override
    public Double call(String s) throws Exception {
      return Double.parseDouble(s.replaceAll("[^0-9.]", ""));
    }
  }, DataTypes.DoubleType);

  // cache
  df2.cache();

  // 초등학교 남 키, 몸무게
  Dataset<Row> df3 = df2.select(df2.col("_c0").as("year"),
          callUDF("toDouble", df2.col("_c2")).as("height"),
          callUDF("toDouble", df2.col("_c4")).as("weight"))
          .withColumn("grade", lit("elementary"))
          .withColumn("gender", lit("man"));

  // 초등학교 여 키, 몸무게
  Dataset<Row> df4 = df2.select(df2.col("_c0").as("year"),
          callUDF("toDouble", df2.col("_c3")).as("height"),
          callUDF("toDouble", df2.col("_c5")).as("weight"))
          .withColumn("grade", lit("elementary"))
          .withColumn("gender", lit("woman"));
```

```
// 중학교 남 키, 몸무게
Dataset<Row> df5 = df2.select(df2.col("_c0").as("year"),
        callUDF("toDouble", df2.col("_c6")).as("height"),
        callUDF("toDouble", df2.col("_c8")).as("weight"))
        .withColumn("grade", lit("middle"))
        .withColumn("gender", lit("man"));

// 중학교 여 키, 몸무게
Dataset<Row> df6 = df2.select(df2.col("_c0").as("year"),
        callUDF("toDouble", df2.col("_c7")).as("height"),
        callUDF("toDouble", df2.col("_c9")).as("weight"))
        .withColumn("grade", lit("middle"))
        .withColumn("gender", lit("woman"));

// 고등학교 남 키, 몸무게
Dataset<Row> df7 = df2.select(df2.col("_c0").as("year"),
        callUDF("toDouble", df2.col("_c10")).as("height"),
        callUDF("toDouble", df2.col("_c12")).as("weight"))
        .withColumn("grade", lit("high"))
        .withColumn("gender", lit("man"));

// 고등학교 여 키, 몸무게
Dataset<Row> df8 = df2.select(df2.col("_c0").as("year"),
        callUDF("toDouble", df2.col("_c11")).as("height"),
        callUDF("toDouble", df2.col("_c13")).as("weight"))
        .withColumn("grade", lit("high"))
        .withColumn("gender", lit("woman"));

Dataset<Row> df9 = df3.union(df4).union(df5).union(df6).union(df7).union(df8);

// 연도, 키, 몸무게, 학년, 성별
df9.show(5, false);
df9.printSchema();

// 문자열 컬럼을 double로 변환
StringIndexer gradeIndexer = new StringIndexer()
        .setInputCol("grade")
        .setOutputCol("gradecode");

StringIndexer genderIndexer = new StringIndexer()
        .setInputCol("gender")
        .setOutputCol("gendercode");

Dataset<Row> df10 = gradeIndexer.fit(df9).transform(df9);
Dataset<Row> df11 = genderIndexer.fit(df10).transform(df10);

df11.show(3, false);
```

```
    VectorAssembler assembler = new VectorAssembler()
            .setInputCols(new String[]{"height", "gradecode", "gendercode"})
            .setOutputCol("features");

    Dataset<Row> df12 = assembler.transform(df11);

    df12.show(false);

    Dataset<Row>[] dataArr = df12.randomSplit(new double[]{0.7, 0.3});
    Dataset<Row> training = dataArr[0];
    Dataset<Row> test = dataArr[1];

    LinearRegression lr = new LinearRegression()
            .setMaxIter(5)
            .setRegParam(0.3)
            .setLabelCol("weight")
            .setFeaturesCol("features");

    LinearRegressionModel model = lr.fit(training);

    System.out.println("결정계수(R2):" + model.summary().r2());

    Dataset<Row> d13 = model.setPredictionCol("predic_weight").transform(test);
    d13.cache();

    d13.select("weight", "predic_weight").show(5, false);

    RegressionEvaluator evaluator = new RegressionEvaluator();
    evaluator.setLabelCol("weight").setPredictionCol("predic_weight");

    // root mean squared error
    double rmse = evaluator.evaluate(d13);
    // mean squared error
    double mse = evaluator.setMetricName("mse").evaluate(d13);
    // R2 metric
    double r2 = evaluator.setMetricName("r2").evaluate(d13);
    // mean absolute error
    double mae = evaluator.setMetricName("mae").evaluate(d13);

    System.out.println("rmse:" + rmse + ", mse:" + mse + ", r2:" + r2 + ", mae:" + mae);

    // 파이프라인
    Pipeline pipeline = new Pipeline().setStages(new PipelineStage[]{gradeIndexer, genderIndexer,
assembler, lr});

    Dataset<Row>[] dataArr2 = df9.randomSplit(new double[]{0.7, 0.3});
    Dataset<Row> training2 = dataArr2[0];
    Dataset<Row> test2 = dataArr2[1];
```

```
    // 파이프라인 모델 생성
    PipelineModel pipelineModel = pipeline.fit(training2);

    // 파이프라인 모델을 이용한 예측값 생성
    pipelineModel.transform(test2)
            .select("weight", "prediction").show(5, false);

    spark.stop();
  }
}
```

- 결과

```
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)
 |-- _c6: string (nullable = true)
 |-- _c7: string (nullable = true)
 |-- _c8: string (nullable = true)
 |-- _c9: string (nullable = true)
 |-- _c10: string (nullable = true)
 |-- _c11: string (nullable = true)
 |-- _c12: string (nullable = true)
 |-- _c13: string (nullable = true)
```

▲ 컬럼 스키마

```
+----+---+-----+-----+----+----+-----+-----+----+----+-----+-----+----+----+
|_c0 |_c1|_c2  |_c3  |_c4 |_c5 |_c6  |_c7  |_c8 |_c9 |_c10 |_c11 |_c12|_c13|
+----+---+-----+-----+----+----+-----+-----+----+----+-----+-----+----+----+
|기간  |구분 |초등학교 |초등학교 |초등학교|초등학교|중학교  |중학교  |중학교 |중학교 |고등학교 |고등학교 |고등학교|고등학교|
|기간  |구분 |키    |키    |몸무게 |몸무게 |키    |키    |몸무게 |몸무게 |키    |키    |몸무게 |몸무게 |
|기간  |구분 |남자   |여자   |남자  |여자  |남자   |여자   |남자  |여자  |남자   |여자   |남자  |여자  |
|2001|구분 |149.6|150  |44.4|41.7|168.9|158.9|60.3|51.7|173.8|160.4|68.5|55.9|
|2002|구분 |149.3|150.7|46.1|44.1|169.2|159.8|63.5|53.9|173.7|161.9|68.4|56  |
+----+---+-----+-----+----+----+-----+-----+----+----+-----+-----+----+----+
only showing top 5 rows
```

▲ 첫 3개 행은 일종의 헤더에 해당하는 값

```
+----+---+-----+-----+----+----+-----+-----+----+----+-----+-----+----+----+
|_c0 |_c1|_c2  |_c3  |_c4 |_c5 |_c6  |_c7  |_c8 |_c9 |_c10 |_c11 |_c12|_c13|
+----+---+-----+-----+----+----+-----+-----+----+----+-----+-----+----+----+
|2001|구분 |149.6|150  |44.4|41.7|168.9|158.9|60.3|51.7|173.8|160.4|68.5|55.9|
|2002|구분 |149.3|150.7|46.1|44.1|169.2|159.8|63.5|53.9|173.7|161.9|68.4|56  |
|2003|구분 |150  |151.2|45  |44.1|168.6|159.2|60.4|53  |174.4|160.9|68.4|54.8|
+----+---+-----+-----+----+----+-----+-----+----+----+-----+-----+----+----+
only showing top 3 rows
```

▲ 헤더 행을 제거한 뒤 가공한 데이터프레임

```
+----+------+------+----------+------+
|year|height|weight|grade     |gender|
+----+------+------+----------+------+
|2001|149.6 |44.4  |elementary|man   |
|2002|149.3 |46.1  |elementary|man   |
|2003|150.0 |45.0  |elementary|man   |
|2004|149.8 |45.3  |elementary|man   |
|2005|150.8 |46.1  |elementary|man   |
+----+------+------+----------+------+
only showing top 5 rows
```

```
root
 |-- year: string (nullable = true)
 |-- height: double (nullable = true)
 |-- weight: double (nullable = true)
 |-- grade: string (nullable = false)
 |-- gender: string (nullable = false)
```

▲ 전체 데이터가 모두 포함된 데이터프레임으로부터 연도별, 학년별, 성별 데이터를 뽑아 만든 데이터프레임

```
+----+------+------+----------+------+---------+----------+
|year|height|weight|grade     |gender|gradecode|gendercode|
+----+------+------+----------+------+---------+----------+
|2001|149.6 |44.4  |elementary|man   |2.0      |1.0       |
|2002|149.3 |46.1  |elementary|man   |2.0      |1.0       |
|2003|150.0 |45.0  |elementary|man   |2.0      |1.0       |
+----+------+------+----------+------+---------+----------+
only showing top 3 rows
```

```
+----+------+------+----------+------+---------+----------+--------------+
|year|height|weight|grade     |gender|gradecode|gendercode|features      |
+----+------+------+----------+------+---------+----------+--------------+
|2001|149.6 |44.4  |elementary|man   |2.0      |1.0       |[149.6,2.0,1.0]|
|2002|149.3 |46.1  |elementary|man   |2.0      |1.0       |[149.3,2.0,1.0]|
|2003|150.0 |45.0  |elementary|man   |2.0      |1.0       |[150.0,2.0,1.0]|
|2004|149.8 |45.3  |elementary|man   |2.0      |1.0       |[149.8,2.0,1.0]|
|2005|150.8 |46.1  |elementary|man   |2.0      |1.0       |[150.8,2.0,1.0]|
|2006|150.5 |45.4  |elementary|man   |2.0      |1.0       |[150.5,2.0,1.0]|
|2007|151.6 |47.7  |elementary|man   |2.0      |1.0       |[151.6,2.0,1.0]|
|2008|151.6 |47.6  |elementary|man   |2.0      |1.0       |[151.6,2.0,1.0]|
|2009|151.5 |46.9  |elementary|man   |2.0      |1.0       |[151.5,2.0,1.0]|
|2010|150.6 |46.4  |elementary|man   |2.0      |1.0       |[150.6,2.0,1.0]|
|2011|151.0 |46.8  |elementary|man   |2.0      |1.0       |[151.0,2.0,1.0]|
|2012|151.8 |47.3  |elementary|man   |2.0      |1.0       |[151.8,2.0,1.0]|
|2013|150.5 |45.5  |elementary|man   |2.0      |1.0       |[150.5,2.0,1.0]|
|2014|152.0 |47.1  |elementary|man   |2.0      |1.0       |[152.0,2.0,1.0]|
|2015|152.2 |47.5  |elementary|man   |2.0      |1.0       |[152.2,2.0,1.0]|
|2016|151.7 |47.0  |elementary|man   |2.0      |1.0       |[151.7,2.0,1.0]|
|2001|150.0 |41.7  |elementary|woman |2.0      |0.0       |[150.0,2.0,0.0]|
|2002|150.7 |44.1  |elementary|woman |2.0      |0.0       |[150.7,2.0,0.0]|
|2003|151.2 |44.1  |elementary|woman |2.0      |0.0       |[151.2,2.0,0.0]|
|2004|150.1 |42.9  |elementary|woman |2.0      |0.0       |[150.1,2.0,0.0]|
+----+------+------+----------+------+---------+----------+--------------+
only showing top 20 rows
```

▲ 성별과 학년을 나타내는 문자열 칼럼에 대응하는 숫자형 칼럼 생성 (StringIndexer 사용)

```
결정계수(R2):0.9810890356311008
```

▲ 결정계수 출력

```
+------+------------------+
|weight|predic_weight     |
+------+------------------+
|44.4  |44.962906263364815|
|45.3  |45.15659233154081 |
|47.6  |46.89976694512474 |
|46.4  |45.93133660424478 |
|47.1  |47.28713908147675 |
+------+------------------+
only showing top 5 rows
```

▲ 생성된 모델을 이용한 실제 예측 결과

```
rmse:1.2641754999852093, mse:1.5981396947628537, r2:0.9771385911944471, mae:1.0305719050854458
```

▲ 예측 후 결과에 대한 평가

```
+------+------------------+
|weight|prediction        |
+------+------------------+
|46.1  |45.60938176487799 |
|45.0  |46.10542635850782 |
|45.5  |46.45974392538627 |
|47.5  |47.66442365277302 |
|41.7  |43.37749035428961 |
+------+------------------+
only showing top 5 rows
```

▲ 파이프라인 모델을 이용한 예측 값

■ 분류

- 분류는 특정 데이터를 사전에 정해진 기준에 따라 몇 개의 카테고리로 분류
- 군집(Clustering)과 달리 레이블에 해당하는 카테고리 정보가 포함된 입력 데이터를 사용
- 지도학습에 포함됨
- 스파크에서는 로지스틱 회귀(Logistic regression), 의사결정 트리(Decision tree), 랜덤 포레스트(Random forest), 그레디언트 부스티드 트리(Gradient-boosted tree), 다중 퍼셉트론(Multilayer perceptron), 선형 SVM(Linear Support Vector Machine), One-vs-Rest, 나이브 베이즈(Naive Bayes) 등 다양한 분류 알고리즘 제공
- 예제: 의사결정 트리 알고리즘을 이용해 서울시 주요 도시도로의 교통 흐름을 “원활” 또는 “지연”으로 분류하는 모델 구현

```
package org.jbm.spark_0627;

import org.apache.commons.lang3.StringUtils;
import org.apache.spark.api.java.function.Function2;
import org.apache.spark.ml.Pipeline;
import org.apache.spark.ml.PipelineModel;
import org.apache.spark.ml.PipelineStage;
import org.apache.spark.ml.classification.DecisionTreeClassificationModel;
import org.apache.spark.ml.classification.DecisionTreeClassifier;
import org.apache.spark.ml.classification.LogisticRegression;
import org.apache.spark.ml.classification.LogisticRegressionModel;
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator;
import org.apache.spark.ml.feature.StringIndexer;
import org.apache.spark.ml.feature.VectorAssembler;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.RowFactory;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.api.java.UDF1;
import org.apache.spark.sql.api.java.UDF2;
import org.apache.spark.sql.types.DataTypes;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;
import scala.Function0;
import scala.collection.JavaConversions;
import scala.collection.Seq;
```

```
import static org.apache.spark.sql.functions.*;

import java.util.Arrays;
import java.util.List;

public class ClassficationApp8 {

  public static void main(String[] args) throws Exception {

    SparkSession spark = SparkSession.builder()
            .appName("ClassficationSample")
            .master("local[*]")
            .getOrCreate();

    spark.udf().register("label", new UDF2<Double, Double, Double>() {
      @Override
      public Double call(Double avr_month, Double avr_total) throws Exception {
        return ((avr_month - avr_total) >= 0) ? 1.0d : 0.0d;
      }
    }, DataTypes.DoubleType);

    // 원본데이터
    /                                                                                                                    /
https://data.seoul.go.kr/dataList/datasetView.do?infId=OA-2604&srvType=S&serviceKind=1&currentPageNo=1
    // 서울시 도시고속도로 월간 소통 통계
    Dataset<Row> d1 = spark.read().option("header", "true")
            .option("sep", ",").option("inferSchema", true)
            .option("mode", "DROPMALFORMED")
            .option("timestampFormat", "yyyy")
            .csv("src/assets/seoul_road.csv");

    Dataset<Row> d2 = d1.toDF("year", "month", "road", "avr_traffic_month", "avr_velo_month", "mon",
"tue", "wed", "thu", "fri", "sat", "sun");

    // data 확인
    d2.printSchema();

    // null 값 제거
    Dataset<Row> d3 = d2.where("avr_velo_month is not null");

    // 도로별 평균 속도
    Dataset<Row> d4 = d3.groupBy("road")
            .agg(round(avg("avr_velo_month"), 1).as("avr_velo_total"))
            .select(col("road").as("groad"), col("avr_velo_total"));

    Dataset<Row> d5 = d3.join(d4, d3.col("road").equalTo(d4.col("groad")));

    // label 부여
```

```
    Dataset<Row>     d6     =     d5.withColumn("label",     callUDF("label",     d5.col("avr_velo_month"),
d5.col("avr_velo_total")));
    d6.select("road", "avr_velo_month", "avr_velo_total", "label").show(5, false);
    d6.groupBy("label").count().show(false);

    Dataset<Row>[] samples = d6.randomSplit(new double[]{0.7, 0.3});

    Dataset<Row> train = samples[0];
    Dataset<Row> test = samples[1];

    StringIndexer indexer = new StringIndexer().setInputCol("road").setOutputCol("roadcode");

    VectorAssembler assembler = new VectorAssembler()
            .setInputCols(new String[]{"roadcode", "mon", "tue", "wed", "thu", "fri", "sat", "sun"})
            .setOutputCol("features");

    DecisionTreeClassifier dt = new DecisionTreeClassifier()
            .setLabelCol("label")
            .setFeaturesCol("features");

    Pipeline pipeline = new Pipeline().setStages(new PipelineStage[]{indexer, assembler, dt});

    PipelineModel model = pipeline.fit(train);

    Dataset<Row> predict = model.transform(test);

    predict.select("label", "probability", "prediction").show(3, false);

    // areaUnderROC, areaUnderPR
    BinaryClassificationEvaluator evaluator = new BinaryClassificationEvaluator()
            .setLabelCol("label")
            .setMetricName("areaUnderROC");

    System.out.println(evaluator.evaluate(predict));

    DecisionTreeClassificationModel treeModel = ((DecisionTreeClassificationModel) model.stages()[2]);
    System.out.println("Learned classification tree model:\n" + treeModel.toDebugString());

    spark.stop();
  }
}
```

- 결과

```
root
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- road: string (nullable = true)
 |-- avr_traffic_month: integer (nullable = true)
 |-- avr_velo_month: double (nullable = true)
 |-- mon: double (nullable = true)
 |-- tue: double (nullable = true)
 |-- wed: double (nullable = true)
 |-- thu: double (nullable = true)
 |-- fri: double (nullable = true)
 |-- sat: double (nullable = true)
 |-- sun: double (nullable = true)
```

▲ 컬럼 스키마

```
+------+-------------+-------------+-----+
|road  |avr_velo_month|avr_velo_total|label|
+------+-------------+-------------+-----+
|동부간선도로|52.5          |54.9          |0.0  |
|내부순환로 |60.3          |63.4          |0.0  |
|분당수서로 |59.2          |61.3          |0.0  |
|경부고속도로|42.3          |44.7          |0.0  |
|올림픽대로 |56.6          |59.0          |0.0  |
+------+-------------+-------------+-----+
only showing top 5 rows
```

▲ 데이터 출력

```
+-----+-----+
|label|count|
+-----+-----+
|0.0  |136  |
|1.0  |132  |
+-----+-----+
```

▲ 0.0: 원활, 1.0: 혼잡

```
BinaryClassificationEvaluator evaluator = new BinaryClassificationEvaluator()
        .setLabelCol("label")
        .setMetricName("areaUnderROC");

System.out.println(evaluator.evaluate(predict));
```

```
0.8527472527472527
```

▲ 예측 결과의 정확도

```
DecisionTreeClassificationModel treeModel = ((DecisionTreeClassificationModel) model.stages()[2]);
System.out.println("Learned classification tree model:\n" + treeModel.toDebugString());
```

```
Learned classification tree model:
DecisionTreeClassificationModel (uid=dtc_bb0e2dc3dc8b) of depth 5 with 25 nodes
  If (feature 4 <= 65.15)
   If (feature 0 in {2.0,3.0,6.0,8.0})
    If (feature 3 <= 61.2)
     Predict: 0.0
    Else (feature 3 > 61.2)
     If (feature 6 <= 63.95)
      If (feature 0 in {2.0,6.0})
       Predict: 0.0
      Else (feature 0 not in {2.0,6.0})
       Predict: 1.0
```

▲ 분류 트리모델 출력

■ 클러스터링

- 레이블을 사용하는 지도학습과는 달리 데이터 간의 유사도만을 이용해 각 데이터를 유사 그룹으로 분류하는 방법
- 스파크에서는 K-평균(K-means), LDA(Latent Dirichlet allocation), Eisecting K-평균(Bisecting K-means), 가우시안 혼합 모델(GMM, Gaussian Mixture Model) 알고리즘 제공
- 예제: K-means 알고리즘을 이용한 클러스터링

```
package org.jbm.spark_0627;


import org.apache.spark.ml.Pipeline;
import org.apache.spark.ml.PipelineModel;
```

```
import org.apache.spark.ml.PipelineStage;
import org.apache.spark.ml.classification.DecisionTreeClassificationModel;
import org.apache.spark.ml.classification.DecisionTreeClassifier;
import org.apache.spark.ml.clustering.KMeans;
import org.apache.spark.ml.clustering.KMeansModel;
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator;
import org.apache.spark.ml.feature.StringIndexer;
import org.apache.spark.ml.feature.VectorAssembler;
import org.apache.spark.ml.linalg.Vector;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.api.java.UDF2;
import org.apache.spark.sql.types.DataTypes;
import scala.collection.JavaConversions;
import scala.collection.Seq;

import java.util.Arrays;

import static org.apache.spark.sql.functions.*;

public class ClusteringApp9 {

  public static void main(String[] args) throws Exception {

    SparkSession spark = SparkSession.builder()
            .appName("ClusteringSample")
            .master("local[*]")
            .getOrCreate();

    // 원본데이터
    /                                                                                                                    /
https://data.seoul.go.kr/dataList/datasetView.do?infId=OA-13061&srvType=S&serviceKind=1&currentPageNo=1
    // 서울시 공공와이파이 위치정보 (영어)
    Dataset<Row> d1 = spark.read().option("header", "true")
            .option("sep", ",")
            .option("inferSchema", true)
            .option("timestampFormat", "yyyy/MM/dd HH:mm:ss ZZ")
            .option("mode", "DROPMALFORMED")
            .csv("src/assets/seoul_wifi.csv");

    d1.printSchema();

    Dataset<Row> d2 = d1.toDF("number", "name", "SI", "GOO", "DONG", "x", "y", "b_code", "h_code",
"utmk_x", "utmk_y", "wtm_x", "wtm_y");

    Dataset<Row> d3 = d2.select(d2.col("GOO").as("loc"), d2.col("x"), d2.col("y"));
    d3.show(5, false);
```

```
    StringIndexer indexer = new StringIndexer().setInputCol("loc").setOutputCol("loccode");

    VectorAssembler assembler = new VectorAssembler()
            .setInputCols(new String[]{"loccode", "x", "y"})
            .setOutputCol("features");

    KMeans kmeans = new KMeans().setK(5).setSeed(1L).setFeaturesCol("features");

    Pipeline pipeline = new Pipeline().setStages(new PipelineStage[]{indexer, assembler, kmeans});

    PipelineModel model = pipeline.fit(d3);

    Dataset<Row> d4 = model.transform(d3);

    d4.groupBy("prediction").agg(collect_set("loc").as("loc"))
            .orderBy("prediction").show(100, false);

    double WSSSE = ((KMeansModel) model.stages()[2]).computeCost(d4);
    System.out.println("Within Set Sum of Squared Errors = " + WSSSE);

    // Shows the result.
    System.out.println("Cluster Centers: ");

    for (Vector v : ((KMeansModel) model.stages()[2]).clusterCenters()) {
      System.out.println(v);
    }

    spark.stop();
  }
}
```

- 결과

prediction	loc
0	[Gangseo-gu, Jongno-gu, Eunpyeong-gu, Gangbuk-gu, Yongsan-gu]
1	[Geumcheon-gu, Gwacheon-si, Songpa-gu, Gwanak-gu, Dobong-gu, Jungnang-gu, Gangdong-gu]
2	[Yangcheon-gu, Dongdaemun-gu, Seodaemun-gu, Yeongdeungpo-gu]
3	[Seongbuk-gu, Seongdong-gu, Seocho-gu, Dongjak-gu, Mapo-gu, Guro-gu]
4	[Jung-gu, Nowon-gu, Gangnam-gu, Gwangjin-gu]

▲ 0번 클러스터에 강서구, 동대문구, 종로구 등이, 1번 클러스터에는 금천구, 도봉구, 강동구 등이 포함된 것을 확인 가능

```
Within Set Sum of Squared Errors = 3968.2126412260473
```

```
Cluster Centers:
18/06/27 11:27:57 INFO BlockManagerInfo: Removed broadcast_
[5.949561403508771,126.95013749451752,37.58109566811401]
[21.35126582278481,127.0346014764304,37.53881882070065]
[10.435897435897436,126.94107998767326,37.54890696613428]
[15.428947368421053,126.96865751102624,37.533038317921054]
[1.2930056710775049,127.04247152835552,37.562662111739094]
```

▲ 파이프라인 모델의 stages() 메서드를 이용해 원래의 모델의 정보를 찾아서 해당 모델 인스턴스가 제공하는 WSSSE(Within Set Sum of Squared Errors)값과 각 클러스터의 정보등을 확인 가능