In [1]:
```
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
from pyspark.sql.types import *

spark = SparkSession\
    .builder\
    .appName("chapter-24-ML")\
    .getOrCreate()

import os
SPARK_BOOK_DATA_PATH = os.environ['SPARK_BOOK_DATA_PATH']
```

```
### Vector
```

In [2]:
```
from pyspark.ml.linalg import Vectors
denseVec = Vectors.dense(1.0, 2.0, 3.0)
```

In [3]:
```
denseVec
```

Out[3]:
```
DenseVector([1.0, 2.0, 3.0])
```

In [4]:
```
denseVec.array
```

Out[4]:
```
array([1., 2., 3.])
```

In [5]:
```
denseVec.values
```

Out[5]:
```
array([1., 2., 3.])
```

In [6]:
```
size = 3
idx = [1, 2] # locations of non-zero elements in vector
values = [2.0, 3.0]
sparseVec = Vectors.sparse(size, idx, values)
```

In [7]:
```
sparseVec
```

Out[7]:
```
SparseVector(3, {1: 2.0, 2: 3.0})
```

In [8]:
```
sparseVec.values
```

Out[8]:
```
array([2., 3.])
```

In [9]:
```
# COMMAND ----------

df = spark.read.json(SPARK_BOOK_DATA_PATH + "/data/simple-ml")
```

In [10]:
```
df.count()
```

Out[10]:
```
110
```

In [11]:
```
1 df.printSchema()
```

```
root
 |-- color: string (nullable = true)
 |-- lab: string (nullable = true)
 |-- value1: long (nullable = true)
 |-- value2: double (nullable = true)
```

In [12]:
```
1 df.show(3)
```

```
+-----+----+------+------------------+
|color| lab|value1|            value2|
+-----+----+------+------------------+
|green|good|     1|14.386294994851129|
| blue| bad|     8|14.386294994851129|
| blue| bad|    12|14.386294994851129|
+-----+----+------+------------------+
only showing top 3 rows
```

In [13]:
```
1 df.orderBy("value1").show(10)
```

```
+-----+----+------+------------------+
|color| lab|value1|            value2|
+-----+----+------+------------------+
|green|good|     1|14.386294994851129|
|green|good|     1|14.386294994851129|
|  red| bad|     1| 38.97187133755819|
|green|good|     1|14.386294994851129|
|  red| bad|     1| 38.97187133755819|
|  red| bad|     1| 38.97187133755819|
|  red| bad|     1| 38.97187133755819|
|green|good|     1|14.386294994851129|
|  red| bad|     1| 38.97187133755819|
|green|good|     1|14.386294994851129|
+-----+----+------+------------------+
only showing top 10 rows
```

In [14]:
```
1 df.groupBy("color", "lab").count()\
2     .orderBy("color", "lab")\
3     .show(10)
```

```
+-----+----+-----+
|color| lab|count|
+-----+----+-----+
| blue| bad|   20|
|green| bad|   10|
|green|good|   30|
|  red| bad|   30|
|  red|good|   20|
+-----+----+-----+
```

In [20]:
```
# COMMAND ----------

from pyspark.ml.feature import RFormula
supervised = RFormula(formula="lab ~ . + color:value1 + color:value2
```

In [23]:
```
# COMMAND ----------

## prepare feature columns

fittedRF = supervised.fit(df)
preparedDF = fittedRF.transform(df)
preparedDF.show(10, False)
```

```
+-----+----+------+------------------+-------------------------------
-------------------------------------+-----+
|color|lab |value1|value2            |features
|label|
+-----+----+------+------------------+-------------------------------
-------------------------------------+-----+
|green|good|1     |14.386294994851129|(10,[1,2,3,5,8],[1.0,1.0,14.3862
94994851129,1.0,14.386294994851129])  |1.0  |
|blue |bad |8     |14.386294994851129|(10,[2,3,6,9],[8.0,14.3862949948
51129,8.0,14.386294994851129])        |0.0  |
|blue |bad |12    |14.386294994851129|(10,[2,3,6,9],[12.0,14.386294994
851129,12.0,14.386294994851129])      |0.0  |
|green|good|15    |38.97187133755819 |(10,[1,2,3,5,8],[1.0,15.0,38.971
87133755819,15.0,38.97187133755819])  |1.0  |
|green|good|12    |14.386294994851129|(10,[1,2,3,5,8],[1.0,12.0,14.386
294994851129,12.0,14.386294994851129])|1.0  |
|green|bad |16    |14.386294994851129|(10,[1,2,3,5,8],[1.0,16.0,14.386
294994851129,16.0,14.386294994851129])|0.0  |
|red  |good|35    |14.386294994851129|(10,[0,2,3,4,7],[1.0,35.0,14.386
294994851129,35.0,14.386294994851129])|1.0  |
|red  |bad |1     |38.97187133755819 |(10,[0,2,3,4,7],[1.0,1.0,38.9718
7133755819,1.0,38.97187133755819])    |0.0  |
|red  |bad |2     |14.386294994851129|(10,[0,2,3,4,7],[1.0,2.0,14.3862
94994851129,2.0,14.386294994851129])  |0.0  |
|red  |bad |16    |14.386294994851129|(10,[0,2,3,4,7],[1.0,16.0,14.386
294994851129,16.0,14.386294994851129])|0.0  |
+-----+----+------+------------------+-------------------------------
-------------------------------------+-----+
only showing top 10 rows
```

In [24]:
```
# COMMAND ----------

## split train/test

train, test = preparedDF.randomSplit([0.7, 0.3])
```

In [25]:

```
# COMMAND ----------

## create model

from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(labelCol="label",featuresCol="features")
```

In [26]:

```
# COMMAND ----------

print (lr.explainParams())
```

```
aggregationDepth: suggested depth for treeAggregate (>= 2). (default:
2)
elasticNetParam: the ElasticNet mixing parameter, in range [0, 1]. For
alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 pe
nalty. (default: 0.0)
family: The name of family which is a description of the label distrib
ution to be used in the model. Supported options: auto, binomial, mult
inomial (default: auto)
featuresCol: features column name. (default: features, current: featur
es)
fitIntercept: whether to fit an intercept term. (default: True)
labelCol: label column name. (default: label, current: label)
lowerBoundsOnCoefficients: The lower bounds on coefficients if fitting
under bound constrained optimization. The bound matrix must be compati
ble with the shape (1, number of features) for binomial regression, or
(number of classes, number of features) for multinomial regression. (u
ndefined)
lowerBoundsOnIntercepts: The lower bounds on intercepts if fitting und
er bound constrained optimization. The bounds vector size must beequal
with 1 for binomial regression, or the number oflasses for multinomial
regression. (undefined)
maxIter: max number of iterations (>= 0). (default: 100)
predictionCol: prediction column name. (default: prediction)
probabilityCol: Column name for predicted class conditional probabilit
ies. Note: Not all models output well-calibrated probability estimate
s! These probabilities should be treated as confidences, not precise p
robabilities. (default: probability)
rawPredictionCol: raw prediction (a.k.a. confidence) column name. (def
ault: rawPrediction)
regParam: regularization parameter (>= 0). (default: 0.0)
standardization: whether to standardize the training features before f
itting the model. (default: True)
threshold: Threshold in binary classification prediction, in range [0,
1]. If threshold and thresholds are both set, they must match.e.g. if
threshold is p, then thresholds must be equal to [1-p, p]. (default:
0.5)
thresholds: Thresholds in multi-class classification to adjust the pro
bability of predicting each class. Array must have length equal to the
number of classes, with values > 0, excepting that at most one value m
ay be 0. The class with largest value p/t is predicted, where p is the
original probability of that class and t is the class's threshold. (un
defined)
tol: the convergence tolerance for iterative algorithms (>= 0). (defau
lt: 1e-06)
upperBoundsOnCoefficients: The upper bounds on coefficients if fitting
under bound constrained optimization. The bound matrix must be compati
ble with the shape (1, number of features) for binomial regression, or
(number of classes, number of features) for multinomial regression. (u
ndefined)
upperBoundsOnIntercepts: The upper bounds on intercepts if fitting und
er bound constrained optimization. The bound vector size must be equal
with 1 for binomial regression, or the number of classes for multinomi
al regression. (undefined)
```

```
weightCol: weight column name. If this is not set or empty, we treat a
ll instance weights as 1.0. (undefined)
```

In [27]:
```
# COMMAND ----------

## train model

fittedLR = lr.fit(train)
```

In [28]:
```
fittedLR
```

Out[28]:
```
LogisticRegressionModel: uid = LogisticRegression_5b6ada8feb48, numCla
sses = 2, numFeatures = 10
```

In [29]:
```
# COMMAND ----------

train, test = df.randomSplit([0.7, 0.3])
```

In [30]:
```
df.show(3,False)
```

```
+-----+----+------+------------------+
|color|lab |value1|value2            |
+-----+----+------+------------------+
|green|good|1     |14.386294994851129|
|blue |bad |8     |14.386294994851129|
|blue |bad |12    |14.386294994851129|
+-----+----+------+------------------+
only showing top 3 rows
```

In [31]:
```
# COMMAND ----------

rForm = RFormula()
lr = LogisticRegression().setLabelCol("label").setFeaturesCol("featu
```

In [32]:
```
# COMMAND ----------

from pyspark.ml import Pipeline
stages = [rForm, lr]
pipeline = Pipeline().setStages(stages)
```

In [33]:
```
# COMMAND ----------

from pyspark.ml.tuning import ParamGridBuilder
params = ParamGridBuilder()\
  .addGrid(rForm.formula, [
    "lab ~ . + color:value1",
    "lab ~ . + color:value1 + color:value2"])\
  .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0])\
  .addGrid(lr.regParam, [0.1, 2.0])\
  .build()
```

In [34]:
```
# COMMAND ----------

from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()\
  .setMetricName("areaUnderROC")\
  .setRawPredictionCol("prediction")\
  .setLabelCol("label")
```

In [35]:
```
# COMMAND ----------

from pyspark.ml.tuning import TrainValidationSplit
tvs = TrainValidationSplit()\
  .setTrainRatio(0.75)\
  .setEstimatorParamMaps(params)\
  .setEstimator(pipeline)\
  .setEvaluator(evaluator)
```

In [36]:
```
# COMMAND ----------

tvsFitted = tvs.fit(train)
```

In [37]:
```
type(tvsFitted)
```

Out[37]:
```
pyspark.ml.tuning.TrainValidationSplitModel
```

In [ ]:
```

```