Table of contents

Spark MLlib provides a set of regression algorithms

- Linear regression
- Decision tree regression
- Random forest regression
- Survival regression
- Isotonic regression

A regression algorithm is used to predict the value of a continuous attribute (the target attribute) by applying a model on the predictive attributes. The model is trained on a set of training data (i.e., a set of data for which the value of the target attribute is know), and it is applied on new data to predict the target attribute.

The regression algorithms available in Spark work only on numerical data. They work similarly to classification algorithms, but they predict continuous numerical values (the target attribute is a continuous numerical attribute).

The input data must be transformed in a DataFrame having the following attributes:

- label: the continuous numerical double value to be predicted
- features: the double vector with the predictive features.

The main steps used to infer a regression model with MLlib are the same we use to infer a classification model, and the difference is only given by the type of the target attribute to predict.

1 Linear regression

Linear regression (LR) is a popular, effective and efficient regression algorithm. The following paragraphs show how to instantiate a linear regression algorithm in Spark and apply it on new data.

1.1 LR with structured data

The input dataset is a structured dataset with a fixed number of attributes

- One attribute is the target attribute (the label): it is assumed that the first column contains the target attribute;
- The others are predictive attributes that are used to predict the value of the target attribute.

Example

Consider the following example file

```
label,attr1,attr2,attr3
2.0,0.0,1.1,0.1
5.0,2.0,1.0,-1.0
5.0,2.0,1.3,1.0
2.0,0.0,1.2,-0.5
```

Each record has three predictive attributes and the target attribute

- The first attribute ("label") is the target attribute;
- The other attributes ("attr1", "attr2", "attr3") are the predictive attributes.

```
from pyspark.mllib.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml import PipelineModel

# input and output folders
trainingData = "ex_dataregression/trainingData.csv"
unlabeledData = "ex_dataregression/unlabeledData.csv"
outputPath = "predictionsLinearRegressionPipeline/"

# ************************
# Training step
# ************************
# Create a DataFrame from trainingData.csv
# Training data in raw format
trainingData = spark.read.load(
    trainingData,
    format="csv",
    header=True,
    inferSchema=True
)

# Define an assembler to create a column (features) of type Vector
# containing the double values associated with columns attr1, attr2, attr3
assembler = VectorAssembler(
    inputCols=["attr1", "attr2", "attr3"],
    outputCol="features"
)

# Create a LinearRegression object.
# LinearRegression is an Estimator that is used to
# create a regression model based on linear regression
lr = LinearRegression()

# Set the values of the parameters of the
# Linear Regression algorithm using the setter methods.
# There is one set method for each parameter
# For example, we are setting the number of maximum iterations to 10
# and the regularization parameter to 0.01
lr.setMaxIter(10)
lr.setRegParam(0.01)

# Define a pipeline that is used to create the linear regression
# model on the training data. The pipeline includes also
# the preprocessing step
pipeline = Pipeline().setStages([assembler, lr])

# Execute the pipeline on the training data to build the
# regression model
regressionModel = pipeline.fit(trainingData)
# Now, the regression model can be used to predict the target attribute value
```

1.2 LR with textual data

The linear regression algorithms can be used also when the input dataset is a collection of documents/texts. Also in this case the text must be mapped to a set of continuous attributes.

1.3 Parameter setting

The tuning approach that we used for the classification problem can also be used to optimize the regression problem, the only difference is given by the used evaluator: in this case the difference between the actual value and the predicted one must be computed.