Spark Machine Learning Library Tutorial

Spark MLlib | Basic Statistics | Linear Models

Binary Classification

Binary classification aims to divide items into two categories: positive and negative. MLlib supports two linear methods for binary classification: linear support vector machines (SVMs) and logistic regression. For both methods, MLlib supports L1 and L2 regularized variants. The training data set is represented by an RDD of LabeledPoint in MLlib. Note that, in the mathematical formulation in this guide, a training label y is denoted as either +1 (positive) or −1 (negative), which is convenient for the formulation. However, the negative label is represented by 0 in MLlib instead of −1, to be consistent with multi-class labeling.

The simplest method to solve optimization problems of the form $\min_{w \in \mathbb{R}} df(w)$ is gradient descent. This methodology is very well-suited for large-scale and distributed computation. Spark MLlib uses stochastic gradient descent (SGD) to solve these optimization problems, which are the core of supervised machine learning, for optimizations and much higher performance.

Evaluation Metrics

Precision is the fraction of retrieved documents that are relevant to the find:

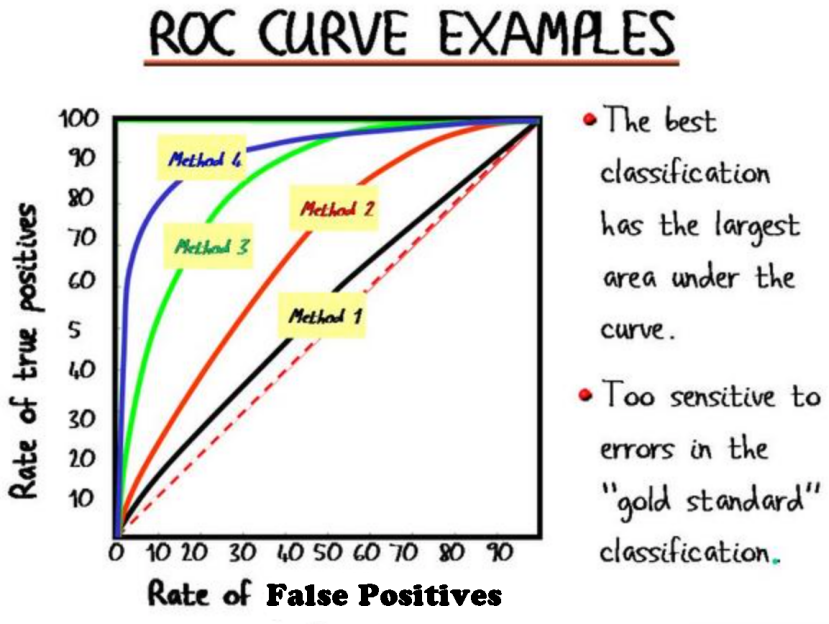
$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

Recall is the fraction of the documents that are relevant to the query that are successfully retrieved:

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

Receiver operating characteristic (ROC), or ROC curve, is a graphical plot that illustrates the performance of a binary classifier system and created by plotting the true positive rate against the false positive rate.



A classifier with the Red dashed line is guessing the label randomly. Closer the ROC curve gets to top-left part of the chart, better the classifier is. Area under the curves (AUC) is the area below these ROC curves. Therefore, in other words, AUC is a great indicator of how well a classifier functions.

AUC is commonly used to compare the performance of various models while precision/recall/F-measure can help determine the appropriate threshold to use for prediction purposes.

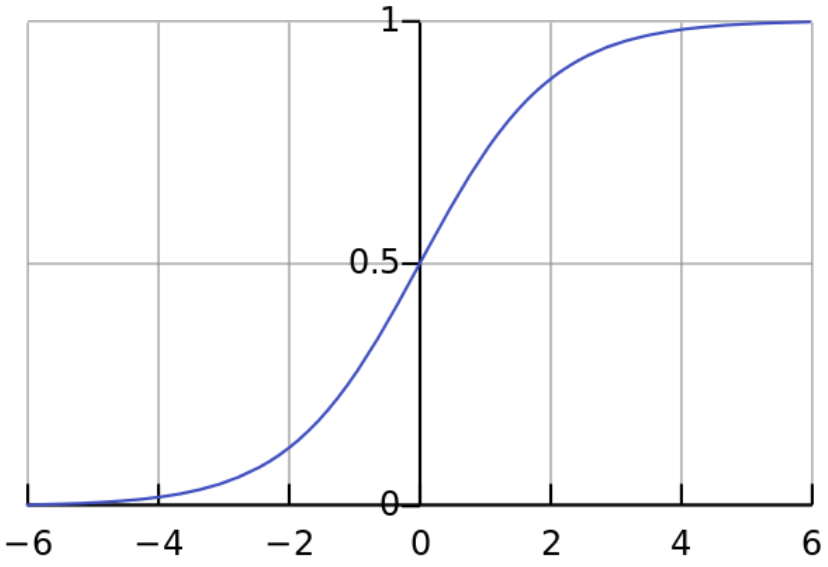
Logistic Regression

Logistic regression is a type of probabilistic statistical classification model. Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables, which are usually (but not necessarily) continuous, by using probability scores as the predicted values of the dependent variable.

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}},$$

$$t = \beta_0 + \beta_1 x$$

$$F(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$



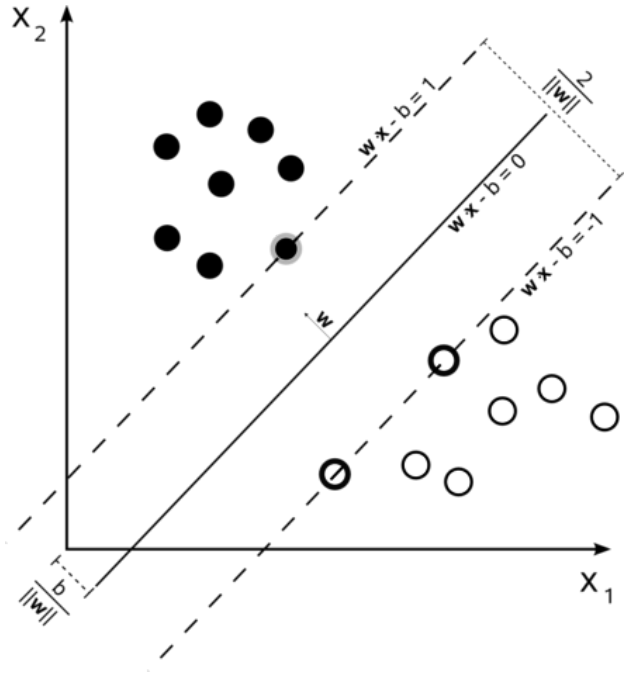

```
from pyspark import SparkContext
from pyspark.mllib.classification import LogisticRegressionWithSGD
from pyspark.mllib.regression import LabeledPoint
from numpy import array

# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split(' ')]
    return LabeledPoint(values[0], values[1:])

sc = SparkContext(appName="LogisticRegression")


data = sc.textFile("data/mllib/sample_svm_data.txt")
parsedData = data.map(parsePoint)

# Build the model
model = LogisticRegressionWithSGD.train(parsedData)

# Evaluating the model on training data
labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
print("Training Error = " + str(trainErr))

sc.stop()
```

---Downloads/spark-1.2.1-bin-hadoop2.4 » ./bin/pyspark examples/src/main/python/mllib/logistic_regression.py > logistic_regression.txt
---Downloads/spark-1.2.1-bin-hadoop2.4 » cat logistic_regression.txt

Linear Support Vector Machines (SVMs)

In machine learning, support vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis. Given a set of training examples, each belonging to one of two class labels, an SVM algorithm builds a model that assigns new examples into one label or another. Unlike Logistic Regression, SVM is a non-probabilistic binary linear classifier.

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.



More formally, an SVM constructs a hyperplane to separate data points belonging to two class labels in feature space. A good separation is achieved by the hyperplane that has the largest distance to the nearest training data point of any class. Maximizing the separability between the two classes reduces the generalization error of the classifier.

```
import org.apache.spark.SparkContext
import org.apache.spark.mllib.classification.SVMWithSGD
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.util.MLUtils

// Load training data in LIBSVM format.
val data = MLUtils.loadLibSVMFile(sc, "data/mllib/sample_libsvm_data.txt")

// Split data into training (60%) and test (40%).
val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)
val training = splits(0).cache()
val test = splits(1)

// Run training algorithm to build the model
val numIterations = 100
val model = SVMWithSGD.train(training, numIterations)

// Clear the default threshold.
model.clearThreshold()

// Compute raw scores on the test set.
val scoreAndLabels = test.map { point =>
  val score = model.predict(point.features)
    (score, point.label)
}

// Get evaluation metrics.
val metrics = new BinaryClassificationMetrics(scoreAndLabels)
val auROC = metrics.areaUnderROC()

println("Area under ROC = " + auROC)
```

---Downloads/spark-1.2.1-bin-hadoop2.4 » ./bin/spark-shell
---scala > :load svm.scala

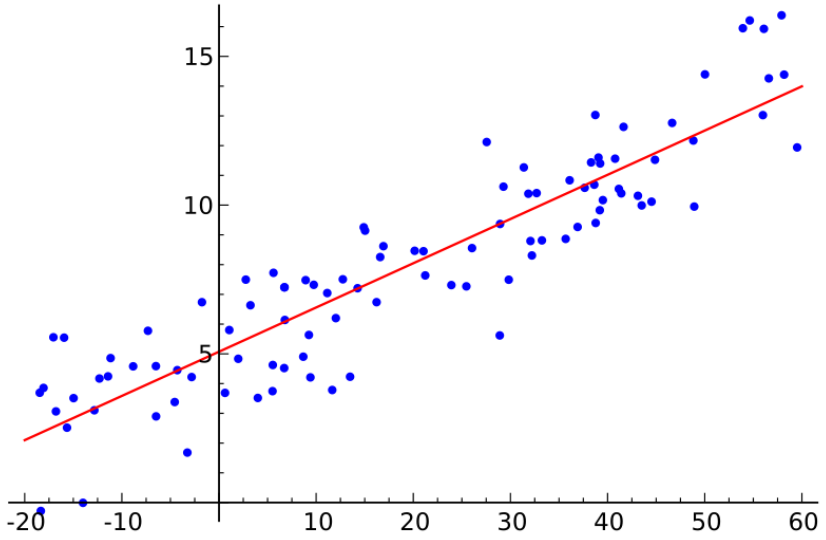
Linear Regression

Linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory (independent) variables. It is used for correlation analysis and tries to come up with the best model that fits the values of independent variables.

Linear regression is not a classification algorithm. Unlike SVM and logistic regression, it is used for predicting the value of dependent variable with as little error as possible rather than predicting the class label. In order to do this, it needs to learn the correlated features by calculating the linear coefficients of the independent variables according to the following formula:

$$y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathrm{T}} \boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n,$$

As seen from the regression formula, a linear regression model assumes that the relationship between the dependent variable yi and the vector of regressors xi is linear. Error variable εi is an unobserved random variable that adds noise to the linear relationship between the dependent variable and regressors.



After coefficients are calculated through learning and model is created that will produce the red line above, this model will be used to predict the value of the dependent variable based on the input values of the independent variables.

Evaluation Metrics for Model Correctness

Calculation of residuals is crucial for assessing the correctness of the regression model. Residuals are the differences between the observed and predicted values by the model. Using these residual values, evaluation metrics can be calculated:

The sum of squared residuals (SSE)

$$SSE = \Sigma(y_i - \hat{y}_i)^2$$

The sum of squared total residuals (SST)

$$SST = \Sigma(y_i - \bar{y}0)^2$$

$$R^2 = 1 - \frac{SSE}{SST}$$

$$R^2{}_a = \frac{(n-1)R^2 - k]}{[n-(k+1)]}$$

$$\hat{\sigma}^2 = \frac{SSE}{n-(k+1)} = MSE$$

$$f = \frac{R^2/k}{(1-R^2)/[n-(k+1)]}$$

$$f \geq F_{a,k,n-(k+1)}$$

```
import sys
from pyspark import SparkContext
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD
from numpy import array

# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.replace(',', ' ').split(' ')]
    return LabeledPoint(values[0], values[1:])

sc = SparkContext(appName="LinearRegressionPredict")
data = sc.textFile("data/mllib/ridge-data/lpsa.data")
parsedData = data.map(parsePoint)

# Build the model
model = LinearRegressionWithSGD.train(parsedData)

# Evaluate the model on training data
valuesAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
MSE = valuesAndPreds.map(lambda (v, p): (v - p)**2).reduce(lambda x, y: x + y) /
valuesAndPreds.count()
print("Mean Squared Error = " + str(MSE))
sc.stop()
```

```
---Downloads/spark-1.2.1-bin-hadoop2.4 » ./bin/pyspark examples/src/main/python/mllib/linear_regression.py > linear_regression.txt
---Downloads/spark-1.2.1-bin-hadoop2.4 » cat linear_regression.txt
```

```
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.StreamingLinearRegressionWithSGD

val trainingData =
ssc.textFileStream("/training/data/dir").map(LabeledPoint.parse).cache()
val testData = ssc.textFileStream("/testing/data/dir").map(LabeledPoint.parse)
val numFeatures = 3
val model = new
StreamingLinearRegressionWithSGD().setInitialWeights(Vectors.zeros(numFeatures))
model.trainOn(trainingData)
model.predictOnValues(testData.map(lp => (lp.label, lp.features))).print()
ssc.start()
ssc.awaitTermination()
```