Linear Methods – RDD-Based APIs

Kazi Aminul Islam
Department of Computer Science
Kennesaw State University

Mathematical Formulation

- Many standard machine learning methods can be formulated as a convex optimization problem, i.e., minimize f(w), where w is weights in d dimension.
- $f(w) = \lambda R(w) + \frac{1}{n}\sum_{i=1}^{n} L(w; x_i, y_i)$ where $x_i \in R^d$ are the training data samples and $y_i \in R$ are their corresponding labels.
- $R(w)$ is a regularizer that controls the complexity of the model.
- We call the method linear if $L(w; x_i, y_i)$ can be represented as a function of $w^T x$ and y.

Minimize Objective Function

- In general, we are minimizing the objective function (if convex) that has an error measurement and a penalty.

$$minimize\ f(w) = error(w) + penalty(w)$$

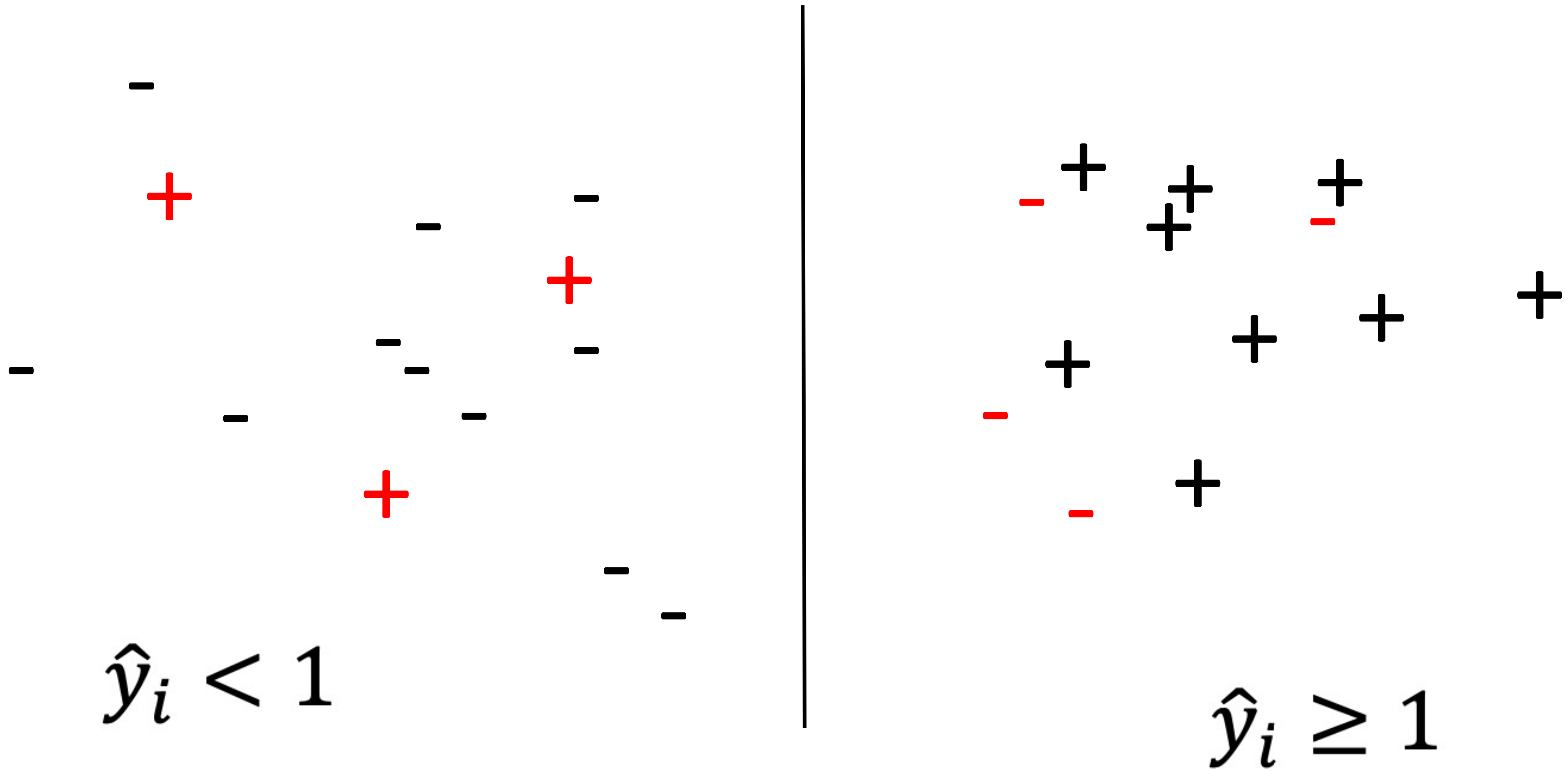
Hinge Loss

- Hinge loss is a loss function for training classifiers.
- In linear regression, we may computer a raw score by $Y = W^T X$ where W is the weigh vector and X is a feature vector.
- If we want to classify inputs into two categories, we can set a threshold, e.g., 1.0, that a raw score about 1.0 is one class and otherwise the other.
- We can let the label be +1 or -1. The output will be

$$\hat{t}_i = \begin{cases} +1 \; if \; \widehat{y_i} \geq 1 \\ -1 \; if \widehat{y_i} < 1 \end{cases}$$

- However, we may miss-classify some inputs.

Miss-Classification

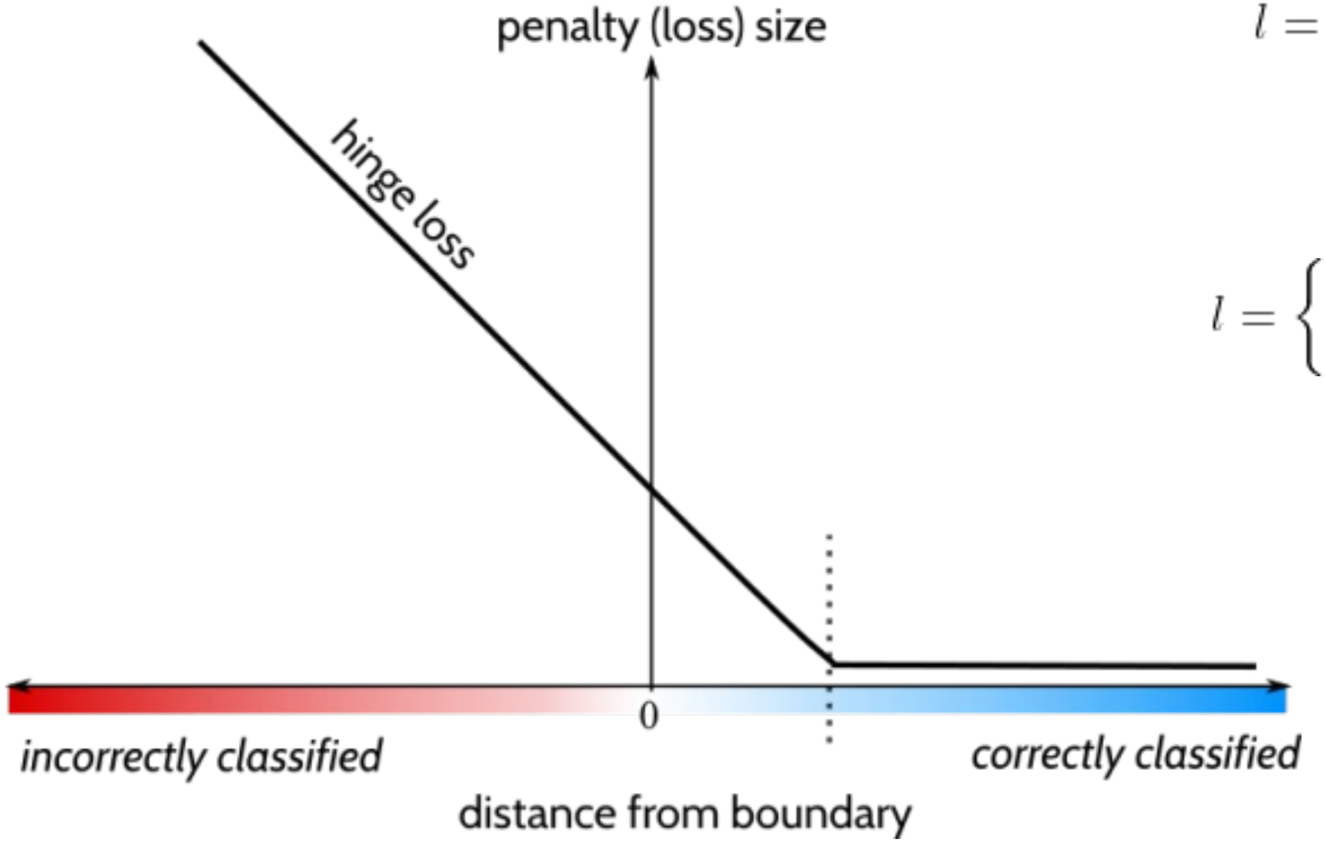


Penalize Miss-Classification Count

- Miss-count can be computed by $\sum y_i \neq \hat{t}_i$
- Miss-classification can be compute by $y_i\hat{t}_i < 0$, opposite signs.
- If $y_i\hat{t}_i > 0$, that would be great and we don't penalize it.
- Since $\hat{t}_i$ is computed by $\hat{y}_i$, the penalty condition becomes $y_i\hat{y}_i < 1$

The less, the more penalty!

Hinge Loss Function



$$l = max(0, 1 - y^i(x^i - b))$$

$$l = \begin{cases} 0 \quad if\ y \cdot (w \cdot x) \ \geq 1 \\ 1 - y \cdot (w \cdot x)\ otherwise \end{cases}$$

Logistic Loss

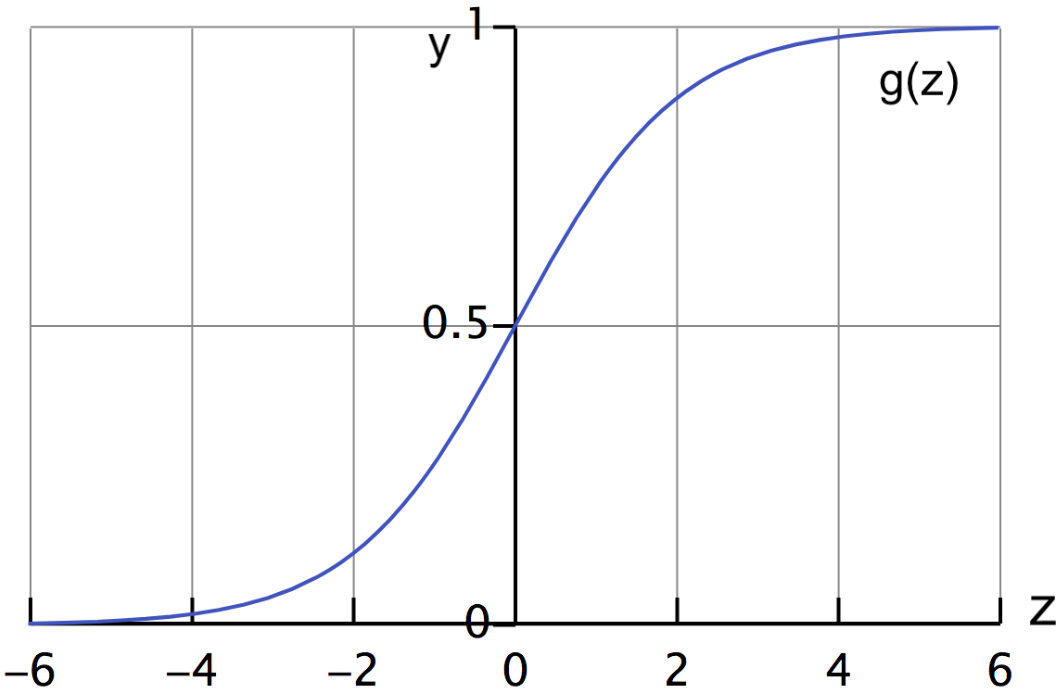
We can compute yW^Tx and take exponential like e^{-yW^Tx}, where $y \in \{+1, -1\}$.

If $W^Tx \gg 0$ and $\mathrm{y} = +1$, $\mathrm{e}^{-yW^Tx} \to 0$. $W^Tx \ll 0$ and $\mathrm{y} = -1$, $\mathrm{e}^{-yW^Tx} \to 0$.

If the sign of W^Tx and y are different, $W^Tx \ll 0$ or $W^Tx \gg 0$ implies $\mathrm{e}^{-yW^Tx} \to \infty$.

Logistic Loss (cont.)

- Logistic loss: $\log(1+e^{-yW^T x})$
- Subgrident: $\frac{\partial \log(1+e^{-yW^T x})}{\partial w} = \frac{1}{1+e^{-yW^T x}} e^{-yW^T x}(-y^T x)$
- $\Rightarrow \frac{1+e^{-yW^T x}-1}{1+e^{-yW^T x}}(-y^T x)$
- $\Rightarrow (1-\frac{1}{1+e^{-yW^T x}})(-y^T x)$



Loss Functions Supported by spark.mllib

	loss function $L(\mathbf{w}; \mathbf{x}, y)$	gradient or sub-gradient
hinge loss	$\max\{0, 1 - y\mathbf{w}^T\mathbf{x}\}, \quad y \in \{-1, +1\}$	$\begin{cases} -y \cdot \mathbf{x} & \text{if } y\mathbf{w}^T\mathbf{x} < 1, \\ 0 & \text{otherwise.} \end{cases}$
logistic loss	$\log(1 + \exp(-y\mathbf{w}^T\mathbf{x})), \quad y \in \{-1, +1\}$	$-y\left(1 - \frac{1}{1+\exp(-y\mathbf{w}^T\mathbf{x})}\right) \cdot \mathbf{x}$
squared loss	$\frac{1}{2}(\mathbf{w}^T\mathbf{x} - y)^2, \quad y \in \mathbb{R}$	$(\mathbf{w}^T\mathbf{x} - y) \cdot \mathbf{x}$

Note: spark.mllib uses 0 instead of -1 to be consistent with multiclass labeling.

Regularizers

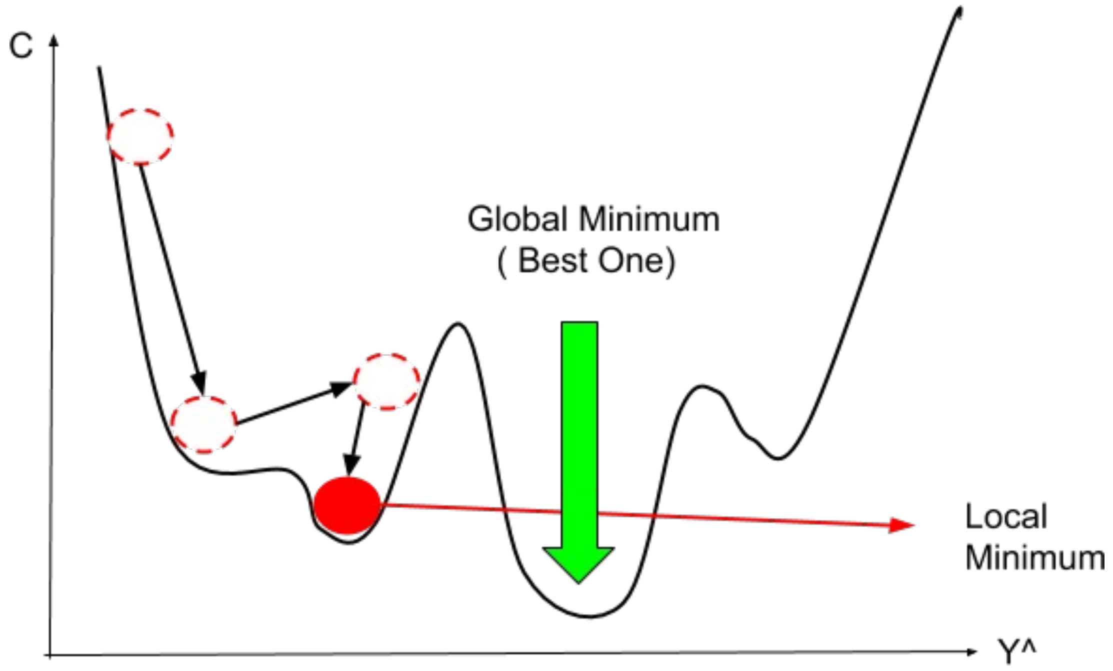
- The purpose of the regularizer is to encourage simple models and avoid overfitting.
- L2-regularized problems are generally easier to solve than L1-regularized due to smoothness.
- However, L1 regularization can help promote sparsity in weights leading to smaller and more interpretable models, the latter of which can be useful for feature selection.
- Elastic net is a combination of L1 and L2 regularization. It is not recommended to train models without any regularization, especially when the number of training examples is small.

Spark.mllib Supported Regularizers

	regularizer $R(\mathbf{w})$	gradient or sub-gradient
zero (unregularized)	0	$\mathbf{0}$
L2	$\frac{1}{2}\|\mathbf{w}\|_2^2$	$\mathbf{w}$
L1	$\|\mathbf{w}\|_1$	$\mathrm{sign}(\mathbf{w})$
elastic net	$\alpha\|\mathbf{w}\|_1 + (1-\alpha)\frac{1}{2}\|\mathbf{w}\|_2^2$	$\alpha \mathrm{sign}(\mathbf{w}) + (1-\alpha)\mathbf{w}$

Here $\mathrm{sign}(\mathbf{w})$ is the vector consisting of the signs (± 1) of all the entries of $\mathbf{w}$.

Gradient Descent



Optimization

- Stochastic Gradient Descent (SGD)
  - A stochastic subgradient is a randomized choice of a vector, such that in expectation, we obtain a true subgradient of the original objective function.
- Limited-Memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS)
  - L-BFGS is an optimization algorithm in the family of quasi-Newton methods to solve the optimization problems. The L-BFGS method approximates the objective function locally as a quadratic without evaluating the second partial derivatives of the objective function.

Linear Regression Model

- The following example demonstrates linear regression model and extract model summary statistics.

Import LinearRegression Library

- Import Spark LinearRegression library

```
import org.apache.spark.ml.regression.LinearRegression
```

```
scala> import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.regression.LinearRegression
```

Load Training Data

- The data file sample_linear_regression_data.txt is stored under c:/spark/data/mslib.

```scala
// Load training data
val training = spark.read.format("libsvm")
  .load("data/mllib/sample_linear_regression_data.txt")
```

```
scala> val training = spark.read.format("libsvm").load("data/mllib/sample_linear_regression_d
ata.txt")
20/09/21 10:55:34 WARN LibSVMFileFormat: 'numFeatures' option not specified, determining the
number of features by going though the input. If you know the number in advance, please speci
fy it via 'numFeatures' option to avoid the extra scan.
training: org.apache.spark.sql.DataFrame = [label: double, features: vector]

scala> training.first
res6: org.apache.spark.sql.Row = [-9.490009878824548,(10,[0,1,2,3,4,5,6,7,8,9],[0.45512736006
57362,0.36644694351969087,-0.38256108933468047,-0.4458430198517267,0.33109790358914726,0.8067
445293443565,-0.2624341731773887,-0.44850386111659524,-0.07269284838169332,0.5658035575800715
]))]
```

Create a Linear Model

- Create a LinearRegression model and set parameters
- Set regulator parameter (λ) for L1 or L2 regulation.
- Set Elastic Net Parameter to 1 for L1 and 0 for L2 regulation.
- Any number between 0 and 1 for Elastic Net Regulation.

```
val lr = new LinearRegression()
  .setMaxIter(10)
  .setRegParam(0.3)
  .setElasticNetParam(0.8)
```

Create a Linear Model (cont.)

- Create a linear regression model
- Set L2 parameter (λ) to 0.3
- Choose L2 regulation by setting Elastic Net parameter to 0

```
scala> val lr = new LinearRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0)
lr: org.apache.spark.ml.regression.LinearRegression = linReg_c2433a3436c1
```

Fit the Model

- Fit the model

```scala
// Fit the model
val lrModel = lr.fit(training)
```

```
scala> val lrModel = lr.fit(training)
20/09/21 11:42:36 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.Nat
iveSystemBLAS
20/09/21 11:42:36 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.Nat
iveRefBLAS
20/09/21 11:42:36 WARN LAPACK: Failed to load implementation from: com.github.fommil.netlib.N
ativeSystemLAPACK
20/09/21 11:42:36 WARN LAPACK: Failed to load implementation from: com.github.fommil.netlib.N
ativeRefLAPACK
lrModel: org.apache.spark.ml.regression.LinearRegressionModel = LinearRegressionModel: uid=li
nReg_c2433a3436c1, numFeatures=10
```

Display β

- Print coefficients and intercept

```scala
// Print the coefficients and intercept for linear regression
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")
```

```
scala> println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")
Coefficients: [0.010541828081257216,0.8003253100560949,-0.7845165541420371,2.3679887171421914
,0.5010002089857577,1.1222351159753026,-0.2926824398623296,-0.49837174323213035,-0.6035797180
675657,0.6725550067187461] Intercept: 0.14592176145232041
```

Summarize the Model and Some Metrics

- Show performance metrics over the training data such as total # of iterations, residuals, RMSE, R2, etc.

```
// Summarize the model over the training set and print out some metrics
val trainingSummary = lrModel.summary
println(s"numIterations: ${trainingSummary.totalIterations}")
println(s"objectiveHistory: [${trainingSummary.objectiveHistory.mkString(",")}]")
trainingSummary.residuals.show()
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
println(s"r2: ${trainingSummary.r2}")
```

Performance Metrics

- Use lrModel.summary

```
scala> val trainingSummary = lrModel.summary
trainingSummary: org.apache.spark.ml.regression.LinearRegressionTrainingSummary = org.apache.
spark.ml.regression.LinearRegressionTrainingSummary@6c1ab83e

scala> println(s"numIterations: ${trainingSummary.totalIterations}")
numIterations: 1

scala> println(s"objectiveHistory: [${trainingSummary.objectiveHistory.mkString(",")}]")
objectiveHistory: [0.0]
```

Residuals

- Residuals $(y_i - \hat{y}_i)$

```
scala> trainingSummary.residuals.show()
+--------------------+
|           residuals|
+--------------------+
|-10.974359174246889|
| 0.8872320138420559|
| -4.596541837478908|
|-20.411667435019638|
|-10.270419345342642|
|-6.015605895679905|
|-10.663939415849267|
| 2.1153960525024713|
| 3.9807132379137675|
|-17.225218272069533|
| -4.611647633532147|
| 6.4176669407698546|
| 11.407137945300537|
| -20.70176540467664|
| -2.683748540510967|
|-16.755494794232536|
|  8.154668342638725|
|-1.4355057987358848|
```

RMSE and R2

- Print RMSE and R2

```
scala> println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
RMSE: 10.163223095528005

scala> println(s"r2: ${trainingSummary.r2}")
r2: 0.027814017194997764
```

Using Loop to Tune Parameters

- We can iterate over different regulation parameter and find out the best result in terms of rootMeanSquaredError.

```
scala> for (i <- 0 to 20) {
     | lr.setRegParam(i)
     | println(i, lr.fit(training).summary.rootMeanSquaredError)
     | }
20/09/21 12:15:14 WARN Instrumentation: [48e60d57] regParam is zero, which might cause numeri
cal instability and overfitting.
(0,10.16309157133015)
(1,10.164365290987059)
(2,10.167342927390015)
(3,10.171196183679307)
(4,10.175457155913001)
(5,10.179856774153544)
(6,10.184239543150142)
(7,10.18851658294818)
(8,10.192638847311805)
(9,10.196581408275962)
(10,10.200334018050128)
(11,10.203895334967278)
(12,10.207269340563075)
```

Scala Code

```scala
import org.apache.spark.ml.regression.LinearRegression

// Load training data
val training = spark.read.format("libsvm")
  .load("data/mllib/sample_linear_regression_data.txt")

val lr = new LinearRegression()
  .setMaxIter(10)
  .setRegParam(0.3)
  .setElasticNetParam(0.8)

// Fit the model
val lrModel = lr.fit(training)

// Print the coefficients and intercept for linear regression
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

// Summarize the model over the training set and print out some metrics
val trainingSummary = lrModel.summary
println(s"numIterations: ${trainingSummary.totalIterations}")
println(s"objectiveHistory: [${trainingSummary.objectiveHistory.mkString(",")}]")
trainingSummary.residuals.show()
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
println(s"r2: ${trainingSummary.r2}")
```

https://spark.apache.org/docs/latest/ml-classification-regression.html#linear-regression

References

- Spark 3.0.1 https://spark.apache.org/docs/latest/ml-classification-regression.html#linear-regression
- Spark 3.0.1 ScalaDoc https://spark.apache.org/docs/latest/api/scala/org/apache/spark/ml/regression/LinearRegression.html