Generalized Linear Models in Spark MLlib and SparkR

Xiangrui Meng
joint with Joseph Bradley, Eric Liang, Yanbo Liang (MiningLamp), DB Tsai (Netflix), et al.

2016/02/17 - Spark Summit East

databricks™

About me

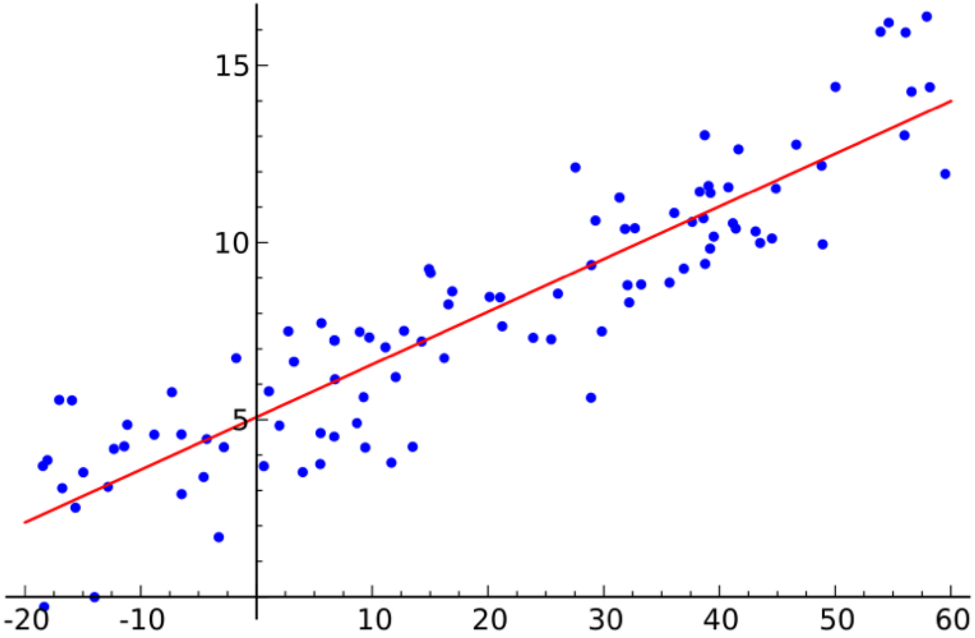
- Software Engineer at Databricks
- Spark PMC member and MLlib/PySpark maintainer
- Ph.D. from Stanford on randomized algorithms for large-scale linear regression problems

Outline

- Generalized linear models (GLMs)
  - linear regression / logistic regression / general form
  - accelerated failure time (AFT) model for survival analysis
  - intercept / regularization / weights
- GLMs in MLlib and SparkR
  - demo: R formula in Spark
- Implementing GLMs
  - gradient descent / L-BFGS / OWL-QN
  - weighted least squares / iteratively re-weighted least squares (IRLS)
  - performance tips

Generalized linear models

Linear regression



inference / prediction

https://en.wikipedia.org/wiki/Linear_regression

Linear least squares

- m observations: $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$
- x: explanatory variables, y: dependent variable
- assumes linear relationship between x and y

$$y = x^T \beta + \varepsilon$$

- minimizes the sum of the squares of the errors

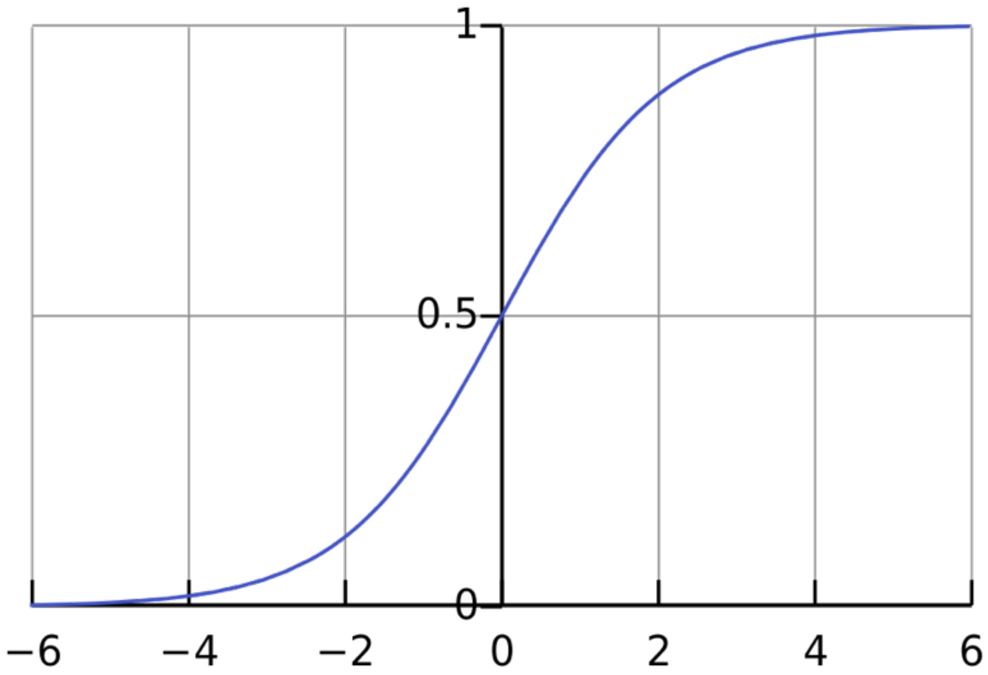
$$\text{minimize}_{\beta \in \mathbb{R}^n} \quad \frac{1}{2} \sum_{i=1}^{m} \|y_i - x_i^T \beta\|_2^2$$

Linear least squares

- the oldest linear model, trace back to Gauss
- the simplest and the most studied linear model
- has analytic solutions
- easy to solve
- easy to inspect

- sensitive to outliers

Logistic regression



https://en.wikipedia.org/wiki/Logistic_regression

Logistic regression

- classification with binary response: $y \in \{1, -1\}$
  - true/false, clicked/not clicked, liked/disliked
- uses logistic function to indicate the likelihood

$$P(y = 1) = \frac{1}{1 + e^{-x^T \beta}}$$

- maximizes the sum of the log-likelihoods, i.e.,

$$\text{minimize}_\beta \quad \sum_{i=1}^{m} \log(1 + e^{-y_i \cdot x_i^T \beta})$$

Logistic regression

- one of the simplest binary classification models
- widely used in industry
- relatively easy to solve
- easy to interpret

Multinomial logistic regression

- classification with multiclass response: $y \in \{1, 2, \ldots, K\}$
- uses softmax function to indicate likelihood

$$P(y = k) = e^{x^T \beta_k} / Z, \text{ where } Z = \sum_{l=1}^{K} e^{x^T \beta_l}$$

- maximizes the sum of log-likelihoods

$$\text{maximize}_{\beta_1, \ldots, \beta_K} \quad \sum_{i=1}^{m} \sum_{l=1}^{K} I_{y_i = l} \log \left(e^{x_i^T \beta_l} / Z_i \right)$$

Generalized linear models (GLMs)

- Both linear least squares and logistic regression are special cases of generalized linear models.
- A GLM is specified by the following:
  - a distribution of the response (from the exponential family),
  - a link function g such that $\mathbf{E}(y) = g^{-1}(x^T\beta)$
- maximizes the sum of log-likelihoods

$$\text{maximize}_\beta \quad \sum_{i=1}^{m} \log p(y_i | x_i; \beta)$$

Distributions and link functions

Model	Distribution	Link
linear least squares	normal	identity
logistic regression	binomial	logit
multinomial logic regression	multinomial	generalized logit
Poisson regression	Poisson	log
gamma regression	gamma	inverse

Accelerated failure time (AFT) model

- m observations: $(x_1, y_1, c_1), \ldots, (x_m, y_m, c_m)$
- y: survival time, c: censor variable (alive or dead)
- assumes the effect of an explanatory variable is to accelerate or decelerate the life time by some constant
- uses maximum likelihood estimation while treating censored and uncensored observations differently

AFT model for survival analysis

- one popular parametric model for survival analysis
- widely used for lifetime estimation and churn analysis
- could be solved under the same framework as GLMs

Intercept, regularization, and weights

In practice, a linear model is often more complex

$$\text{maximize}_{\beta} \quad \sum_{i=1}^{m} w_i \cdot \log p(y_i | x_i^T \beta + \beta_0) + \lambda \cdot \sigma(\beta)$$

where w describes instance weights, beta_0 is the intercept term to adjust bias, and sigma regularized beta with a constant lambda > 0 to avoid overfitting.

databricks

Types of regularization

- Ridge (L2): $\frac{1}{2}\|\beta\|_2^2$
  - easy to solve (strongly convex)
- Lasso (L1): $\|\beta\|_1$
  - enhance model sparsity
  - harder to solver (though still convex)
- Elastic-Net: $\alpha\|\beta\|_1 + \frac{1-\alpha}{2}\|\beta\|_2^2, \quad \alpha \in [0,1]$
- Others: group lasso, nonconvex, etc

GLMs in MLlib and SparkR

GLMs in Spark MLlib

Linear models in MLlib are implemented as ML pipeline estimators. They accept the following params:

- **featuresCol**: a vector column containing features (x)
- **labelCol**: a double column containing responses (y)
- **weightCol**: a double column containing weights (w)
- **regType**: regularization type, “none”, “l1”, “l2”, “elastic-net”
- **regParam**: regularization constant
- **fitIntercept**: whether to fit an intercept term
- …

Fit a linear model in MLlib

```
from pyspark.ml.classification import LogisticRegression

# Load training data
training = sqlContext.read.parquet(“path/to/training“)

lr = LogisticRegression(
    weightCol=“weight”, fitIntercept=False, maxIter=10,
    regParam=0.3, elasticNetParam=0.8)

# Fit the model
model = lr.fit(training)
```

Make predictions and evaluate models

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

test = sqlContext.read.parquet(“path/to/test“)

# make predictions by calling transform
predictions = model.tranform(test)

# create a binary classification evaluator
evaluator = BinaryClassificationEvaluator(
    metricName=“areaUnderROC”)
evaulator.evaluate(predictions)
```

databricks

GLMs in SparkR

In Python/Scala/Java, we keep the APIs about the same for consistency. But in SparkR, we make the APIs similar to existing ones in R (or R packages).

```
# Create the DataFrame
df <- read.df(sqlContext, “path/to/training”)

# Fit a Gaussian GLM model
model <- glm(y ~ x1 + x2, data = df, family = "gaussian")
```

R formula in SparkR

- R provides model formula to express linear models.
- We support the following R formula operators in SparkR:
  - `~` separate target and terms
  - `+` concat terms, "+ 0" means removing intercept
  - `-` remove a term, "- 1" means removing intercept
  - `:` interaction (multiplication for numeric values, or binarized categorical values)
  - `.` all columns except target
- For example, "y ~ x + z + x:z -1" means using x, z, and their interaction (x:z) to predict y without intercept (-1).

Demo: GLMs in Spark

… using Databricks Community Edition!

Implementing GLMs

Row-based distributed storage

w	x	y

partition 1

w	x	y

partition 2

Gradient descent methods

- Stochastic gradient descent (SGD): $\beta := \beta - \mu \cdot g(\beta; x_i, y_i)$
  - trade-offs on the merge scheme and convergence
- Mini-batch SGD: $\beta := \beta - \mu \cdot \sum_{i \in \mathcal{B}_j} g(\beta; x_i, y_i)$
  - hard to sample mini-batches efficiently
  - communication overhead on merging gradients
- Batch gradient descent: $\beta := \beta - \mu \cdot \sum_{i=1}^{m} g(\beta; x_i, y_i)$
  - slow convergence

Quasi-Newton methods

- Newton's method converges much than GD, but it requires second-order information: $\beta := \beta - H^{-1}g$
- L-BFGS works for smooth objectives. It approximates the inverse Hessian using only first-order information.
- OWL-QN works for objectives with L1 regularization.
- MLlib calls L-BFGS/OWL-QN implemented in breeze.

Direct methods for linear least squares

- Linear least squares has an analytic solution:

$$\beta = (X^T X)^{-1} X^T y$$

- The solution could be computed directly or through QR factorization, both of which are implemented in Spark.
- requires only a single pass
- efficient when the number of features is small (<4000)
- provides R-like model summary statistics

Iteratively re-weighted least squares (IRLS)

- Generalized linear models with exponential family can be solved via iteratively re-weighted least squares (IRLS).
  - linearizes the objective at the current solution
  - solves the weighted linear least squares problem
  - repeat above steps until convergence
- efficient when the number of features is small (<4000)
- provides R-like model summary statistics
- This is the implementation in R.

Verification using R

Besides normal tests, we also verify our implementation using R.

```
/*
   df <- as.data.frame(cbind(A, b))
   for (formula in c(b ~ . -1, b ~ .)) {
     model <- lm(formula, data=df, weights=w)
     print(as.vector(coef(model)))
   }

   [1] -3.727121  3.009983
   [1] 18.08  6.08 -0.60
 */
val expected = Seq(Vectors.dense(0.0, -3.727121, 3.009983),
                   Vectors.dense(18.08, 6.08, -0.60))
```

Standardization

To match the result in both R and glmnet, the most popular R package for GLMs, we provide options to standardize features and labels before training:

$$\sigma(\beta) = \frac{1}{2\delta} \sum_{j=1}^{n} (\sigma_j \beta_j)^2$$

where delta is the stddev of labels, and sigma_j is the stddev of the j-th feature column.

Performance tips

- Utilize sparsity.
- Use tree aggregation and torrent broadcast.
- Watch numerical issues, e.g., log(1+exp(x)).
- Do not change input data. Scaling could be applied after each iteration and intercept could be derived later.

Future directions

- easy handling of categorical features and labels
- better R formula support
- more model summary statistics
- feature parity in different languages
- model parallelism
  - vector-free L-BFGS with 2D partitioning (WIP)
- using matrix kernels

Other GLM implementations on Spark

- CoCoA+: communication-efficient optimization
- LIBLINEAR for Spark: a Spark port of LIBLINEAR
- sparkGLM: an R-like GLM package for Spark
- TFOCS for Spark: first-order conic solvers for Spark
- General-purpose packages that implement GLMs
  - aerosolve, DistML, sparkling-water, thunder, zen, etc
- … and more on Spark Packages

Thank you.

- MLlib user guide and roadmap for Spark 2.0
- GLMs on Wikipedia
- Databricks Community Edition, blog posts, and careers

databricks™