HPDGLM – Distributed version of glm

User Guide

Contents

HPDGLM – Distributed version of glm

User Guide

1. Introduction

The HPDGLM package provides a distributed Generalized Linear Model. HPDGLM is written based on the distributed R infrastructure developed by HP-Labs. The main functions of the HPDGLM package are:

- hpdglm: A distributed alternative for the R glm function.
- v.hpdglm: Evaluates a model built by hpdglm using the Split-Sample-Validation method.
- cv.hpdglm: Evaluates a model built by hpdglm using the Cross-Validation method.

2. hpdglm function

The hpdglm function is a distributed alternative for the R glm function. The R glm function is available in stats package and is available in the standard R distribution. glm is a popular function for performing many forms of regression.

More detail about the interface of the function and its input arguments can be found in the manual of the package.

3. Peripheral functions

There are a few peripheral functions which can be used in relevance to the output model learned by hpdglm.

3.1. summary

This function prints a summary of the learned model.

3.2. coefficients

This function prints coefficients of the learned model. The abbreviated function is *coef*.

3.3. residuals

This function extracts model residuals in a darray. The abbreviated function is *resid*.

3.4. predict

This function produces predicted values, obtained by evaluating the regression function on provided new data. New data can be either a darray or a normal matrix. The signature of the function can be found in the manual.

4. v.hpdglm function

The v,hpdglm function evaluates a model built by hpdglm using the Split-Sample-Validation method. A percent of the data is randomly selected as test data, and a new model is built using the remaining data. Finally, the prediction cost of the new model on the test data is measured.

5. cv.hpdglm function

This function is implemented for evaluating a model built by hpdglm using the Cross-Validation method. The data is randomly divided into K folds, and the evaluation is repeated K times. Every iteration, one fold is reserved as the test data and the rest is used for training. Finally, the prediction costs of the new models on all folds are aggregated.

6. Examples

In this section, different modes of hpdglm function are displayed through some examples.

6.1. Linear Regression

This linear regression example uses a small data-frame called 'faithful', which is available in the standard distribution of R (package datasets). This example details how to build a model to predict value of `faithful$eruptions`, given the value of `faithful$waiting`, using four partitions in each darray.

1. Create a darray for response and predictors

```
Y <- as.darray(as.matrix(faithful$eruptions),c(ceiling(length(faithful$eruption)/4),1))
X <- as.darray(as.matrix(faithful$waiting),c(ceiling(length(faithful$waiting)/4),1))
```

2. Build the model

```
myModel <- hpdglm(responses=Y, predictors=X, family= gaussian(link=identity))
```

You can also use the shorthand version:

```
myModel <- hpdglm(Y, X)
```

3. Print the summary

```
> summary(myModel)
progress: 100%
progress: 100%
Calculating the covariant matrix
progress: 100%
Building residuals
progress: 100%

Call:
hpdglm(responses = Y, predictors = X)

progress: 100%
progress: 100%
Deviance Residuals:
   Min     Max
-1.299   1.193

Coefficients:
      Estimate Std. Error t value Pr(>|t|)
[1,] -1.874016   0.160143  -11.70   <2e-16 ***
[2,]  0.075628   0.002219   34.09   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.2465251)

    Null deviance: 353.039  on 271  degrees of freedom
Residual deviance:  66.562  on 270  degrees of freedom
AIC: 395.02

Number of Fisher Scoring iterations: 2
```

4. Predict the response for a few new samples:

```
> newSamples <- matrix(c(1:3),,1)
> predict(myModel, newSamples, "link")
          [,1]
[1,] -1.798388
[2,] -1.722760
[3,] -1.647132
```

5. Run Split-Sample-Validation:

```
> testV <- v.hpdglm(Y, X, myModel)
> testV$delta
[1] 0.2281465 0.2274641
```

6. Run Cross-Validation:

```
> testCV <- cv.hpdglm(Y, X, myModel)
> testCV$delta
[1] 0.2445006 0.2442940
```

6.2. Logistic Regression

This logistic regression example uses a small data-frame called 'mtcars', which is available in the standard distribution of R (package datasets). This example details how to build a model to predict the value of `mtcars$am`, given the value of `mtcars$wt` and `mtcars$hp` using four partitions in each darray.

1. Create a darray for response and predictors

```
Y <- as.darray(as.matrix(mtcars$am),c(ceiling(length(mtcars$am)/4),1))
X <- as.darray(as.matrix(cbind(mtcars$wt,mtcars$hp)),c(ceiling(length(mtcars$hp)/4),2))
```

2. Build the model

```
myModel <- hpdglm(responses=Y, predictors=X, family= binomial(logit))
```

You can also use the shorthand version:

```
myModel <- hpdglm(Y, X, binomial)
```

3. Print the summary

```
> summary(myModel)
Calculating the covariant matrix
progress: 100%
Building residuals
progress: 100%

Call:
hpdglm(responses = Y, predictors = X, family = binomial)

progress: 100%
progress: 100%
Deviance Residuals:
   Min     Max
-2.254   1.345

Coefficients:
     Estimate Std. Error z value Pr(>|z|)
[1,] 18.86630    7.44356   2.535  0.01126 *
[2,] -8.08348    3.06868  -2.634  0.00843 **
[3,]  0.03626    0.01773   2.044  0.04091 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 43.230  on 31  degrees of freedom
Residual deviance: 10.059  on 29  degrees of freedom
AIC: 16.059

Number of Fisher Scoring iterations: 8
```

4. Predict the response for a few new samples

```
> newSamples <- matrix(c(1:3),3,2)
> predict(myModel, newSamples, "response")
```

```
            [,1]
[1,] 0.999979986
[2,] 0.941136088
[3,] 0.005090073
```

5. Run Split-Sample-Validation

```
> testV <- v.hpdglm(Y, X, myModel)
> testV$delta
[1] 0.05269544 0.04091332
```

6. Run Cross-Validation

```
> testCV <- cv.hpdglm(Y, X, myModel)
> testCV$delta
[1] 0.04473004 0.04462759
```

6.3. Poisson Regression

This example for poisson regression uses the same dataset used for logistic regression example.

1. Create a darray for response and predictors

```
Y <- as.darray(as.matrix(mtcars$am),c(ceiling(length(mtcars$am)/4),1))
X <- as.darray(as.matrix(cbind(mtcars$wt,mtcars$hp)),c(ceiling(length(mtcars$hp)/4),2))
```

2. Build the model

```
myModel <- hpdglm(Y, X, poisson)
```

3. Print the summary

```
> summary(myModel)
Calculating the covariant matrix
progress: 100%
Building residuals
progress: 100%

Call:
hpdglm(responses = Y, predictors = X, family = poisson)

progress: 100%
progress: 100%
Deviance Residuals:
    Min      Max
-1.049     1.060

Coefficients:
      Estimate  Std. Error z value Pr(>|z|)
[1,]  2.287139   0.907534   2.520   0.01173 *
[2,] -1.580433   0.494010  -3.199   0.00138 **
[3,]  0.008831   0.005259   1.679   0.09309 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 23.420  on 31  degrees of freedom
Residual deviance: 10.481  on 29  degrees of freedom
```

```
AIC: 42.481

Number of Fisher Scoring iterations: 6
```

4. Predict the response for a few new samples

```
> newSamples <- matrix(c(1:3),3,2)
> predict(myModel, newSamples, "response")
          [,1]
[1,] 2.04528411
[2,] 0.42483043
[3,] 0.08824246
```

5. Run Split-Sample-Validation

```
> testV <- v.hpdglm(Y, X, myModel)
> testV$delta
[1] 0.1481145 0.1661539
```

6. RunCross-Validation

```
> testCV <- cv.hpdglm(Y, X, myModel)
> testCV$delta
[1] 0.14383761 0.09261486
```

6.4. Prediction by Vertica

You can use HPDGLM to connect to HP Vertica to both read data and store results. Assume that you have a table in Vertica database called mortgage. The table has three columns "*def*", "*mltvspline1*", and "*mltvspline2*". You want to find the logistic regression where def is response, and "*mltvspline1*" and "*mltvspline2*" are predictors. First, you make the model in the R environment using the HPDGLM package. Then, you can use the learned coefficients of the model in *vsql* to predict some new samples that are then stored in another table called newSamples. Eventually, you can store the results (the scores of the samples) in a table called prediction.

*** Inside R ***

```
library(distributedR)
library(HPDGLM)
distributedR_start()
loadedData <- db2darrays("mortgage", list("def"), list("mltvspline1", "mltvspline2"))
myModel <- hpdglm(loadedData$Y, loadedData$X, family=binomial(link=logit),
control=list(trace=FALSE))
coef(myModel)
             [,1]
(Intercept) -2.260290
mltvspline1  3.332633
mltvspline2 -1.537822
```

*** Inside vsql ***

```
CREATE TABLE prediction(p real);
```

```
  \set b0  -2.260290
  \set b1   3.332633
  \set b2  -1.537822
INSERT INTO prediction SELECT 1/(1+exp(-(:b0 + mltvspline1 * :b1 + mltvspline2 * :b2))) FROM
newSamples;
```

7. Experiments

The following experiments detail hardware requirements and the scalability of hpdglm function for different numbers of nodes.

Specification of each node for experiments presented in Table 1 and Figure 1

- HP-DL380
- 2 CPU per Node
- CPU model: Intel® Xeon® E5-2670 (20MB Cache, 2.60GHz)
- 120GB RAM
- 10Gb/s Ethernet Network
- CentOS release 6.4 (Final)
- R version 3.0.1

Specification of each node for experiments in Table 2, Figure 2, Figure 3, and Figure 4

- 2 CPU per Node
- CPU model: Intel® Xeon® X5650 (12MB Cache, 2.67GHz)
- 96GB RAM
- 1Gb/s Ethernet Network
- CentOS release 6.4 (Final)
- R version 3.0.1

7.1. Memory requirement

The following table shows the maximum data tested that hpdlgm can handle on a 1 to 8-node cluster. Columns “# node” and “# executors” display the number of nodes and total number of R-executors used

in the experiment. Column “Samples” indicates number of samples × number of features of a sample. Column “Input data size” indicated the size of samples in bytes.

Table 1. Experiments to measure memory requirement of logistic regression

# node	# executors	Samples	Input data size	Running time (sec)
1	31	300M×7	16.8GB	145.02
2	62	700M×7	39.2GB	172.375
3	93	1.0B×7	56.0GB	165.996
4	124	1.4B×7	78.4GB	179.925
5	155	1.7B×7	95.2GB	173.465
6	186	2.1B×7	117.6GB	180.064
7	217	2.5B×7	140GB	190.26
8	248	2.9B×7	162.4GB	270.906

7.2. Performance evaluation

While the standard glm function is limited to the resources of a single node, hpdglm utilizes the resources of many nodes. The main benefit of the hpdglm function is achieved when it is employed to process a massive number of samples, because the ratio of communication time to computation time is decreased. In the following experiments, the number of blocks of the input darrays is equal to the total number of instances.

Speedup for logistic regression

To compare the speedup of hpdglm to glm for logistic regression, a set of experiments is performed on fairly small number of samples, 100M×7 (5.6GB). The dataset is selected small enough to be handled by glm. The speedup of hpdglm to glm is illustrated in Figure 1.

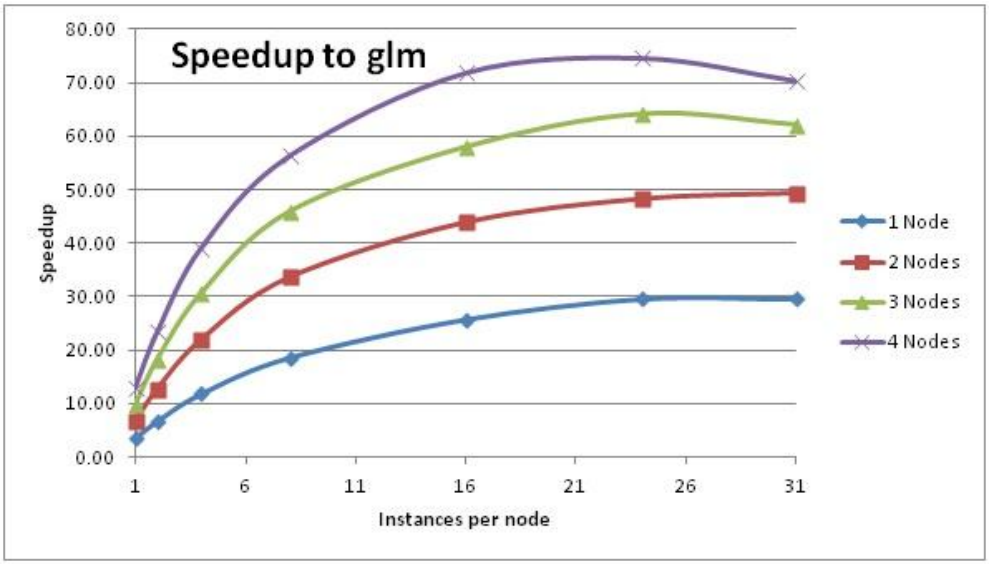

Figure 1. Speedup of hpdglm to glm for logistic regression

Speedup for linear regression

To compare the speedup of hpdglm to glm for linear regression, some experiments are performed on fairly small number of samples, 100M×7 (5.6GB). The dataset is selected small enough to be handled by glm. The speedup of hpdglm to glm is illustrated in Figure 2. Here, the number of instances per node is fixed to 23.

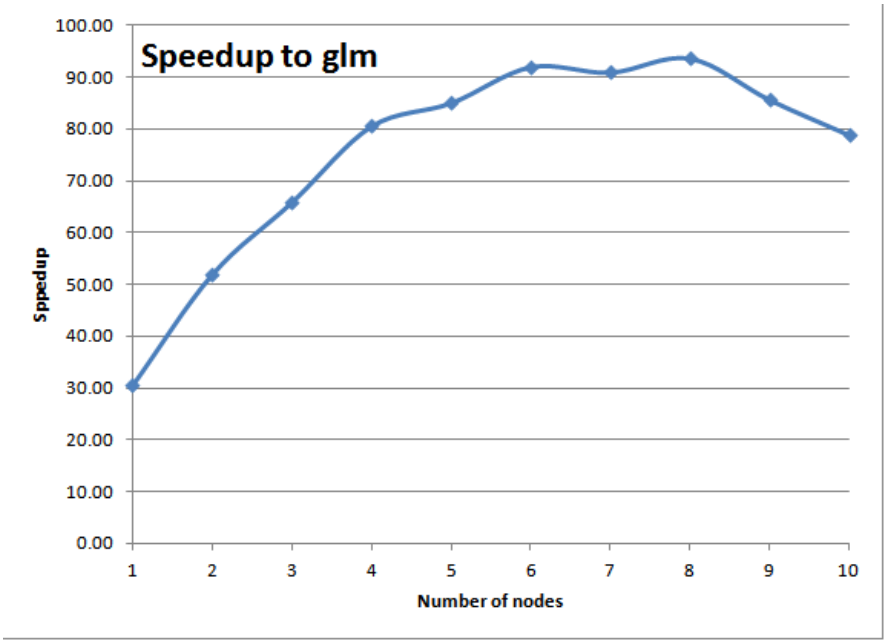

Figure 2. Speedup of hpdglm to glm for linear regression

Strong Scalability for linear regression

In this experiment the strong scalability of hpdglm for linear regression is measured using a dataset of size 750Mx7col (42GB). Here, the number of instances per node is fixed to 23. Figure 3 shows the running time of the experiments. The experiment is not reported for a single node because it was running out of memory.

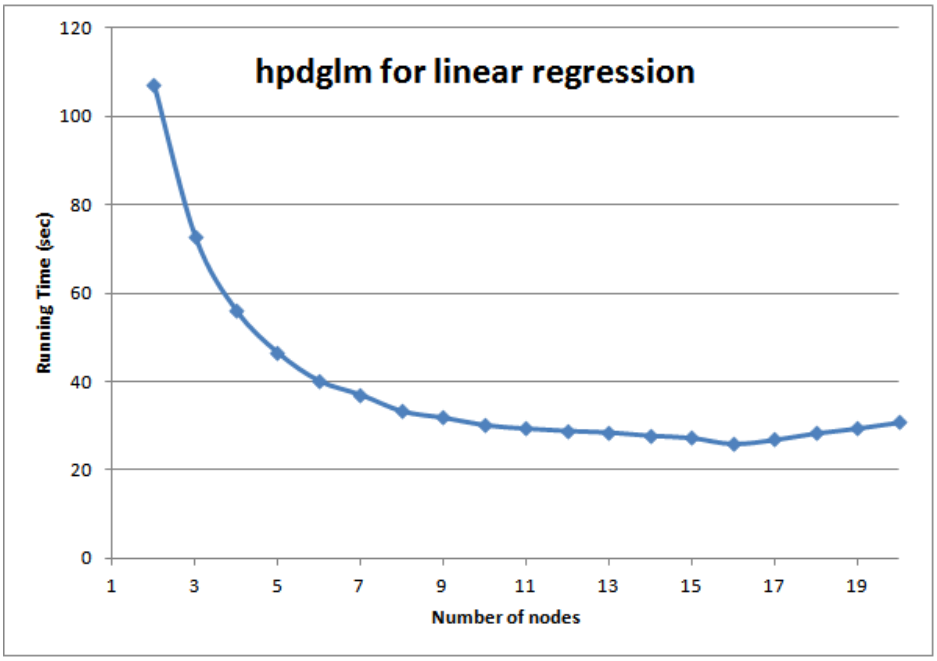

Figure 3. Running time of hpdglm for linear regression

Weak Scalability for linear regression

In this experiment the weak scalability of hpdglm for linear regression is measured. As presented in Table 2, the size of dataset increases along with cluster size (number of nodes). Ideally, the ratio of running times should remain the same. In reality, as it is illustrated in Figure 4 the performance is gradually deteriorating. Here, the number of instances per node is fixed to 23.

Table 2. Running time for measuring weak scalability

Number of nodes	Samples	Size of dataset (GB)	Running time (sec)
1	375Mx7col	21	106.524
2	750Mx7col	42	107.224
3	1125Mx7col	63	107.731
4	1500Mx7col	84	107.388
5	1875Mx7col	105	109.845
6	2250Mx7col	126	109.190
7	2625Mx7col	147	110.598
8	3000Mx7col	168	112.208
9	3375Mx7col	189	111.864
10	3750Mx7col	210	112.412

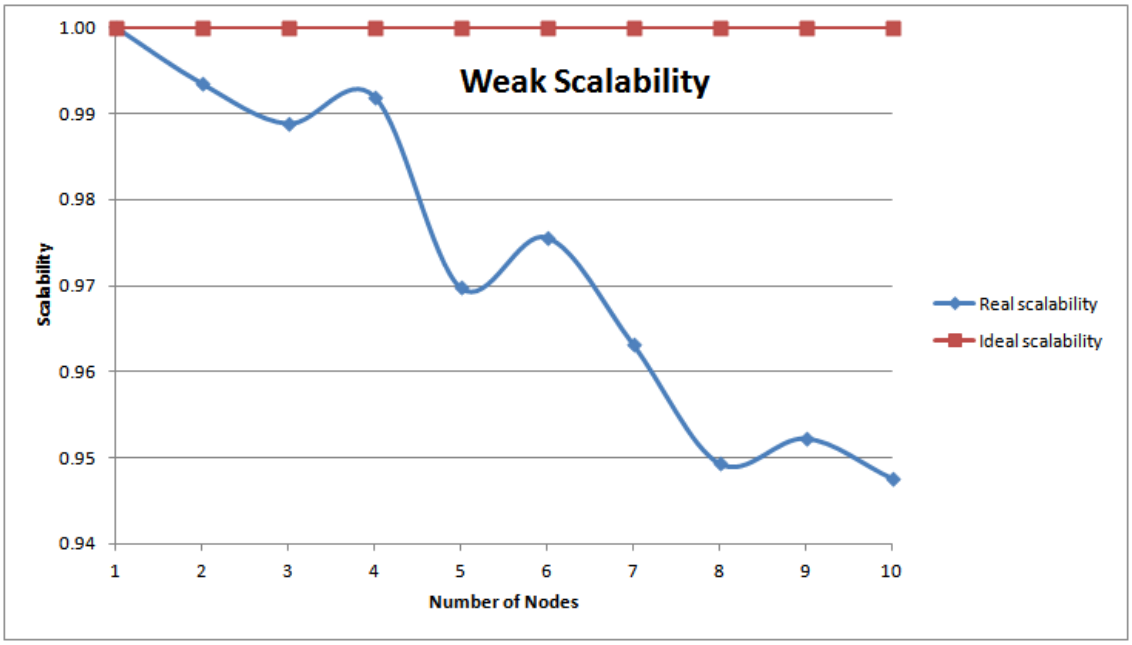

Figure 4. Weak Scalability: size of dataset grows as cluster size grows