Generalized Linear Modeling with H2O's R Package

November 2014

Contents

1 Introduction

This document describes Generalized Linear Model implementation on H2O platform, list of supported features and how to use them from R.

1.1 What is H2O?

H2O is an open source analytics platform for data scientists and business analysts who need scalable and fast machine learning capabilities. Our product helps organizations like PayPal, ShareThis, and Cisco reduce model building, training, and scoring times from months to days. Our use cases range from predictive modeling, fraud detection, and even customer intelligence in industries as diverse as insurance, SaaS, finance, ad tech, and recruiting.

With its in-memory compression techniques, H2O can handle billions of data rows in-memory even with a fairly small cluster. The platform includes interfaces for R, Python, Scala, Java, JSON and Coffeescript/JavaScript, along with its built-in web interface that makes it easier for non-engineers to stitch together a complete analytic workflow. The platform was built alongside (and on top of) both Hadoop and Spark Clusters.

H2O implements almost all common machine learning algorithms such as generalized linear modeling (linear regression, logistic regression, etc.), Naive Bayes, principal components analysis, time series, k-means clustering and others. H2O also implements best-in-class algorithms such as Random Forest, Gradient Boosting and Deep Learning at scale. Customers can build thousands of models and compare them to get the best prediction results.

H2O was built by a passionate team of computer scientists, systems engineers and data scientists, from the ground up, for the data science community. Were driven by strong curiosity, a desire to learn, and strong drive to tackle the scalability challenges of real world data analysis. Our team members have come to H2O from organizations as diverse as Marketo, Oracle, Azul, Teradata, and SAS. Our advisory board comes from Stanford Universitys engineering, statistics, and health research departments.

We host meetups, run experiments, and spend our days learning alongside our customers.

Try it out

H2O offers an R package that can be installed from CRAN. H2O can be downloaded from www.h2o.ai/download.

Join the community

Connect with h2ostream@googlegroups.com and https://github.com/h2oai to learn about our meetups, training sessions, hackathons, and product updates.

Learn more about H2O

Visit www.h2o.ai

1.2 What is GLM?

Generalized linear models (GLM) are the workhorse for most predictive analysis use cases. GLM can be used for both regression and classification, it scales well to large datasets and is based on solid statistical background. It is a ganrelaztion of linear models, allowing for modeling of data with exponential distributions and for categorical data (classification). GLM models are fitted solving the maximum likelihood optimization problem.

1.3 GLM on H2O

H2O's GLM algorithm fits the generalized linear model with elastic net penalties. The model fitting computation is distributed, extremely fast, and scales extremely well for models with a limited number (low thousands) of predictors with non-zero coefficients. The algorithm can compute models for a single value of a penalty argument or the full regularization path, similar to glmnet package for R[1]. H2O's GLM fits the model by solving following problem:

$$\min_{\beta} \frac{1}{N} log - likelihood(family, \beta) + \lambda(\alpha\|\beta\|_1 + \frac{1-\alpha}{2}\|\beta\|_2^2)$$

The elastic net parameter α controls the penalty distribution between L1 and L2 penalty. It can have any value between 0 and 1. When $\alpha = 0$, we have no L1 penalty and the problem becomes ridge regression. If $\alpha = 1$, there is no L2 penalty and we have lasso.

The main advantage of an L1 penalty is that with sufficiently high λ, it produces a sparse solution; the L2-only penalty does not reduce coefficients to exactly 0. The two penalties also differ in the case of correlated predictors. The L2 penalty shrinks coefficients for correlated columns towards each other, while the L1 penalty will pick one and drive the others to zero. Using the elastic net argument α, you can combine these two behaviors. It is also useful to always add a small L2 penalty to increase numerical stability.

Similarly to [1], H2O can compute the full regularization path, starting from null-model (maximum penalty) going down to minimally penalized model. This search is made efficient by employing strong-rules [2] to filter out inactive coefficients (coefficients pushed to zero by penalty). Computing full regularization path is useful in that it gives more insight about the importance of individual coefficients and quality of the model while allowing selection of the optimal amount of penalization for the given problem and data.

1.3.1 Summary of features

In summary, H2O's GLM functionalities include:

- fits generalized linear model with elastic net penalty
- supported GLM families include Gaussian, Binomial, Poisson and Gamma
- efficient handling of categorical variables
- efficient computation full regularization path
- efficient distributed n-fold cross validation
- distributed grid search over elastic-net parameter α
- upper and lower bounds for coefficients
- proximal operator interface

2 Installation

You can load the latest CRAN H2O package by running:

```
> install.packages("h2o")
```

Alternatively, you can (and should for this tutorial) download the latest H2O build by following the "Install in R" instructions in the H2O download table:
http://s3.amazonaws.com/h2o-release/h2o/master/latest.html.
Open your R Console and run the following to install the latest H2O build in R:

```
# The following two commands remove any previously installed H2O packages for R.
> if ("package:h2o" %in% search()) { detach("package:h2o", unload=TRUE) }
> if ("h2o" %in% rownames(installed.packages())) { remove.packages("h2o") }

# Next, we download, install and initialize the H2O package for R
# replace the *s below with the release number found on our download page
> install.packages("h2o", repos=(c("http://s3.amazonaws.com/h2o-release/h2o/
master/****/R", getOption("repos"))))

# Load h2o library in R
> library(h2o)
```

To launch on a single node, initialize H2O on all the cores of your machine with

```
> localH2O = h2o.init(nthreads = -1)
```

The function `h2o.init()` will initialize a H2O instance and instantiates a H2O client module. By default, the H2O instance will launch on `localhost:54321`. To establish a connection to an existing H2O cluster node, explicitly state the IP address (`ip = "localhost"`) and port number (`port = 54321`) in the `h2o.init()` call.
Run the following command to observe an example classification model built through H2O's GLM:

```
# Build a GLM model on prostate data with formula: CAPSULE ~ AGE + RACE + PSA + DCAPS
> prostatePath = system.file("extdata", "prostate.csv", package = "h2o")
> prostate.hex = h2o.importFile(localH2O, path = prostatePath, key = "prostate.hex")
> h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"), data = prostate.hex,
family = "binomial", nfolds = 0, alpha = 0.5, lambda_search = FALSE,
use_all_factor_levels = FALSE, variable_importances = FALSE, higher_accuracy = FALSE)
```

The output of the model build will include coefficients, as well as some validation statistics:

```
IP Address: 127.0.0.1
Port      : 54321
Parsed Data Key: prostate.hex
```

```
GLM2 Model Key: GLMModel__827586bb2c59ba79dc129b8500174940

Coefficients:
      AGE      RACE     DCAPS       PSA Intercept
 -0.01104  -0.63136   1.31888   0.04713  -1.10896

Normalized Coefficients:
      AGE      RACE     DCAPS       PSA Intercept
 -0.07208  -0.19495   0.40972   0.94253  -0.33707

Degrees of Freedom: 379 Total (i.e. Null);  375 Residual
Null Deviance:     512.3
Residual Deviance: 449.5  AIC: 459.5
Deviance Explained: 0.12254
 Best Threshold: 0.28

Confusion Matrix:
        Predicted
Actual   false true Error
  false     75  152 0.670
  true      18  135 0.118
  Totals    93  287 0.447

AUC =  0.7157151 (on train)
```

2.1 Support

Users of the H2O package may submit general enquiries and bug reports privately to H2O via email: support@h2o.ai; or publicly post them to: h2ostream@googlegroups.com. Specific bugs or issues will be filed to H2O's JIRA: https://0xdata.atlassian.net/secure/Dashboard.jspa.

3 Generalized Linear Modeling

This section contains a brief overview of generalized linear models and follows up with a few details for each model family.

Generalized linear models are generalization of linear regressions. Linear regression models the dependency of response y on a vector of predictors x ($y \sim x^T\beta + \beta_0$). The models are built with the assumptions that y has a gaussian distribution with a variance of σ^2 and the mean is a linear function of x with an offset of some constant β_0, i.e. $y = \mathcal{N}(x^T\beta + \beta_0, \sigma^2)$. These assumptions can be overly restrictive for real-world data that do not necessarily follow have a gaussian distribution. GLM generalizes linear regression in the following ways:

- adds a non-linear link function that transforms the expectation of response variable, so that $link(y) \sim x^T\beta + \beta_0$.

- allows variance to depend on the predicted value by specifying the conditional distribution of the response variable or the family argument.

This generalization allows us to use GLM on problems such as binary classification (Logistic regression).

3.0.1 Model Fitting

GLM models are fitted by maximizing the likelihood. For the gaussian family, maximum likelihood is simply the minimal mean squared error, which has an analytical solution and can be solved with ordinary least squares. For all other families, the maximum likelihood problem has no analytical solution so we must use an iterative method such as IRLSM, Newton method, gradient descent, and L-BFGS.

3.0.2 Model Validation

Evaluating the quality of the model is a critical part of any data-modeling and scoring process. There are several standard ways on how to evaluate the quality of the fitted GLM model; the most common method is to use the resulting deviance. Deviance is calculated by comparing the log likelihood of the fitted model with log likelihood of the saturated model (or theoretically perfect model).

$$deviance = 2(ln(L_s) - ln(L_m))$$

Another metric frequently used for model selection is the Akaike information criterion (AIC). AIC is a measure of the relative quality of a statistical model for a given set of data that is obtained by calculating the information loss when replacing the original data with the model itself. Unlike deviance, which would assign a perfect value for the saturated model and measures the absolute quality of the fit with a comparison against the null-hypothesis, it takes into account the complexity of the given model. AIC is defined as follows:

$$aic = 2(k - ln(L_m))$$

where k is the number of model parameters and $ln(L_m)$ is the log likelihood of the fitted model over the data.

3.1 Regularization with Elastic Net Penalty

We introduce penalty to model-fitting to avoid over-fitting, reduce variance of the prediction error, and deal with correlated predictors. There are two common penalized linear models: Ridge Regression and Lasso. Ridge regression provides greater numerical stability and is easier (faster) to compute. On the other hand, Lasso leads to a sparse solution, which is a big advantage in many situations, as it can be used for feature selection and to produce models with fewer parameters. When encountering highly correlated columns, the L2 penalty tends to push all of the coefficients towards each other, while the L1 penalty will pick one and remove the others (0 coefficients).

Elastic net combines the two and adds another parameter, α, which controls distribution of the penalty between L1 and L2. The combination of the two penalties is beneficial, since

L1 gives sparsity while L2 gives stability and encourages the grouping effect (where a group of correlated variables tends to be dropped or added into the model all at once).One possible use of the α argument is to do lasso with very little L2 penalty (α almost 1) to stabilize the computation (improve convergence speed)

Model-fitting problem with elastic net penalty becomes:

$$\min_{\beta,\beta_0} \frac{1}{N} ln(L(family, \beta, \beta_0)) + \lambda(\alpha\|\beta\|_1 + \frac{1-\alpha}{2}\|\beta\|_2^2)$$

3.2 GLM Models

The following subsection describes GLM families supported in H2O.

3.2.1 Linear Regression (Gaussian family)

Linear regression refers to the gaussian family model. It is the simplest example of GLM, but it has many uses and several advantages over the other families, such as faster and more stable computation.

It models the dependency as a purely linear function (with link = identity):

$$\hat{y} = x^T\beta + \beta_0$$

The model is fitted by solving the least squares problem (maximum likelihood for gaussian family):

$$\min_{\beta,\beta_0} \frac{1}{2N}\sum_{i=1}^{N}(x_i^T\beta + \beta_0 - y_i)^T(x_i^T\beta + \beta_0 - y_i) + \lambda(\alpha\|\beta\|_1 + \frac{1-\alpha}{2})\|\beta\|_2^2$$

Deviance is simply the sum of squared errors:

$$D = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Example

Included in the H2O package is a prostate cancer data set. The data was collected for a study done by Dr. Donn Young at The Ohio State University Comprehensive Cancer Center of patients with varying degrees of prostate cancer. Below a model is built to predict the volume (VOL) of tumors obtained from ultrasounds based on features such as age and race.

```
> filepath = system.file("extdata", "prostate.csv",package = "h2o")
> prostate.hex = h2o.importFile(object = localH2O, filepath, key = "prostate.hex")
> gaussian.fit = h2o.glm(x = c("AGE", "RACE", "PSA", "GLEASON"), y = "VOL", data
= prostate.hex, family = "gaussian")
```

3.2.2 Logistic Regression (Binomial Family)

Logistic regression can be used in case of a binary classification problem, where the response is categorical with two levels. It models dependency as $Pr(y = 1|x)$. The canonical link for binomial family is logit (log of the odds) and its inverse is a logistic function that takes any real number on the input and projects it onto the 0,1 range (s-curve).

$$\hat{y} = Pr(y = 1|x) = \frac{e^{x^T\beta+\beta_0}}{1 + e^{x^T\beta+\beta_0}}$$

or alternatively:

$$log\frac{\hat{y}}{1-\hat{y}} = log\frac{Pr(y = 1|x)}{Pr(y = 0|x)} = x^T\beta + \beta_0$$

The model is fitted by solving:

$$\min_{\beta,\beta_0} \frac{1}{N}\sum_{i=1}^{N}(y_i(x_i^T\beta + \beta_0) - log(1 + e^{x_i^T\beta+\beta_0}) + \lambda(\alpha\|\beta\|_1 + \frac{1-\alpha}{2})\|\beta\|_2^2$$

Deviance is -2 log likelihood:

$$D = -2\sum_{i=1}^{N}(ylog(\hat{y}) + (1 - y)log(1 - \hat{y}))$$

Decision threshold

Example

Using the prostate data set, build a binomial model that classifies if there is penetration of the prostatic capsule (CAPSULE). Make sure the entries in the CAPSULE column are binary entries by using the `h2o.table()` function. Change the regression by setting the family to binomial.

```
> h2o.table(prostate.hex[,"CAPSULE"])
  row.names Count
1         0   227
2         1   153
> binomial.fit = h2o.glm(x = c("AGE", "RACE", "PSA", "GLEASON"), y = "CAPSULE", data
= prostate.hex, family = "binomial")
```

3.2.3 Poisson Models

Poisson regression is generally used in cases where the response represents counts and we assume errors have a Poisson distribution. In general, it can be applied to any data where the response is non-negative.

When building a Poisson model, we usually model dependency of the mean on the log scale, i.e. canonical link is log and prediction is:

$$\hat{y} = e^{x^T\beta+\beta_0}$$

The model is fitted by solving:

$$\min_{\beta,\beta_0} \frac{1}{N} \sum_{i=1}^{N} (y_i (x_i^T \beta + \beta_0) - e^{x_i^T \beta + \beta_0}) + \lambda(\alpha \|\beta\|_1 + \frac{1-\alpha}{2}) \|\beta\|_2^2$$

Deviance is

$$D = -2 \sum_{i=1}^{N} (ylog(\hat{y}) - y - \hat{y}$$

Example

Load the Insurance data from the MASS library and import into H2O. Run a poisson model that predicts the number of claims (Claims) based on the district of the policy holder (District), their age (Age), and the type of car they own (Group).

```
> library(MASS)
> data(Insurance)
> insurance.hex = as.h2o(localH2O, Insurance)
> poisson.fit = h2o.glm(x = c("District", "Group", "Age"), y = "Claims",
data = insurance.hex, family = "poisson")
```

3.2.4 Gamma Models

The gamma distribution is useful for modeling a positive continuous response variable, where the conditional variance of the response grows with its mean but the coefficient of variation of the response $\sigma^2(x)/(x)$ is constant for all x, i.e., it has a constant coefficient of variation.

It is usually used with inverse or log link, inverse is the canonical link.

The model is fitted by solving:

$$\min_{\beta,\beta_0} \frac{1}{N} \sum_{i=1}^{N} \frac{y_i}{(x_i^T \beta + \beta_0)} - log(x_i^T \beta + \beta_0) + \lambda(\alpha \|\beta\|_1 + \frac{1-\alpha}{2}) \|\beta\|_2^2$$

Deviance is

$$D = -2 \sum_{i=1}^{N} log(\frac{y_i}{\hat{y}_i}) - \frac{y_i - \hat{y}_i}{\hat{y}_i}$$

Example

To change the link function from the default inverse function to the log link function, modify the *link* argument.

```
> gamma.inverse <- h2o.glm(x=c("AGE","RACE","CAPSULE","DCAPS","PSA","VOL"), y="DPROS",
data=prostate.hex, family="gamma", link="inverse")
```

```
> gamma.log <- h2o.glm(x=c("AGE","RACE","CAPSULE","DCAPS","PSA","VOL"), y="DPROS",
data=prostate.hex, family="gamma", link="log")
```

4 GLM on H2O

This section describes the specifics of GLM implementation on H2O, such as selecting regularization parameters and handling of categoricals.

H2O's GLM implementation presents a high-performance distributed algorithm, which scales linearly with the number of rows and works extremely well for datasets with limited number of active predictors.

4.1 Input Parameters

4.1.1 Predictors & Response

Every model must specify its predictors and response. It is the equivalent formula object in R. Predictors and response are specified by *source* and x and y parameters, with an optional *offset* parameter.

source refers to a frame containing a training dataset. All predictors and the response (and offset, if there is one) must be part of this frame.

x contains the list of column names or column indices referring to vectors from the source frame; it can not contain periods.

y is a column name or index referring to a vector from the source frame.

offset is a column name or index referring to a vector from the source frame.

4.1.2 Family & Link

Family and Link are both optional parameters. The default family is *Gaussian* and the default link is a canonical link for the selected family. These are passed in as strings, e.g. *family='gamma', link = 'log'*. While it is possible to select something other than a canonical link, it can lead to an unstable computation. Recommended combinations are Gaussian and Log, or Inverse and Gamma with log.

4.1.3 Regularization

H2O supports elastic net regularization which is parametrized by *alpha* and *lambda* arguments (similarly to [1])

The *alpha* argument controls the elastic net penalty distribution to L1 and L2 norms. It can have any value in [0,1] range (inclusive) or a vector of values (triggers grid search). Alpha = 0 leads to **Ridge Regression**, alpha = 1 leads to **LASSO**.

The *lambda* argument controls the penalty strength; it can have any positive value or a vector of values (which triggers grid search). **Note:** Lambda values are capped at λ_{max}, which is the smallest λ s.t. the solution is empty model (all zeros except for intercept).

4.1.4 Lambda-Search

Lambda search is special case of automatic and efficient grid search over lambda argument and is described in its own section. Lambda search can be enabled by using the *lambda_search = T* option. It can be further parametrized by the *n_lambdas* and *lambda_min_ratio* parameters. *n_lambdas* specifies the number of lambda values on the regularization path *lambda_min_ratio* specifies the minimal lambda value to be computed as a ration of λ_{max}

4.2 Coefficient Constraints

Coefficient constraints allow you to set special conditions over the model coefficients. Currently supported constraints are upper and lower bounds and proximal operator [6] interface.

The constraints are specified as a frame with following vecs (matched by name, all vecs can be sparse):

- *names (mandatory)* - coefficient names.
- *lower_bounds (optional)* - coefficient lower bounds , must be $<= 0$
- *upper_bounds (optional)* - coefficient upper bounds , must be $>= 0$
- *beta_given (optional)* - specifies the given solution in proximal operator interface
- *rho (mandatory if beta_given is specified, otherwise ignored)* - specifies per-column L2 penalties on the distance from the given solution

The proximal operator interface allows you to run the GLM with a proximal penalty on a distance from a specified given solution. It has various uses: for example, it can be used as part of ADMM consensus algorithm to obtain unified solution over separate H2O clouds, or in Bayesian regression approximation.

4.3 H2O GLM Model Output

The detailed output of the GLM model varies depending on the distribution used for the model. In general, there are common parts of the output for all Families: Coefficients & Normalized Coefficients, Validation, and Prediction. We'll cover the model output from an R user's point of view.

First, let's build a simple GLM on a small dataset and see what we get out:

```
library(h2o)

# instantiate h2o
h <- h2o.init()

# path to the data
# path is split up so it's document friendly
data.bucket <- "https://raw.githubusercontent.com/h2oai/h2o/master/smalldata"
data.path <- paste(data.bucket, "/logreg/prostate_train.csv", sep = "")

# import the data from the url
hex <- h2o.importFile(h, data.path)

# build a binomial regression
m <- h2o.glm(x = 3:9, y = 2, data = hex, family = "binomial")  # no other features tweaked...
```

The default `show` of this model has the following output:

```
IP Address: 127.0.0.1
Port      : 54321
Parsed Data Key: prostate_train.hex

GLM2 Model Key: GLMModel__8b954fd700d924f3e9dee8717b8246ef

Coefficients:
      AGE      RACE     DPROS     DCAPS       PSA       VOL   GLEASON Intercept
 -0.07668  -0.10320   0.59479   0.13549   0.43841  -0.22215   1.19657  -0.49458

Normalized Coefficients:
      AGE      RACE     DPROS     DCAPS       PSA       VOL   GLEASON Intercept
 -0.07668  -0.10320   0.59479   0.13549   0.43841  -0.22215   1.19657  -0.49458

Degrees of Freedom: 304 Total (i.e. Null);  297 Residual
Null Deviance:     412.1
Residual Deviance: 301.9  AIC: 317.9
Deviance Explained: 0.26733
 Best Threshold: 0.44

Confusion Matrix:
        Predicted
Actual   false true Error
  false    147   34 0.188
  true      32   92 0.258
  Totals   179  126 0.216

AUC =  0.8318927 (on train)
```

Briefly, the output contains the IP address and port number of the H2O cluster where the model was trained. It includes the data key, the model key, the coefficients, and some model metrics. Let's look at these more in-depth.

4.3.1 Coefficients & Normalized Coefficients

Coefficients are the predictor weights, i.e. the actual model used for prediction.

If the *standardize* option is set, H2O returns another set of coefficients, *Normalized Coefficients*. These are the predictor weights of the standardized data and are included only for informative purposes (e.g. to compare relative variable importance). In this case, the *Coefficients* are obtained from *Normalized Coefficients* by **reversing** the data standardization process (de-scaled, intercept adjusted by added offset) so that they can be applied to data in its original form (i.e. no standardization prior to scoring). *Note:* These are **not** the same as coefficients of a model built on non-standardized data.

4.3.2 Validation

H2O's GLM performs 10-fold cross-validation by default. It does this by generating additional models with 1/10 of the original data. This is fast and gives some idea of how the model will generalize. However, it's always best to evaluate the model using some holdout data.

4.3.3 Generating Predictions

The following R code generates predictions:

```
test.path <- paste(data.bucket, "/logreg/prostate_test.csv", sep = "")

# import the data from the url
test <- h2o.importFile(h, test.path)

# generate the predictions
predictions <- h2o.predict(m, test)

# look at the first 6 entries of predictions
predictions

# generate the model metrics
h2o.performance(data=predictions[,3], reference=test[,1])
```

4.4 Categorical Variables

When applying linear models to datasets with categorical variables, the usual approach is to expand the categoricals into a set of binary vectors, with one vector per each categorical level (e.g. by calling `model.matrix` in R). H2O performs similar expansions automatically and no prior changes to the dataset are needed. Each categorical column is treated as a set of sparse binary vectors.

4.4.1 Largest Categorical Trick

Categoricals have special handling during GLM computation as well. When forming the gram matrix, we can exploit the fact that columns belonging to the same categorical never co-occur and the gram matrix region belonging to these columns will not have any non-zero elements outside of the diagonal. We can thus keep it in sparse representation taking only O(N) elements instead of O(N*N). Furthermore, the complexity of Choelsky decomposition of a matrix that starts with a diagonal region can be greatly reduced. H2O's GLM exploits these two facts to handle the largest categorical "for free". Therefore, when analyzing the performance of GLM in the equation expressed above, we can subtract the size of the largest categoricals from the number of predictors.

$$N = \sum_{c \in C} (\|c.domain\|) - \underset{c \in C}{\operatorname{argmax}} \|c.domain\| + \|Nums\|$$

4.5 Cross-Validation

All validation values can be computed either on the training data set (the default) or using nfold cross-validation ($nfolds > 1$). When using nfold cross-validation, we randomly split data into n equally-sized parts and train each of the n models on n-1 parts and compute validation on the part which was not used for training.The reported validation parameters are then obtained as follows:

- null deviance is sum of null deviances of n-models (each uses null model based on the subset of the data)
- residual deviance is sum of residual deviances of all n-models.
- AIC is based on log-likelihood, which is summed up similarly to deviance
- AUC is based on ROC curve build by summing up confusion matrices built for all n-models. This means for each threshold, we get a confusion matrix that includes all the rows from the training set. However, each row is classified exclusively by the model that did not have it in its training set. The computation of AUC itself is then the same as in a non-cross-validated case.

4.6 Selecting Regularization Parameters

To get the best possible model, we need to find the optimal values of the regularization parameters α and λ. To this end, H2O provides grid search over α and a special form of grid search called "lambda search over λ". The recommended way to find optimal regularization settings on H2O is to do a grid search over a few α values with an automatic lambda search for each α. Both are described below in greater detail.

4.6.1 Grid Search Over Alpha

Alpha search is not always needed and simply changing its value to 0.5 (or 0 or 1 if we only want Ridge or Lasso, respectively) works in most cases. If α search is needed, usually only a few values are sufficient. Alpha search is invoked by supplying a list of values for α instead of a single value. H2O then produces one model per α value. The grid search computation can be done in parallel (depending on the cluster resources) and it is generally more efficient than computing different models separately from R.

Use caution when including $\alpha = 0$ or $\alpha = 1$ in the grid search. $\alpha = 0$ will produce a dense solution and it can be really slow (or even impossible) to compute in large N situations. $\alpha = 1$ has no L2 penalty, so it is therefore less numerically stable and can be very slow as well due to slower convergence. In general, it is safer to run with $alpha = 1 - \epsilon$ instead.

4.6.2 Lambda Search

Lambda search can be enabled by using the *lambda_search* = *T* option. It can be further parametrized by the *n_lambdas* and *lambda_min_ratio* parameters. When this option is enabled, H2O performs a specialized grid search over the list of *n_lambdas* λ values, producing one model each per λ value.

The λ-list is automatically generated as an exponentially decreasing sequence, going from λ_{max}, the smallest λ s.t.the solution is a model with all 0s, to $\lambda_{min} = lambda\_min\_ratio * \lambda_{max}$.

H2O computes λ-models sequentially and in decreasing order, warm-starting the model for λ_k with the solution for λ_{k-1}. By warm-starting the models, we get better performance: typically models for subsequent λs are close to each other, so we need only a few iterations per λ (typically 2 or 3). We also achieve greater numerical stability, since models with a higher penalty are easier to compute; so, we start with an easy problem and then keep making only small changes to it.

Note: *nlambda*, *lambda.min.ratio* also specify the relative distance of any two lambdas in the sequence. This is important for the application of recursive strong rules, which are only effectove if the neighbouring lambdas are *"close"* to each other. The default values are *nlambda* = 100 and $\lambda_{min} = \lambda_{max} 1e^{-4}$, which gives us the ratio of. In order for strong rules to work, you should keep the ratio close to the default.

4.6.3 Grid Search Over Lambdas

While automatic lambda search is the preferred method, it is also possible to do a grid search over lambda values by passing in vector of lambdas and disabling the lambda-search option. The behavior will be identical to lambda search, except H2O will use a user-supplied list of lambdas instead (still capped at λ_{max}).

4.7 Strong Rules

H2O's GLM employs strong rules [2] to discard predictors that are likely to have 0 coefficients prior to model building. According to [2], we can identify such predictors based on a gradient with great accuracy (very few false negatives and virtually no false positives in practice), greatly reducing the computational complexity of model fitting and enabling it to run on wide datasets with tens of thousands of predictors, provided that there is a limited number of active predictors.

When applying the strong rules, we evaluate the gradient at the starting solution, filter out inactive coefficients, and fit a model using only a subset of the available predictors. Since strong rules may have false positives (which are extremely rare in practice), we need to check the solution by testing the kkt conditions and verify that all discarded predictors indeed have 0 coefficients.

4.8 Performance Characteristics

The implementation is based on iterative reweighted least squares with ADMM [5] inner solver to deal with L1 penalty. Every iteration of the algorithm consists of following steps:

1. Generate weighted least squares problem based on previous solution, i.e. vector of weights w and response z
2. Compute the weighted gram matrix $X^T W X$ and $X^T z$ vector
3. Decompose the gram matrix (Cholesky decomposition) and apply ADMM solver to solve the L1 penalized least squares problem

Steps 1 and 2 are performed distributively and step 3 is computed in parallel on a single node. We can thus characterize the computational complexity and scalability of dataset with M observations and N columns (predictors) on a cluster with n nodes with p CPUs each as follows:

$$O(\frac{MN^2}{pn} + \frac{N^3}{p})$$

And the overall memory cost is given by:

$$O(MN + N^2pn)$$

In case of $M >> N$, we can forget the second term and the algorithm scales linearly both in the number of nodes and number of CPUs per node. However, the equation above also implies that our algorithm is limited in the number of predictors it can handle, since the size of the Gram matrix grows quadratically (due to a memory and network throughput issue) with the number of predictors and its decomposition cost grows as the cube of the number of predictors (which is computational cost issue). H2O can get around these limitations in many cases due to its handling of categoricals and by employing strong rules to filter out inactive predictors; both are described later in this chapter.

5 Use case: Classification with Airline data

5.1 Airline dataset overview

The Airline dataset can be downloaded here: https://github.com/h2oai/h2o/blob/master/smalldata/airlines/allyears2k_headers.zip. Remember to save the .csv file to your working directory by clicking "View Raw." Before running the Airline demo, we'll first review how to load data with H2O.

5.1.1 Loading data

Loading a dataset in R for use with H2O is slightly different from the usual methodology because we must convert our datasets into `H2OParsedData` objects. In this example, we will use a toy weather dataset that can be downloaded here:
https://raw.githubusercontent.com/h2oai/h2o/master/smalldata/weather.csv. First, load the data to your current working directory in your R Console (do this for any future dataset downloads), and then run the following command:

```
weather.hex = h2o.uploadFile(localH2O, path = "weather.csv", header = TRUE, sep =
",", key = "weather.hex")
```

To see a quick summary of the data, run the following command:

```
summary(weather.hex)
```

5.2 Performing a trial run

Returning to the Airline dataset demo, we first load the dataset into H2O and select the variables we want to use to predict a chosen response. For example, we can model if flights are delayed based on the departure's scheduled day of the week and day of the month.

```
library(h2o)
localH2O = h2o.init(nthreads = -1)
#Load the data and prepare for modeling
air_train.hex = h2o.uploadFile(localH2O, path = "~/Downloads/AirlinesTrain.csv",
                               header = TRUE, sep = ",", key = "airline_train.hex")
air_test.hex = h2o.uploadFile(localH2O, path = "~/Downloads/AirlinesTest.csv",
                              header = TRUE, sep = ",", key = "airline_test.hex")
x = c("fYear", "fMonth", "fDayofMonth", "fDayOfWeek", "UniqueCarrier", "Origin",
      "Dest", "Distance")
y = "IsDepDelayed"
```

Now we train the GLM model:

```
airline.glm <- h2o.glm(x=x,
                       y=y,
                       data=air_train.hex,
                       key = "glm_model",
                       family="binomial",
                       lambda_search = TRUE,
                       return_all_lambda = TRUE,
                       use_all_factor_levels = TRUE,
                       variable_importances = TRUE)
```

5.2.1 Extracting and handling the results

We can extract the parameters of our model, examine the scoring process, and make predictions on new data.

```
print("Predict on GLM model")
best_glm = airline.glm@models[[airline.glm@best_model]]
air.results = h2o.predict(object = best_glm, newdata = air_test.hex)
print("Check performance and AUC")
perf = h2o.performance(air.results$YES,air_test.hex$IsDepDelayed )
print(perf)
perf@model$auc
print("Show distribution of predictions with quantile.")
quant = quantile.H2OParsedData(air.results$YES)
print("Extract strongest predictions.")
top.air <- h2o.assign(air.results[air.results$YES > quant["75%"]], key="top.air")
top.air
```

Once we have a satisfactory model, the `h2o.predict()` command can be used to compute and store predictions on the new data, which can then be used for further tasks in the interactive modeling process.

```
#Perform classification on the held out data
prediction = h2o.predict(object = best_glm, newdata=air_test.hex)
#Copy predictions from H2O to R
pred = as.data.frame(prediction)
head(pred)
```

5.3 Web interface

H2O R users have access to an intuitive web interface that mirrors the model building process in R. After loading data or training a model in R, point your browser to your IP address and port number (e.g., localhost:54321) to launch the web interface. From here, you can click on Admin > Jobs to view your specific model details. You can also click on Data > View All to track datasets currently in use.

5.3.1 Variable importances

One useful feature is the variable importances option, which can be enabled with the additional argument `importance=TRUE`. This feature allows us to view the absolute and relative predictive strength of each feature in the prediction task. From R, you can access these strengths with the command `air.model@model$varimp`. You can also view a visualization of the variable importances on the web interface.

5.3.2 Java model

Java models are currently not available for GLM.

To download the model,

1. Open the terminal window.
2. Create a directory where the model will be saved.
3. Set the new directory as the working directory.
4. Follow the curl and java compile commands displayed in the instructions at the top of the Java model.

6 Appendix: Parameters

- `x`: A vector containing the names of the predictors in the model. No default.
- `y`: The name of the response variable in the model. No default.
- `data`: An `H2OParsedData` object containing the training data. No default.
- `key`: The unique hex key assigned to the resulting model. If none is given, a key will automatically be generated.

- `family`: A description of the error distribution and corresponding link function to be used in the model. Currently, Gaussian, binomial, Poisson, gamma, and Tweedie are supported. When a model is specified as Tweedie, users must also specify the appropriate Tweedie power. No default.
- `link`: The link function relates the linear predictor to the distribution function. Default is the canonical link for the specified family. The full list of supported links:
  - **gaussian**: identity, log, inverse
  - **binomial**: logit, log
  - **poisson**: log, identity
  - **gamma**: inverse, log, identity
  - **tweedie**: tweedie
- `nfolds`: A logical value indicating whether the algorithm should conduct classification. Otherwise, regression is performed on a numeric response variable.
- `nfolds`: Number of folds for cross-validation. Default is 0.
- `validation`: An `H2OParsedData` object indicating the validation dataset used to construct confusion matrix. If blank, the default is the training data.
- `alpha`: The elastic-net mixing parameter, which must be in $[0, 1]$. The penalty is defined to be $P(\alpha, \beta) = (1-\alpha)/2||\beta||_2^2 + \alpha||\beta||_1 = \sum_j [(1-\alpha)/2\beta_j^2 + \alpha|\beta_j|]$ so `alpha=1` is the lasso penalty, while `alpha=0` is the ridge penalty. Default is 0.5.
- `nlambda`: The number of lambda values when performing a search. Default is -1.
- `lambda.min.ratio`: Smallest value for lambda as a fraction of lambda.max, the entry value, which is the smallest value for which all coefficients in the model are zero. Default is -1.
- `lambda`: The shrinkage parameter, which multiplies $P(\alpha, \beta)$ in the objective. The larger lambda is, the more the coefficients are shrunk toward zero (and each other). Default is 1e-5.
- `epsilon`: Number indicating the cutoff for determining if a coefficient is zero. Default is 1e-4.
- `standardize`: Logical value indicating whether the data should be standardized (set to mean = 0, variance = 1) before running GLM. Default is true.
- `prior`: Prior probability of class 1. Only used if `family = "binomial"`. Default is the frequency of class 1 in the response column.
- `variable_importances`: A logical value (either TRUE or FALSE) to indicate whether the variable importances should be computed. Compute variable importances for input features. NOTE: If `use_all_factor_levels` is disabled, the importance of the base level will NOT be shown. Default is false.

- `use_all_factor_levels`: A logical value (either TRUE or FALSE) to indicate whether all factor levels should be used. By default, the first factor level is skipped from the possible set of predictors. Set this flag if you want use all of the levels. Note: Needs sufficient regularization to solve! Default is false.
- `tweedie.p`: The index of the power variance function for the tweedie distribution. Only used if `family = "tweedie"`. Default is 1.5.
- `iter.max`: Maximum number of iterations allowed. Default is 100.
- `higher_accuracy`: A logical value indicating whether to use line search. This will cause the algorithm to run slower, so generally, it should only be set to TRUE if GLM does not converge otherwise. Default is false.
- `lambda_search`: A logical value indicating whether to conduct a search over the space of lambda values, starting from `lambda_max`. When this is set to TRUE, lambda will be interpreted as `lambda_min`. Default is false.
- `return_all_lambda`: A logical value indicating whether to return every model built during the lambda search. Only used if `lambda_search = TRUE`. If `return_all_lambda = FALSE`, then only the model corresponding to the optimal lambda will be returned. Default is false.
- `max_predictors`: When `lambda_search = TRUE`, the algorithm will stop training if the number of predictors exceeds this value. Ignored when `lambda_search = FALSE` or `max_predictors = -1`. Default is -1.
- `offset`: Column to be used as an offset, if you have one. No default.
- `has_intercept`: A logical value indicating whether or not to include the intercept term. If there are factor columns in your model, then the intercept must be included. Default is true.

References

[1] Jerome Friedman, Trevor Hastie, Rob Tibshirani Regularization Paths for Generalized Linear Models via Coordinate Descent April 29, 2009

[2] Robert Tibshirani, Jacob Bien, Jerome Friedman, Trevor Hastie, Noah Simon, Jonathan Taylor, and Ryan J. Tibshirani Strong Rules for Discarding Predictors in Lasso-type Problems J. R. Statist. Soc. B, vol. 74, 2012.

[3] Hui Zou and Trevor Hastie Regularization and variable selection via the elastic net J. R. Statist. Soc. B (2005) 67, Part 2, pp. 301320

[4] Robert Tibshirani Regression Shrinkage and Selection via the Lasso Journal of the Royal Statistical Society. Series B (Methodological), Volume 58, Issue 1 (1996), 267-288

[5] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers Foundations and Trends in Machine Learning, 3(1):1122, 2011. (Original draft posted November 2010.)

[6] N. Parikh and S. Boyd Proximal Algorithms Foundations and Trends in Optimization, 1(3):123-231, 2014.

[7] http://github.com/h2oai/h2o.git

[8] http://docs.h2o.ai

[9] https://groups.google.com/forum/#!forum/h2ostream

[10] http://h2o.ai

[11] https://0xdata.atlassian.net/secure/Dashboard.jspa