Generalized Linear Modeling with H2O

Tomas Nykodym Ariel Rao Amy Wang Tom Kraljevic Jessica Lanford

---

http://h2o.gitbooks.io/glm-with-h2o/

August 2015: Third Edition

Generalized Linear Modeling with H2O
by Tomas Nykodym, Ariel Rao, Amy Wang, Tom Kraljevic, & Jessica Lanford

Published by H2O.ai, Inc.
2307 Leghorn St.
Mountain View, CA 94043

August 2015: Third Edition

Printed in the United States of America.

Contents

1 Introduction

This document introduces the reader to Generalized Linear Modeling with H2O. Examples are written in R and python. The reader is walked through the installation of H2O, basic GLM concepts, building GLM models in H2O, how to interpret model output, how to make predictions, and various implementation details.

2 What is H2O?

H2O is fast, scalable, open-source machine learning and deep learning for Smarter Applications. With H2O, enterprises like PayPal, Nielsen, Cisco, and others can use all their data without sampling to get accurate predictions faster. Advanced algorithms, like Deep Learning, Boosting, and Bagging Ensembles are built-in to help application designers create smarter applications through elegant APIs. Some of our initial customers have built powerful domain-specific predictive engines for Recommendations, Customer Churn, Propensity to Buy, Dynamic Pricing, and Fraud Detection for the Insurance, Healthcare, Telecommunications, AdTech, Retail, and Payment Systems industries.

Using in-memory compression, H2O handles billions of data rows in-memory, even with a small cluster. To make it easier for non-engineers to create complete analytic workflows, H2O's platform includes interfaces for R, Python, Scala, Java, JSON, and Coffeescript/JavaScript, as well as a built-in web interface, Flow. H2O was built alongside (and on top of) Hadoop and Spark Clusters and typically deploys within minutes.

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

H2O is nurturing a grassroots movement of physicists, mathematicians, and computer scientists to herald the new wave of discovery with data science by collaborating closely with academic researchers and Industrial data scientists. Stanford university giants Stephen Boyd, Trevor Hastie, Rob Tibshirani advise the H2O team on building scalable machine learning algorithms. With hundreds of meetups over the past two years, H2O has become a word-of-mouth phenomenon, growing amongst the data community by a hundred-fold, and is now used by 12,000+ users and is deployed using R, Python, Hadoop, and Spark in 2000+ corporations.

Try it out

H2O's R package can be installed from CRAN at `https://cran.r-project.org/web/packages/h2o/`. A Python package can be installed from PyPI at `https://pypi.python.org/pypi/h2o/`. Download H2O directly from `http://h2o.ai/download`.

Join the community

Visit the open source community forum at `https://groups.google.com/d/forum/h2ostream`. To learn about our meetups, training sessions, hackathons, and product updates, visit `http://h2o.ai`.

3 Installation

The easiest way to directly install H2O is via an R or Python package.

(**Note**: This document was created with H2O version 3.0.1.4.)

3.1 Installation in R

To load a recent H2O package from CRAN, run:

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

(**Note**: The version of H2O in CRAN is often one release behind the current version.

Alternatively, you can (and should for this tutorial) download the latest stable H2O-3 build from the H2O download page:

1. Go to `http://h2o.ai/download`.
2. Choose the latest stable H2O-3 build.
3. Click the "Install in R" tab.
4. Copy and paste the commands into your R session.

After H2O is installed on your system, verify the installation:

```
library(h2o)

# Start H2O on your local machine using all available cores.
# By default, CRAN policies limit use to only 2 cores.
h2o.init(nthreads = -1)

# Get help
?h2o.glm
?h2o.gbm

# Show a demo
demo(h2o.glm)
demo(h2o.gbm))
```

3.2 Installation in Python

To load a recent H2O package from PyPI, run:

```
pip install h2o
```

Alternatively, you can (and should for this tutorial) download the latest stable H2O-3 build from the H2O download page:

1. Go to `http://h2o.ai/download`.
2. Choose the latest stable H2O-3 build.
3. Click the "Install in Python" tab.
4. Copy and paste the commands into your Python session.

After H2O is installed, verify the installation:

```
import h2o

# Start H2O on your local machine
h2o.init()

# Get help
help(h2o.glm)
help(h2o.gbm)

# Show a demo
h2o.demo("glm")
h2o.demo("gbm")
```

3.3 Pointing to a different H2O cluster

Following the instructions in the previous sections create a one-node H2O cluster on your local machine.

To connect to an established H2O cluster (in a multi-node Hadoop environment, for example) specify the IP address and port number for the established cluster using the `ip` and `port` parameters in the `h2o.init()` command:

```
h2o.init(ip="123.45.67.89", port=54321)
```

3.4 Example code

R code for the examples in this document are available here:

`https://github.com/h2oai/h2o-3/blob/master/h2o-docs/src/booklets/v2_2015/source/GLM_Vignette.R`

Python code for the examples in this document can be found here:

`https://github.com/h2oai/h2o-3/blob/master/h2o-docs/src/booklets/v2_2015/source/GLM_Vignette.py`

The document source itself can be found here:

`https://github.com/h2oai/h2o-3/blob/master/h2o-docs/src/booklets/v2_2015/source/GLM_Vignette.tex`

4 Generalized Linear Models

Generalized linear models (GLMs) are an extension of traditional linear models. They have gained popularity in statistical data analysis because of (1) the flexibility of the model structure which unifies the typical regression methods (such as linear regression and logistic regression for binary classification) and (2) the availability of software to fit the models. GLMs scale well to large datasets.

4.1 Model components

The estimation of the model is obtained by maximizing the log-likelihood of the parameter vector β for the observed data as such

$$\max_{\beta} \text{ (GLM Log-likelihood) .}$$

In the familiar linear regression setup, the independent observations vector y is assumed to be related to its corresponding predictor vector x by

$$y = x^\top \beta + \beta_0 + \epsilon,$$

where β is the parameter vector, β_0 represents the intercept term and $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is a gaussian random variable which is the noise in the model. Therefore the response y is also normally distributed $y \sim \mathcal{N}(x^\top \beta + \beta_0, \sigma^2)$. This model is restrictive as it assumes additivity of the covariates, normality of the error term as well as constancy of the variance. Such assumptions do not hold in numerous applications and datasets, thus a more flexible model is desirable.

GLMs relax the above assumptions by allowing the variance to vary as a function of the mean, non-normal errors as well as a non-linear relation between the response and covariates. More specifically, the response distribution is assumed to belong to the exponential family which includes the gaussian, poisson, binomial, multinomial and gamma distributions as special cases. The components of a GLM are:

- The random component f for the dependent variable y: the density function $f(y; \theta, \phi)$ has a probability distribution from the exponential family parametrized by θ and ϕ. This removes the restriction on the distribution of the error and allows for non-homogeneity of the variance with respect to the mean vector.
- The systematic component η: $\eta = X\beta$, where X is the matrix of all observation vectors x_i.
- The link function g: $E(y) = \mu = g^{-1}(\eta)$ which relates the expected value of the response μ to the linear component η. The link function can be any monotonic differentiable function. This

relaxes the constraints on the additivity of the covariates, and it allows the response to belong to a restricted range of values depending on the chosen transformation g.

This generalization of linear models makes GLMs suitable for a wider range of problems which will be discussed below. For example, the familiar logistic regression model which is widely used for binary classification in medical applications is a particular case of the GLM representation.

4.2 GLM in H2O

H2O's GLM algorithm fits generalized linear models to the data by maximizing the log-likelihood. The elastic net penalty can be used for parameter regularization. The model fitting computation is distributed, extremely fast, and scales extremely well for models with a limited number of predictors with non-zero coefficients ($\sim$ low thousands). H2O's GLM fits the model by solving the following likelihood optimization with parameter regularization:

$$\max_{\beta,\beta_0} \text{ (GLM Log-likelihood } - \text{Regularization Penalty)}.$$

The elastic net regularization penalty is the weighted sum of the ℓ_1 and ℓ_2 norms of the coefficients vector. It is defined as

$$\lambda P_\alpha(\beta) = \lambda \left(\alpha \|\beta\|_1 + \frac{1}{2}(1-\alpha)\|\beta\|_2^2 \right),$$

(with no penalty on the intercept term). It is implemented by subtracting $\lambda P_\alpha(\beta)$ from the likelihood that we are optimizing. This shrinks the coefficients by imposing a penalty on their size and induces sparsity in the solution. Such properties are desirable as they reduce the variance in the predictions and make the model more interpretable, since it selects a subset of the given variables. For a specific α value, the algorithm can compute models for (1) a single value of the tuning parameter λ or (2) the full regularization path as in the glmnet package for R (refer to Regularization Paths for Generalized Linear Models via Coordinate Descent by Friedman et. al).

The elastic net parameter $\alpha \in [0, 1]$ controls the penalty distribution between the ℓ_1 (least absolute shrinkage and selection operator or lasso) and ℓ_2 (ridge regression) penalties. When $\alpha = 0$, the ℓ_1 penalty is not used and we obtain the ridge regression solution with shrunken coefficients. If $\alpha = 1$, the lasso operator soft-thresholds the parameters which consists in reducing all of them by a constant factor and truncating at zero. This sets a different number of coefficients to zero depending on the λ value.

In summary, H2O's GLM solves the following optimization over N observations:

$$\max_{\beta,\beta_0} \sum_{i=1}^{N} \log f\left(y_i; \beta, \beta_0\right) - \lambda \left(\alpha \|\beta\|_1 + \frac{1}{2}(1-\alpha)\|\beta\|_2^2 \right).$$

Similarly to the methods discussed in Regularization Paths for Generalized Linear Models via Coordinate Descent by Friedman et. al, H2O can compute the full regularization path, starting from the null-model (evaluated at the smallest penalty $\lambda_{\max}$ for which all coefficients are set to zero) down to a minimally-penalized model. To improve the efficiency of this search, H2O employs the strong rules as described in Strong Rules for Discarding Predictors in Lasso-type Problems by Bien et. al to filter out

inactive columns (coefficients that will remain equal to zero given the imposed penalty). Computing the full regularization path is useful for convergence because it uses warm starts for consecutive λ values, and gives an insight regarding the order in which the coefficients start entering the model. Moreover, cross-validation can be used after fitting models for the full regularization path. H2O returns the optimal amount of regularization λ for the given problem and data by computing the error on the validation dataset of the model fit on the training data.

4.3 Model fitting

As discussed above, GLM models are fitted by finding the set of parameters that maximizes the likelihood of the data. For the Gaussian family, maximum likelihood simply corresponds to minimizing the mean squared error. This has an analytical solution and can be solved with a standard method of least squares. The same holds when the ℓ_2 penalty is added to the optimization. For all other families and when the ℓ_1 penalty is included, the maximum likelihood problem has no analytical solution. Thus, we must use an iterative method such as IRLSM, L-BFGS, the newton method, gradient descent and coordinate descent with the naive or covariance updates. The solver can be selected by the user after selecting the model by specifying the exponential density.

4.4 Model validation

After selecting the model, it is important to evaluate the precision of the estimates. The quality of the fitted model can be obtained by computing the goodness of fit between the predicted values that it generates and the given input data. Multiple measures of discrepancy may be used. H2O returns the logarithm of the ratio of likelihoods, called deviance, and the Akaike information criterion (AIC) after fitting a GLM. A benchmark for a good model is the saturated or full model, which is the largest model that can be fitted. Assuming the dataset consists of n observations, the saturated model fits n parameters $\hat{\mu}_i$. Since it gives a model with one parameter per observation, its predictions trivially fit the data perfectly.

The deviance is the difference between the maximized log-likelihoods of the fitted and saturated models. Let $\ell(y;\hat{\mu})$ be the likelihood corresponding to the estimated means vector $\hat{\mu}$ from the maximization, and let $\ell(y;y)$ be the likelihood of the saturated model which is the maximum achievable likelihood. The scaled deviance, defined as

$$D^*(y,\hat{\mu}) = 2\ell(y;y) - 2\ell(y;\hat{\mu}),$$

is used as a goodness of fit measure for GLMs. When the deviance obtained is too large, the model does not fit the data well.

Another metric to measure the quality of the fitted statistical model is the AIC, defined as

$$AIC = 2k - 2\log(\ell(y;\hat{\mu})),$$

where k is the number of parameters included in the model and ℓ is the likelihood of the fitted model defined as above. Given a set of models for a dataset, the AIC compares the qualities of the models with respect to one another. This provides a way to select the optimal one, which is the model with

the lowest AIC score. Note that the AIC score does not give an absolute measure of the quality of a given model. Moreover, the AIC takes into account the number of parameters that are included in the model by increasing the penalty as the number of parameters increases. This prevents from obtaining a complex model that overfits the data, an aspect which is not considered in the deviance computation.

4.5 Regularization

In this subsection, we discuss the details of the effect of parameter regularization. Penalties are introduced to the model building process to avoid over-fitting, reduce variance of the prediction error and handle correlated predictors. The two most common penalized models are ridge regression and the lasso. The elastic net combines both penalties.

4.5.1 Lasso and ridge regression

Ridge regression consists in penalizing the ℓ_2 norm of the model coefficients $\|\beta\|_2^2 = \sum_{k=1}^{p} \beta_k^2$. It provides greater numerical stability and is easier and faster to compute than the lasso. It keeps all the predictors in the model and does a proportional shrinkage. It reduces coefficient values simultaneously as the penalty is increased without however setting any of them to zero. The lasso stands for the ℓ_1 penalty and is an alternative regularized least squares method that penalizes the sum of the absolute values of the coefficients $||\beta||_1 = \sum_{k=1}^{p} |\beta_k|$. The lasso leads to a sparse solution when the tuning parameter is sufficiently large. As the tuning parameter value λ is increased, all coefficients are set to zero. Since reducing parameters to zero removes them from the model, the lasso is a good selection tool.

Variable selection is important in numerous modern applications with many covariates, for which the ℓ_1 penalty has proven to be successful. Therefore, if the number of variables is large or if the solution is known to be sparse, it is advantageous to use the lasso as it will select a small number of variables for sufficiently high λ which could be crucial to the interpretability of the model. The ℓ_2 norm does not have that effect, it only shrinks the coefficients without setting them exactly to zero.

The two penalties also differ in the presence of correlated predictors. The ℓ_2 penalty shrinks coefficients for correlated columns towards each other, while the ℓ_1 penalty tends to select only one of them and set the other coefficients to zero. Using the elastic net argument α, you can combine these two behaviors. The elastic net both selects variables and preserves the grouping effect (shrinking coefficients of correlated columns together). Moreover, while the number of predictors that can enter a lasso model saturates at $\min(n, p)$ where n is the number of observations and p is the number of variables in the model, the elastic net does not and can fit models with a larger number of predictors.

4.5.2 Elastic net penalty

H2O supports elastic net regularization, which is a combination of the ℓ_1 and ℓ_2 penalties parametrized by α and λ arguments (similar to Regularization Paths for Generalized Linear Models via Coordinate Descent by Friedman et. al).

- α controls the elastic net penalty distribution between the $\ell 1$ and ℓ_2 norms. It can have any value in the $[0, 1]$ range (inclusive) or a vector of values (which triggers grid search). If $\alpha = 0$, H2O solves the GLM with **ridge regression**, while if $\alpha = 1$ it does so with the **lasso** penalty.
- λ controls the penalty strength; its range is 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 for which the solution is the empty model (all zeros except for the intercept term).

The combination of the ℓ_1 and ℓ_2 penalties is beneficial, since the ℓ_1 induces sparsity while the ℓ_2 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 when focusing on sparsity involves using the ℓ_1 mainly with very little ℓ_2 penalty (α almost 1), to stabilize the computation and improve convergence speed.

4.6 GLM model families

The following subsection describes the GLM families supported in H2O.

4.6.1 Linear regression (Gaussian family)

Linear regression corresponds to the gaussian family model: the link function g is just the identity, and the density f corresponds to a normal distribution. It is the simplest example of a GLM, but has many uses and several advantages over the other families. For instance it is faster, and requires more stable computations. It models the dependency between a response y and a covariates vector x as a linear function:

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

The model is fitted by solving the least squares problem, which is equivalent to maximizing the likelihood for the gaussian family:

$$\max_{\beta,\beta_0} \frac{1}{2N} \sum_{i=1}^{N} (x_i^\top \beta + \beta_0 - y_i)^2 - \lambda \left(\alpha \|\beta\|_1 + \frac{1}{2}(1-\alpha)\|\beta\|_2^2 \right).$$

The deviance is simply the sum of squared prediction errors:

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

Example in R

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

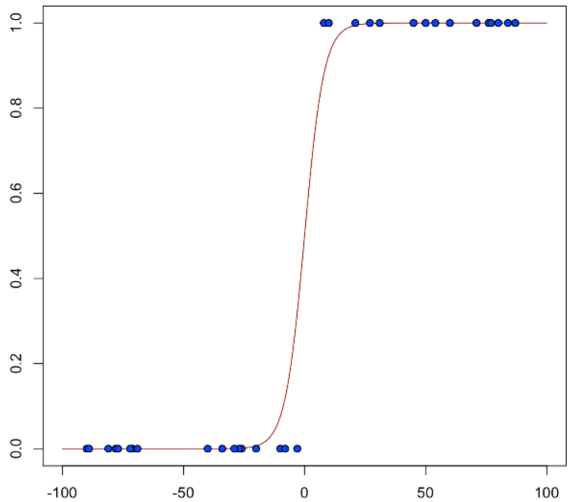
```
library(h2o)
h2o.init()
path = system.file("extdata", "prostate.csv", package = "h2o")
h2o_df = h2o.importFile(path)
gaussian.fit = h2o.glm(y = "VOL", x = c("AGE", "RACE", "PSA", "
   GLEASON"), training_frame = h2o_df, family = "gaussian")
```

Example in Python

```
import h2o
h2o.init()
path = h2o.system_file("prostate.csv")
h2o_df = h2o.import_file(path)
gaussian_fit = h2o.glm(y = "VOL", x = ["AGE", "RACE", "PSA", "
   GLEASON"], training_frame = h2o_df, family = "gaussian")
```

4.6.2 Logistic regression (Binomial family)

Logistic regression is used for binary classification problems, where the response is a categorical variable with two levels. It models the probability of an observation belonging to an output category given the data, for instance $Pr(y = 1|x)$. The canonical link for the binomial family is the logit function (or log odds). Its inverse is the logistic function that takes any real number and projects it onto the $[0, 1]$ range as desired to model the probability of belonging to a class. The corresponding s-curve (or sigmoid function) is shown below,



and the fitted model has the form

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

This can alternatively be written as

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

The model is fitted by maximizing the corresponding penalized likelihood

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

The corresponding deviance is equal to

$$D = -2\sum_{i=1}^{n}\left(y_i\log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right).$$

Example in R

Using the prostate data set, this example builds a binomial model that classifies the incidence of penetration of the prostatic capsule (CAPSULE). Confirm the entries in the CAPSULE column are binary using the `h2o.table()` function. Change the regression by changing the family to binomial.

```
library(h2o)
h2o.init()
path = system.file("extdata", "prostate.csv", package = "h2o")
h2o_df = h2o.importFile(path)
is.factor(h2o_df$CAPSULE)
h2o_df$CAPSULE = as.factor(h2o_df$CAPSULE)
is.factor(h2o_df$CAPSULE)
binomial.fit = h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA",
    "GLEASON"), training_frame = h2o_df, family = "binomial")
```

Example in Python

```
import h2o
h2o.init()
path = h2o.system_file("prostate.csv")
h2o_df = h2o.import_file(path)
h2o_df['CAPSULE'] = h2o_df['CAPSULE'].asfactor()
binomial_fit = h2o.glm(y = "CAPSULE", x = ["AGE", "RACE", "PSA",
   "GLEASON"], training_frame = h2o_df, family = "binomial")
```

4.6.3 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 in R

This example loads the Insurance data from the MASS library, imports it into H2O, and runs 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(h2o)
h2o.init()
library(MASS)
data(Insurance)

# Convert ordered factors into unordered factors.
# H2O only handles unordered factors today.
class(Insurance$Group) <- "factor"
class(Insurance$Age) <- "factor"

# Copy the R data.frame to an H2OFrame using as.h2o()
h2o_df = as.h2o(Insurance)
poisson.fit = h2o.glm(y = "Claims", x = c("District", "Group", "
   Age"), training_frame = h2o_df, family = "poisson")
```

Example in Python

```
# No equivalent running example provided for python due to the
# license of the MASS R package.
#
# poisson_fit = h2o.glm(y = "Claims", x = ["District", "Group", "
   Age"], training_frame = h2o_df, family = "poisson")
```

4.6.4 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, where 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 in R

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

```
library(h2o)
h2o.init()
path = system.file("extdata", "prostate.csv", package = "h2o")
h2o_df = h2o.importFile(path)
gamma.inverse <- h2o.glm(y = "DPROS", x = c("AGE","RACE","CAPSULE
   ","DCAPS","PSA","VOL"), training_frame = h2o_df, family = "
   gamma", link = "inverse")
gamma.log <- h2o.glm(y="DPROS", x = c("AGE","RACE","CAPSULE","
   DCAPS","PSA","VOL"), training_frame = h2o_df, family = "gamma",
    link = "log")
```

Example in Python

```
import h2o
h2o.init()
path = h2o.system_file("prostate.csv")
h2o_df = h2o.import_file(path)
gamma_inverse = h2o.glm(y = "DPROS", x = ["AGE","RACE","CAPSULE",
   "DCAPS","PSA","VOL"], training_frame = h2o_df, family = "gamma"
   , link = "inverse")
gamma_log = h2o.glm(y="DPROS", x = ["AGE","RACE","CAPSULE","DCAPS
   ","PSA","VOL"], training_frame = h2o_df, family = "gamma", link
    = "log")
```

4.6.5 Tweedie models

Tweedie models are a class of distributiions which include Gamma, Normal, Poisson and their combination. It is especially usefull for modelling of variables which have 0s combined with continuous non-zero values (e.g. Gamma distribution).

Tweedie is parametrized by variance power p which can be one of the following:

- 0 Normal
- 1 Poisson
- (1, 2) Compound Poisson, non-negative with mass at zero
- 2 Gamma
- 3 Inverse-Gaussian
- $>$ 2 Stable, with support on the positive reals

TODO fit in likelihood

Deviance for variance p ($p! = 1$ and $p! = 2$) is :

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

Example in R

```
# TODO
```

Example in Python

```
# TODO
```

5 Building GLM Models in H2O

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

5.1 Classification vs. regression

GLM can produce two categories of models: classification (binary classification only) and regression. Binary classification in GLM is also called logistic regression.

The data type of the response column determines the model category. If the response data type is a categorical (also called a factor or an enum) then a classification model is created. If the response column data type is numeric (either integer or real), then a regression model is created.

The following examples show how to coerce the data type of a column to a factor.

Example in R

```
library(h2o)
h2o.init()
path = system.file("extdata", "prostate.csv", package = "h2o")
h2o_df = h2o.importFile(path)
h2o_df$CAPSULE = as.factor(h2o_df$CAPSULE)
summary(h2o_df)
```

Example in Python

```
import h2o
h2o.init()
path = h2o.system_file("prostate.csv")
h2o_df = h2o.import_file(path)
h2o_df["CAPSULE"] = h2o_df["CAPSULE"].asfactor()
h2o_df.summary()
```

5.2 Training and validation frames

In this context, the word Frame refers to an H2OFrame, which is the fundamental method of data storage in H2O's distributed memory.

`training_frame` refers to a frame containing a training dataset. All predictors and the response (as well as offset and weights, if specified) must be part of this frame.

validation_frame refers to an optional frame containing a validation dataset. If specified, this frame must have exactly the same columns as the training dataset. Metrics are calculated on the validation dataset for convenience.

If a validation dataset is not specified during training, there are functions that can calculate the same metrics after the model training is complete.

5.3 Predictor and response variables

Every model must specify its predictors and response. Predictors and response are specified by the x and y parameters.

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

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

5.3.1 Categorical variables

If the response column is categorical, then a classification model is created. GLM only supports binomial classification, so the response column may only have two levels.

Categorical predictor columns may have more than two levels.

We recommend letting GLM handle categorical columns itself, as it can take advantage of the categorical column for better performance and memory utilization.

We strongly recommend avoidance of one-hot encoding categorical columns with many levels into many binary columns, as this is very inefficient. This is especially true for Python users who are used to expanding their categorical variables manually for other frameworks.

5.4 Family and 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, doing so can lead to an unstable computation.

5.5 Regularization parameters

Refer to Regularization above for a detailed explanation.

To get the best possible model, we need to find the optimal values of the regularization parameters α and λ. To find the optimal values, 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.

5.5.1 alpha and lambda

`alpha` controls the distribution between L1 (Lasso) and L2 (Ridge regression) penalties. An `alpha` of 1.0 indicates only Lasso, and an `alpha` of 0.0 indicates only ridge.

`lambda` controls the amount of regularization applied. For a `lambda` value of 0.0, no regularization is applied, and the `alpha` parameter is ignored. The default value for `lambda` is calculated by H2O using a heuristic based on the training data. If you let H2O calculate `lambda`, you can see the chosen value in the model output.

5.5.2 Lambda Search

Lambda search enables efficient and automatic search for optimal value of the `lambda` parameter. When lambda search is enabled, GLM will first fit a model with maximum regularization strength and then keep decreasing it until overfitting. The resulting model will be based on the best `lambda` value. When looking for sparse solution (`alpha` > 0), lambda serach can also be used to efficiently handle very wide datasets as it can filter out inactive predictors (noise) and only build models for a small subset of predictors. Common use of lambda search is to run on a dataset with many predictors but limit the number of active predictors to relatively small value.

Lambda search can be turned on by setting `lambda_search` and can be further customized by following arguments:

- `alpha` - regularization distribution between L1 and L2
- `validation_frame` andor `n_folds`: The best lambda is selected based on performance on the cross-validation or validation or training data. If available, cross-validation performance takes preference. If no validation data is available, the best lambda is selected based on training data performance and is therefore guaranteed to always be the minimal lambda computed as GLM can not overfit on a trainign dataset.

  Note: If running lambda search with validation dataset and no cross-validation, validation data set is used to select the lambda value, validation dataset is thus contaminated and performance should be evaluated on another independent dataset.

- `lambda_min_ratio` and `nlambdas`:

  The sequence of λ-s is automatically generated as an exponentially decreasing sequence, going from λ_{max}, the smallest λ such that the solution is a model with all 0s, to $\lambda_{min} =$ *lambda_min_ratio* * λ_{max}.

  H2O computes λ-models sequentially and in decreasing order, warm-starting (using the previous solution as the initial prediction) 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 two or three). 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` and `lambda_min_ratio` also specify the relative distance of any two lambdas in the sequence. This is important when applying recursive strong rules, which are only effective if the neighboring 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 0.912. For best results when using strong rules, keep the ratio close to the default.

- `max_active_predictors`: Limit the number of active predictors (the actual number of non-zero predictors in the model is going to be slightly lower), usefull when obtaining a sparse solution to avoid costly computation of models with too many predictors.

5.6 Handling wide data and building sparse models

H2O's GLM offers two different approaches to handling wide datasets: L1 regularization and the L_BFGS solver.

Quick recommendations:

- For a sparse solution (i.e. some predictors are dropped from the final model), use the IRLSM solver with `alpha` greater than 0.0 (so you get at least some L1 penalty) and enable `lambda_search`.
- For a dense solution (i.e. all predictors are in the final model), use the L_BFGS solver with `alpha` of 0 so there is no L1 penalty, only L2 penalty.
- If not sure whether the solution should be sparse or dense, try both cases and pick the better one based on validation (cross-validation) data.

5.6.1 Solver selection

The two solvers provided in H2O are Iterative Reweighted Least Squares Method (IRLSM) and Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L_BFGS). IRLSM uses a Gram Matrix approach underneath the hood, which is very efficient for tall and narrow datasets and when running lambda search with sparse solution. For wider and dense datasets (thousands of predictors and up), the L_BFGS solver scales better.

For performance reasons, we recommend not using any L1 penalty when using the L_BFGS solver, since model building can be slower. It is often better to run lambda search with IRLSM solver instead.

5.6.2 Stopping criteria

When using L1 penalty with lambda search, you can specify the `max_active_predictors` parameter to stop the search before it completes. Models built at the beginning of the lambda search have higher lambda values, consider fewer predictors, and take less time to calculate the model. Models built at the end of the lambda search have lower lambda values, incorporate more predictors, and

take a longer time to calculate the model. You can also set the `nlambdas` parameter for a lambda search to specify the number of models attempted across the search.

Example in R

```
library(h2o)
h2o.init()
h2o_df = h2o.importFile("http://s3.amazonaws.com/h2o-public-test-data/smalldata/airlines/allyears2k_headers.zip")
model = h2o.glm(y = "IsDepDelayed", x = c("Year", "Origin"), training_frame = h2o_df, family = "binomial", lambda_search = TRUE, max_active_predictors = 10)
print(model)
v1 = model@model$coefficients
v2 = v1[v1 > 0]
print(v2)
```

Example in Python

```
import h2o
h2o.init()
h2o_df = h2o.import_file("http://s3.amazonaws.com/h2o-public-test-data/smalldata/airlines/allyears2k_headers.zip")
model = h2o.glm(y = "IsDepDelayed", x = ["Year", "Origin"], training_frame = h2o_df, family = "binomial", lambda_search = True, max_active_predictors = 10)
print(model)
#v1 = model@model$coefficients
#v2 = v1[v1 > 0]
#print(v2)
```

5.7 Advanced features

H2O's GLM has several advanced features to help build better models.

5.7.1 Standardizing data

The `standardize` parameter, which is enabled by default, standardizes numeric columns to have zero mean and unit variance. This parameter must be enabled (using `standardize=TRUE`) to produce standardized coefficient magnitudes in the model output.

We recommend enabling standardization when using regularization (i.e. `lambda` is chosen by H2O or set by the user to be greater than 0). Only advanced users should disable standardization.

5.7.2 N-fold cross validation

All validation values can be computed using either the training data set (the default option) or using N-fold cross-validation (`nfolds` > 1). When N-fold cross-validation is enabled, H2O randomly splits data into n equally-sized parts, trains each of the n models on n-1 parts, and computes validation on the part that was not used for training.

You can also specify the rows that are assigned to each fold using the `fold_assignment` or `fold_column` parameters.

Example in R

```
library(h2o)
h2o.init()
path = system.file("extdata", "prostate.csv", package = "h2o")
h2o_df = h2o.importFile(path)
h2o_df$CAPSULE = as.factor(h2o_df$CAPSULE)
binomial.fit = h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA",
    "GLEASON"), training_frame = h2o_df, family = "binomial",
   nfolds = 5)
print(binomial.fit)
print(paste("training␣auc:␣␣␣␣␣␣␣␣", binomial.fit@model$training_
   metrics@metrics$AUC))
print(paste("cross-validation␣auc:", binomial.fit@model$cross_
   validation_metrics@metrics$AUC))
```

5.7.3 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, specifying only a few values is typically 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 very 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, we recommend running with $alpha = 1 - \epsilon$ instead.

Example in R

```
library(h2o)
h2o.init()
path = system.file("extdata", "prostate.csv", package = "h2o")
```

```
h2o_df = h2o.importFile(path)
h2o_df$CAPSULE = as.factor(h2o_df$CAPSULE)
alpha_opts = list(list(0), list(.25), list(.5), list(.75), list(1))
hyper_parameters = list(alpha = alpha_opts)
grid <- h2o.grid("glm", hyper_params = hyper_parameters,
                 y = "CAPSULE", x = c("AGE", "RACE", "PSA", "GLEASON"), training_frame = h2o_df, family = "binomial")
grid_models <- lapply(grid@model_ids, function(model_id) { model = h2o.getModel(model_id) })
for (i in 1:length(grid_models)) {
    print(sprintf("regularization: %-50s  auc: %f", grid_models[[i]]@model$model_summary$regularization, h2o.auc(grid_models[[i]])))
}
```

5.7.4 Grid search over lambda

Grid search over lambda is a feature from H2O-2 that is not currently supported in H2O-3. Instead, use the built-in lambda search capability in GLM.

5.7.5 Offsets

`offset_column` is an optional column name or index referring to a column in the training frame. This column specifies a prior known component to be included in the linear predictor during training. Offsets are per-row bias values that are used during model training. For Gaussian distributions, they can be seen as simple corrections to the response (y) column. Instead of learning to predict the response (y-row), the model learns to predict the (row) offset of the response column. For other distributions, the offset corrections are applied in the linearized space before applying the inverse link function to get the actual response values.

5.7.6 Row weights

`weights_column` is an optional column name or index referring to a column in the training frame. This column specifies on a per-row basis the weight of that row. If no weight column is specified, a default value of 1 is used for each row. Weights are per-row observation weights. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor.

5.7.7 Coefficient constraints

Coefficient constraints allow you to set special conditions over the model coefficients. Currently supported constraints are upper and lower bounds and the proximal operator interface, as described in Proximal Algorithms by Boyd et. al.

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

5.7.8 Proximal operators

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

6 GLM Model Output

The following sections explain the output produced by logistic regression (i.e. binomial classification).

Example in R

```
library(h2o)
h2o.init()
path = system.file("extdata", "prostate.csv", package = "h2o")
h2o_df = h2o.importFile(path)
h2o_df$CAPSULE = as.factor(h2o_df$CAPSULE)
rand_vec <- h2o.runif(h2o_df, seed = 1234)
train <- h2o_df[rand_vec <= 0.8,]
valid <- h2o_df[rand_vec > 0.8,]
binomial.fit = h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA",
    "GLEASON"), training_frame = train, validation_frame = valid,
   family = "binomial")
print(binomial.fit)
```

```
Model Details:
==============

H2OBinomialModel: glm
Model ID:  GLM_model_R_1439511782434_25
GLM Model:
    family  link                                regularization
       number_of_predictors_total number_of_active_predictors
       number_of_iterations training_frame
1 binomial logit Elastic Net (alpha = 0.5, lambda = 4.674E-4 )
                           4                              5
                     5      subset_39

Coefficients:
      names coefficients standardized_coefficients
1 Intercept    -6.467393                 -0.414440
2       AGE    -0.021983                 -0.143745
3      RACE    -0.295770                 -0.093423
4       PSA     0.028551                  0.604644
5   GLEASON     1.156808                  1.298815

H2OBinomialMetrics: glm
** Reported on training data. **

MSE:  0.1735008
R^2:  0.2842015
LogLoss:  0.5151585
AUC:  0.806806
Gini:  0.6136121
Null Deviance:  403.9953
Residual Deviance:  307.0345
AIC:  317.0345

Confusion Matrix for F1-optimal threshold:
         0   1    Error     Rate
0      125  50 0.285714  =50/175
1       24  99 0.195122  =24/123
Totals 149 149 0.248322  =74/298

Maximum Metrics:
                     metric threshold    value idx
1                    max f1  0.301518 0.727941 147
2                    max f2  0.203412 0.809328 235
3               max f0point5  0.549771 0.712831  91
4               max accuracy  0.301518 0.751678 147
```

```
5              max precision  0.997990 1.000000   0
6           max absolute_MCC  0.301518 0.511199 147
7 max min_per_class_accuracy  0.415346 0.739837 134

H2OBinomialMetrics: glm
** Reported on validation data. **

MSE:  0.1981162
R^2:  0.1460683
LogLoss:  0.5831277
AUC:  0.7339744
Gini:  0.4679487
Null Deviance:  108.4545
Residual Deviance:  95.63294
AIC:  105.6329

Confusion Matrix for F1-optimal threshold:
        0  1    Error    Rate
0      35 17 0.326923  =17/52
1       8 22 0.266667   =8/30
Totals 43 39 0.304878  =25/82

Maximum Metrics:
                      metric threshold    value idx
1                     max f1  0.469237 0.637681  38
2                     max f2  0.203366 0.788043  63
3               max f0point5  0.527267 0.616438  28
4               max accuracy  0.593421 0.719512  18
5              max precision  0.949357 1.000000   0
6           max absolute_MCC  0.469237 0.391977  38
7 max min_per_class_accuracy  0.482906 0.692308  36
```

6.1 Coefficients and normalized coefficients

Coefficients are the predictor weights (i.e. the actual model used for prediction). Coefficients should be used to make predictions for new data points:

```
binomial.fit@model$coefficients
```

```
  Intercept          AGE         RACE          PSA      GLEASON
-6.46739299 -0.02198278 -0.29576986  0.02855057  1.15680761
```

If the `standardize` option is enabled, H2O returns another set of coefficients: the standardized coefficients. These are the predictor weights of the standardized data and are included only for

informational purposes (e.g. to compare relative variable importance). In this case, the "normal" coefficients are obtained from the standardized coefficients by **reversing** the data standardization process (de-scaled, with the intercept adjusted by an 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.

Standardized coefficients are useful for comparing the relative contribution of different predictors to the model:

```
binomial.fit@model$coefficients_table
```

```
Coefficients:
      names coefficients standardized_coefficients
1 Intercept    -6.467393                 -0.414440
2       AGE    -0.021983                 -0.143745
3      RACE    -0.295770                 -0.093423
4       PSA     0.028551                  0.604644
5   GLEASON     1.156808                  1.298815
```

This view provides a sorted list of standardized coefficients in descending order for easy comparison:

```
binomial.fit@model$standardized_coefficient_magnitudes
```

```
Standardized Coefficient Magnitudes:
          names coefficients sign
GLEASON GLEASON     1.298815  POS
PSA         PSA     0.604644  POS
AGE         AGE     0.143745  NEG
RACE       RACE     0.093423  NEG
```

6.2 Model statistics

Various model statistics are available:

`MSE` is the mean squared error: $MSE = \frac{1}{N}\sum_{i=1}^{N}(actual_i - prediction_i)^2$

`R^2` is the R squared: $R^2 = 1 - \frac{MSE}{\sigma_y^2}$

`LogLoss` is the log loss. $LogLoss = \frac{1}{N}\sum_i^N \sum_j^C y_i log(p_{i,j})$

`AUC` is available only for binomial models and is defined the area under ROC curve.

`Gini` TODO

`Null deviance` Deviance (defined by selected family) computed for the null model.

`Residual deviance` Deviance of the built model

`AIC` is based on log-likelihood, which is summed up similarly to deviance

Fetch these statistics using the following accessor functions:

```
h2o.num_iterations(binomial.fit)
h2o.null_dof(binomial.fit, train = TRUE, valid = TRUE)
h2o.residual_dof(binomial.fit, train = TRUE, valid = TRUE)

h2o.mse(binomial.fit, train = TRUE, valid = TRUE)
h2o.r2(binomial.fit, train = TRUE, valid = TRUE)
h2o.logloss(binomial.fit, train = TRUE, valid = TRUE)
h2o.auc(binomial.fit, train = TRUE, valid = TRUE)
h2o.giniCoef(binomial.fit, train = TRUE, valid = TRUE)
h2o.null_deviance(binomial.fit, train = TRUE, valid = TRUE)
h2o.residual_deviance(binomial.fit, train = TRUE, valid = TRUE)
h2o.aic(binomial.fit, train = TRUE, valid = TRUE)
```

6.3 Confusion matrix

Fetch the confusion matrix directly using the following accessor function:

```
h2o.confusionMatrix(binomial.fit, valid = FALSE)
h2o.confusionMatrix(binomial.fit, valid = TRUE)
```

6.4 Scoring history

The following output example represents a sample scoring history:

```
binomial.fit@model$scoring_history
```

```
Scoring History:
            timestamp   duration iteration log_likelihood
               objective
1 2015-08-13 19:05:17  0.000 sec         0      201.99764
   0.67784
2 2015-08-13 19:05:17  0.002 sec         1      158.46117
   0.53216
3 2015-08-13 19:05:17  0.003 sec         2      153.74404
   0.51658
4 2015-08-13 19:05:17  0.004 sec         3      153.51935
   0.51590
5 2015-08-13 19:05:17  0.005 sec         4      153.51723
   0.51590
6 2015-08-13 19:05:17  0.006 sec         5      153.51723
   0.51590
```

7 Making Predictions

Once you have built a model, you can use it to make predictions using two different approaches: the in-H2O batch scoring approach and the real-time NanoFast POJO approach.

7.1 Batch in-H2O predictions

Batch in-H2O predictions are made using a normal H2O cluster on a new H2OFrame. In addition to predictions, you can also get metrics like AUC if you include the response column in the new data. Refer to the following example for logistic regression (i.e. binomial classification).

Example in R

```
library(h2o)
h2o.init()
path = system.file("extdata", "prostate.csv", package = "h2o")
h2o_df = h2o.importFile(path)
h2o_df$CAPSULE = as.factor(h2o_df$CAPSULE)
rand_vec <- h2o.runif(h2o_df, seed = 1234)
train <- h2o_df[rand_vec <= 0.8,]
valid <- h2o_df[(rand_vec > 0.8) & (rand_vec <= 0.9),]
test <- h2o_df[rand_vec > 0.9,]
binomial.fit = h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA",
    "GLEASON"), training_frame = train, validation_frame = valid,
   family = "binomial")

# Make and export predictions.
pred = h2o.predict(binomial.fit, test)
h2o.exportFile(pred, "/tmp/pred.csv", force = TRUE)
# Or you can export the predictions to hdfs:
#   h2o.exportFile(pred, "hdfs://namenode/path/to/file.csv")

# Calculate metrics.
perf = h2o.performance(binomial.fit, test)
print(perf)
```

Here is an example of making predictions on new data:

```
# Remove the response column to simulate new data points arriving
    without the answer being known.
newdata = test
newdata$CAPSULE <- NULL
newpred = h2o.predict(binomial.fit, newdata)
head(newpred)
```

```
  predict        p0        p1
1       1 0.1676892 0.8323108
2       0 0.4824181 0.5175819
3       1 0.2000061 0.7999939
4       0 0.9242169 0.0757831
5       0 0.5044669 0.4955331
6       0 0.7272743 0.2727257
```

The three columns in the prediction file are the predicted class, the probability that the prediction is class 0, and the probability that the prediction is class 1. The predicted class is chosen based on the maximum-F1 threshold.

You can change the threshold manually, for example to 0.3, and recalculate the predict column like this:

```
newpred$predict = newpred$p1 > 0.3
head(newpred)
```

```
  predict        p0        p1
1       1 0.1676892 0.8323108
2       1 0.4824181 0.5175819
3       1 0.2000061 0.7999939
4       0 0.9242169 0.0757831
5       1 0.5044669 0.4955331
6       0 0.7272743 0.2727257
```

7.2 Low-latency predictions with the NanoFast scoring engine

For NanoFast scoring, H2O GLM models can be directly rendered as a Plain Old Java Object (POJO). These are very low latency and can easily be embedded in any Java environment (a customer-facing web application, a Storm bolt, or a Spark Streaming pipeline, for example).

The POJO does nothing but pure math, and has no dependencies on any other software packages (not even H2O itself), so it is easy to work with.

Directions for using the POJO in detail are beyond the scope of this document, but here is an example of how to get one and view it. You can also access the POJO from the Flow Web UI.

Example in R

```
library(h2o)
h2o.init()
path = system.file("extdata", "prostate.csv", package = "h2o")
h2o_df = h2o.importFile(path)
```

```
h2o_df$CAPSULE = as.factor(h2o_df$CAPSULE)
binomial.fit = h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA",
    "GLEASON"), training_frame = h2o_df, family = "binomial")
h2o.download_pojo(binomial.fit)
```

8 Best Practices

Here are a few rules of thumb to follow:

- The IRLSM solver works best on tall and skinny datasets
- If you have a wide dataset, use an L1 penalty to eliminate columns from the model
- If you have a wide dataset, try the L_BFGS solver
- When trying lambda search, specify `max_predictors` if it is taking too long. 90% of the time is spent on the larger models with the small lambdas, so specifying `max_predictors` helps a lot
- Retain a small L2 penalty (i.e. ridge regression) for numerical stability (i.e. dont use `alpha` 1.0, use 0.95 instead)
- Use symmetric nodes in your H2O cluster
- For the IRLSM solver, bigger nodes can help the ADMM / Cholesky decomposition run faster
- If you need to impute data, do so before running GLM

8.1 Verifying Model Results

To determine the accuracy of your model:

- Look for suspiciously different cross-validation results between folds

  Example in R

```
library(h2o)
h2o.init()
h2o_df = h2o.importFile("http://s3.amazonaws.com/h2o-public-
   test-data/smalldata/airlines/allyears2k_headers.zip")
model = h2o.glm(y = "IsDepDelayed", x = c("Year", "Origin"),
   training_frame = h2o_df, family = "binomial", nfolds = 5)
print(paste("full␣model␣training␣auc:", model@model$training_
   metrics@metrics$AUC))
print(paste("full␣model␣cv␣auc:", model@model$cross_validation
   _metrics@metrics$AUC))
for (i in 1:5) {
```

```
8     cv_model_name = model@model$cross_validation_models[[i]]$
          name
9     cv_model = h2o.getModel(cv_model_name)
10    print(paste("cv␣fold␣", i, "␣training␣auc:", cv_
          model@model$training_metrics@metrics$AUC, "␣validation␣
          auc:␣", cv_model@model$validation_metrics@metrics$AUC))
11 }
```

- Look for explained deviance ($1 - \frac{NullDev - ResDev}{NullDev}$)
  - Too close to 0: model doesnt predict well
  - Too close to 1: model predicts too well (one of your input columnss is probably a cheating column)
- For logistic regression (i.e. binomial classification) models, look for AUC
  - Too close to 0.5: model doesnt predict well
  - Too close to 1: model predicts too well
- Look at the number_of_iterations or scoring history to see if GLM stops early for a particular lambda that interests you (performing all the iterations probably means the solution is not good). This is controlled by the `max_iterations` parameter.
- Too many NAs in your training data is bad (GLM discards rows with NA values) You should always check degrees of freedom of the output model. Degress of freedom is number of observations used to train the model minus the size of the model. If this number is way smaller than expected, it is likely that too many rows have been excluded due to having missing values.
  - If you have few columns with many NAs, you might accidentally be losing all your rows and it's better to exclude them.
  - If you have many columns with small fraction of uniformly distributed missing values, every row is lkely going to have at least one missing value. In this case, NAs should be imputed (filled in, e.g. with mean) before modelling.

9 Implementation Details

The following sections discuss some of the implementation choices in H2O's GLM.

9.1 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.

9.1.1 Largest categorical speed optimization

Categoricals have special handling during GLM computation as well. When forming the gram matrix, we can take advantage of 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.

$$N = \sum_{c \in C}(\|c.domain\|) - \arg\max_{c \in C} \|c.domain\| + \|Nums\|$$

9.2 Speed and memory performance characteristics

This section discusses the CPU and memory cost of running GLM for each available solver.

9.2.1 IRLSM solver

The implementation is based on iterative re-weighted least squares with an ADMM inner solver (as described in Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers by Boyd et. al) to deal with the 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:

	CPU	Memory
Gram matrix (X^TX) (distributed)	$O(\frac{MN^2}{pn})$	O(training data) + O(gram matrix) $O(MN) + O(N^2pn)$
ADMM + Cholesky decomposition (single node)	$O(\frac{N^3}{p})$	$O(N^2)$

M Number of rows in the training data
N Number of predictors in the training data
p Number of CPUs per node
n Number of nodes in the cluster

If $M >> N$, the algorithm scales linearly both in the number of nodes and the number of CPUs per node. However, the 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 a computational cost issue). In many cases, H2O can work around these limitations due to its handling of categoricals and by employing strong rules to filter out inactive predictors.

9.2.2 L-BFGS solver

In each iteration, L-BFGS computes a gradient at the current vector of coefficients and then computes an updated vector of coefficients in an approximated Newton-method step. The cost of the coefficient update is k*N, where N is number of predictors and k is a constant, the cost of gradient computation is $\frac{M*N}{pn}$ where M is number of observations in the dataset and pnis the number of cpu cores in the cluster. Since k is a (small) constant, the runtime of L-BFGS is dominated by the gradient computation, which is fully paralellized and L-BFGS thus scales almost linearly.

9.3 Data corner cases

- What if: the training data has an NA? All rows conatining (any) missing value are omitted form the training set.
- What if: the test data has an NA? The prediction for that observation is NA.
- What if: a predictor column is categorical, you are making predictions on test data, and the predictor is a level that wasn't seen during training? The value is zero for all predictors associated with the particular categorical vairable.

- What if: the response column is categorical, you are making predictions on test data, and the response is a level that wasn't seen during training? Only binomial model is supported, any extra levels in test-response will trigger an error.

10 H2O Community Resources

For more information about H2O, visit the following open-source sites:

H2O Full Documentation:	`http://docs.h2o.ai`
H2O-3 Github Repository:	`https://github.com/h2oai/h2o-3`
Open-source discussion forum:	h2ostream@googlegroups.com `https://groups.google.com/d/forum/h2ostream`
Issue tracking:	`http://jira@h2o.ai`

To contact us securely and privately, email us at support@h2o.ai.

11 Appendix: Parameters

- `x`: A vector containing the names of the predictors to use while building the GLM model. No default.
- `y`: A character string or index that represents the response variable in the model. No default.
- `training_frame`: An `H2OFrame` object containing the variables in the model.
- `model_id`: (Optional) The unique ID assigned to the generated model. If not specified, an ID is generated automatically.
- `validation_frame`: An `H2OParsedData` object containing the validation dataset used to construct confusion matrix. If blank, the training data is used by default.
- `max_iterations`: A non-negative integer specifying the maximum number of iterations.
- `beta_epsilon`: A non-negative number specifying the magnitude of the maximum difference between the coefficient estimates from successive iterations. Defines the convergence criterion.
- `solver`: A character string specifying the solver used: either `IRLSM`, which supports more features, or `L_BFGS`, which scales better for datasets with many columns.
- `standardize`: A logical value that indicates whether the numeric predictors should be standardized to have a mean of 0 and a variance of 1 prior to model training.
- `family`: A description of the error distribution and corresponding link function to be used in the model. The following options are supported: `gaussian`, `binomial`, `poisson`, `gamma`,

or `tweedie`. 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. The 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
- `tweedie_variance_power`: A numeric specifying the power for the variance function when `family = "tweedie"`.
- `tweedie_link_power`: A numeric specifying the power for the link function when `family = "tweedie"`.
- `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.
- `prior`: (Optional) A numeric specifying the prior probability of class 1 in the response when `family = "binomial"`. The default value is the observation frequency of class 1.
- `lambda`: A non-negative value representing 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). When the value is 0, no elastic-net penalty is applied and ordinary generalized linear models are fit. Default is 1e-05.
- `lambda_search`: A logical value indicating whether to conduct a search over the space of lambda values, starting from the max lambda, given lambda will be interpreted as the min. lambda. Default is false.
- `nlambdas`: The number of lambda values when `lambda_search = TRUE`. 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. If the number of observations is greater than the number of variables then `lambda_min_ratio = 0.0001`; if the number of observations is less than the number of variables then `lambda_min_ratio = 0.01`. Default is -1.0.
- `nfolds`: Number of folds for cross-validation. If `nfolds >=2`, then `validation_frame` must remain blank. Default is 0.
- `fold_column`: (Optional) Column with cross-validation fold index assignment per observation.
- `fold_assignment`: Cross-validation fold assignment scheme, if `fold_column` is not specified. The following options are supported: `AUTO, Random,` or `Modulo`.

- `keep_cross_validation_predictions`: Specify whether to keep the predictions of the cross-validation models.
- `beta_constraints`: A data frame or `H2OParsedData` object with the columns `["names", "lower_bounds", "upper_bounds", "beta_given"]`, where each row corresponds to a predictor in the GLM. `"names"` contains the predictor names, `"lower_bounds"/"upper_bounds"` are the lower and upper bounds (respectively) of the beta, and `"beta_given"` is a user-specified starting value.
- `offset_column`: Specify the offset column. Note: Offsets are per-row bias values that are used during model training. For Gaussian distributions, they can be seen as simple corrections to the response (y) column. Instead of learning to predict the response (y-row), the model learns to predict the (row) offset of the response column. For other distributions, the offset corrections are applied in the linearized space before applying the inverse link function to get the actual response values.
- `weights_column`: Specify the weights column. Note: Weights are per-row observation weights. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor.
- `intercept`: Logical; includes a constant term (intercept) in the model. If there are factor columns in your model, then the intercept must be included.

References

[1] Jerome Friedman, Trevor Hastie, and 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. 301-320

[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`