alpha

- Available in: GLM, GAM
- Hyperparameter: yes

Description

To get the best possible model, GLM and GAM need to find the optimal values of the regularization parameters α and λ. When performing regularization, penalties are introduced to the model buidling process to avoid overfitting, to reduce variance of the prediction error, and to handle correlated predictors. The two most common penalized models are ridge regression and LASSO (least absolute shrinkage and selection operator). The elastic net combines both penalties. These types of penalties are described in greater detail in the Regularization section in GLM for more information.

The `alpha` parameter controls the distribution between the ℓ_1 (LASSO) and ℓ_2 (ridge regression) penalties. The penalty is defined as

$$P(\alpha, \beta) = (1-\alpha)/2||\beta||_2^2 + \alpha||\beta||_1 = \sum_j[(1-\alpha)/2\beta_j^2 + \alpha|\beta_j|]$$

Given the above, a value of 1.0 represents LASSO, and a value of 0.0 produces ridge regression. This value defaults to 0 if `solver=L_BFGS`; otherwise, this value defaults to 0.5.

This option also works closely with the lambda parameter, which controls the amount of regularization applied. The following table describes the type of penalized model that results based on the values specifed for the `lambda` and `alpha` options.

`lambda` value	`alpha` value	Result
`lambda` == 0	`alpha` = any value	No regularization. `alpha` is ignored.
`lambda` > 0	`alpha` == 0	Ridge Regression
`lambda` > 0	`alpha` == 1	LASSO
`lambda` > 0	0 < `alpha` < 1	Elastic Net Penalty

Related Parameters

- lambda
- solver

Example

R | Python

```r
library(h2o)
h2o.init()

# import the boston dataset:
# this dataset looks at features of the boston suburbs and predicts median housing prices
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing
boston <- h2o.importFile("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv")

# set the predictor names and the response column name
predictors <- colnames(boston)[1:13]
# set the response column to "medv", the median value of owner-occupied homes in $1000's
response <- "medv"

# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds river; 0 otherwise))
boston["chas"] <- as.factor(boston["chas"])

# split into train and validation sets
boston_splits <- h2o.splitFrame(data =  boston, ratios = 0.8)
train <- boston_splits[[1]]
valid <- boston_splits[[2]]

# try using the `alpha` parameter:
# train your model, where you specify alpha
boston_glm <- h2o.glm(x = predictors, y = response, training_frame = train,
                      validation_frame = valid,
                      alpha = 0.25)

# print the mse for the validation data
print(h2o.mse(boston_glm, valid = TRUE))

# grid over `alpha`
# select the values for `alpha` to grid over
hyper_params <- list( alpha = c(0, 0.25, 0.5, 0.75, 0.1) )

# this example uses cartesian grid search because the search space is small
# and we want to see the performance of all models. For a larger search space use
# random grid search instead: {'strategy': "RandomDiscrete"}

# build grid search with previously selected hyperparameters
grid <- h2o.grid(x = predictors, y = response, training_frame = train, validation_frame = valid,
                 algorithm = "glm", grid_id = "boston_grid", hyper_params = hyper_params,
                 search_criteria = list(strategy = "Cartesian"))

# Sort the grid models by mse
sorted_grid <- h2o.getGrid("boston_grid", sort_by = "mse", decreasing = FALSE)
sorted_grid
```