h2o-tutorials / tutorials / deeplearning / deeplearning.R

angela0xdata PUBEV-3632 Fixed broken links to github repos d6e222f on Nov 1, 2016

3 contributors

502 lines (495 sloc) 33 KB


```
## Classification and Regression with H2O Deep Learning
#
#* Introduction
#  * Installation and Startup
#  * Decision Boundaries
#* Cover Type Dataset
#  * Exploratory Data Analysis
#  * Deep Learning Model
#  * Hyper-Parameter Search
#  * Checkpointing
#  * Cross-Validation
#  * Model Save & Load
#* Regression and Binary Classification
#* Deep Learning Tips & Tricks
#
### Introduction
#This tutorial shows how a H2O [Deep Learning](http://en.wikipedia.org/wiki/Deep_learning) model can be used to do supervised classificatio
#
#If run from plain R, execute R in the directory of this script. If run from RStudio, be sure to setwd() to the location of this script. h2
#
#More examples and explanations can be found in our [H2O Deep Learning booklet](http://h2o.ai/resources/) and on our [H2O Github Repository
#
#### H2O R Package
#
#Load the H2O R package:
#
## R installation instructions are at http://h2o.ai/download
library(h2o)
#
#### Start H2O
#Start up a 1-node H2O server on your local machine, and allow it to use all CPU cores and up to 2GB of memory:
#
h2o.init(nthreads=-1, max_mem_size="2G")
h2o.removeAll() ## clean slate - just in case the cluster was already running
#
#The `h2o.deeplearning` function fits H2O's Deep Learning models from within R.
#We can run the example from the man page using the `example` function, or run a longer demonstration from the `h2o` package using the `dem
#
args(h2o.deeplearning)
help(h2o.deeplearning)
example(h2o.deeplearning)
#demo(h2o.deeplearning)  #requires user interaction
#
#While H2O Deep Learning has many parameters, it was designed to be just as easy to use as the other supervised training methods in H2O. Ea
#
#### Let's have some fun first: Decision Boundaries
#We start with a small dataset representing red and black dots on a plane, arranged in the shape of two nested spirals. Then we task H2O's 
#
#We visualize the nature of H2O Deep Learning (DL), H2O's tree methods (GBM/DRF) and H2O's generalized linear modeling (GLM) by plotting th
```

```r
#
setwd("~/h2o-tutorials/tutorials/deeplearning") ##For RStudio
spiral <- h2o.importFile(path = normalizePath("../data/spiral.csv"))
grid   <- h2o.importFile(path = normalizePath("../data/grid.csv"))
# Define helper to plot contours
plotC <- function(name, model, data=spiral, g=grid) {
  data <- as.data.frame(data) #get data from into R
  pred <- as.data.frame(h2o.predict(model, g))
  n=0.5*(sqrt(nrow(g))-1); d <- 1.5; h <- d*(-n:n)/n
  plot(data[,-3],pch=19,col=data[,3],cex=0.5,
       xlim=c(-d,d),ylim=c(-d,d),main=name)
  contour(h,h,z=array(ifelse(pred[,1]=="Red",0,1),
          dim=c(2*n+1,2*n+1)),col="blue",lwd=2,add=T)
}
#
#We build a few different models:
#
#dev.new(noRStudioGD=FALSE) #direct plotting output to a new window
par(mfrow=c(2,2)) #set up the canvas for 2x2 plots
plotC( "DL", h2o.deeplearning(1:2,3,spiral,epochs=1e3))
plotC("GBM", h2o.gbm         (1:2,3,spiral))
plotC("DRF", h2o.randomForest(1:2,3,spiral))
plotC("GLM", h2o.glm         (1:2,3,spiral,family="binomial"))
#
#Let's investigate some more Deep Learning models. First, we explore the evolution over training time (number of passes over the data), and
#
#dev.new(noRStudioGD=FALSE) #direct plotting output to a new window
par(mfrow=c(2,2)) #set up the canvas for 2x2 plots
ep <- c(1,250,500,750)
plotC(paste0("DL ",ep[1]," epochs"),
      h2o.deeplearning(1:2,3,spiral,epochs=ep[1],
                              model_id="dl_1"))
plotC(paste0("DL ",ep[2]," epochs"),
      h2o.deeplearning(1:2,3,spiral,epochs=ep[2],
            checkpoint="dl_1",model_id="dl_2"))
plotC(paste0("DL ",ep[3]," epochs"),
      h2o.deeplearning(1:2,3,spiral,epochs=ep[3],
            checkpoint="dl_2",model_id="dl_3"))
plotC(paste0("DL ",ep[4]," epochs"),
      h2o.deeplearning(1:2,3,spiral,epochs=ep[4],
            checkpoint="dl_3",model_id="dl_4"))
#
#You can see how the network learns the structure of the spirals with enough training time. We explore different network architectures next
#
#dev.new(noRStudioGD=FALSE) #direct plotting output to a new window
par(mfrow=c(2,2)) #set up the canvas for 2x2 plots
for (hidden in list(c(11,13,17,19),c(42,42,42),c(200,200),c(1000))) {
  plotC(paste0("DL hidden=",paste0(hidden, collapse="x")),
        h2o.deeplearning(1:2,3,spiral,hidden=hidden,epochs=500))
}
#
#It is clear that different configurations can achieve similar performance, and that tuning will be required for optimal performance. Next,
#
#dev.new(noRStudioGD=FALSE) #direct plotting output to a new window
par(mfrow=c(2,2)) #set up the canvas for 2x2 plots
for (act in c("Tanh","Maxout","Rectifier","RectifierWithDropout")) {
  plotC(paste0("DL ",act," activation"),
        h2o.deeplearning(1:2,3,spiral,
              activation=act,hidden=c(100,100),epochs=1000))
}
#
#Clearly, the dropout rate was too high or the number of epochs was too low for the last configuration, which often ends up performing the
#
#More information about the parameters can be found in the [H2O Deep Learning booklet](http://h2o.ai/resources/).
#
### Cover Type Dataset
#We important the full cover type dataset (581k rows, 13 columns, 10 numerical, 3 categorical).
```

```r
#We also split the data 3 ways: 60% for training, 20% for validation (hyper parameter tuning) and 20% for final testing.
#
df <- h2o.importFile(path = normalizePath("../data/covtype.full.csv"))
dim(df)
df
splits <- h2o.splitFrame(df, c(0.6,0.2), seed=1234)
train  <- h2o.assign(splits[[1]], "train.hex") # 60%
valid  <- h2o.assign(splits[[2]], "valid.hex") # 20%
test   <- h2o.assign(splits[[3]], "test.hex")  # 20%
#
#Here's a scalable way to do scatter plots via binning (works for categorical and numeric columns) to get more familiar with the dataset.
#
#dev.new(noRStudioGD=FALSE) #direct plotting output to a new window
par(mfrow=c(1,1)) # reset canvas
plot(h2o.tabulate(df, "Elevation",                       "Cover_Type"))
plot(h2o.tabulate(df, "Horizontal_Distance_To_Roadways", "Cover_Type"))
plot(h2o.tabulate(df, "Soil_Type",                       "Cover_Type"))
plot(h2o.tabulate(df, "Horizontal_Distance_To_Roadways", "Elevation" ))
#
#### First Run of H2O Deep Learning
#Let's run our first Deep Learning model on the covtype dataset.
#We want to predict the `Cover_Type` column, a categorical feature with 7 levels, and the Deep Learning model will be tasked to perform (mu
#
response <- "Cover_Type"
predictors <- setdiff(names(df), response)
predictors
#
#To keep it fast, we only run for one epoch (one pass over the training data).
#
m1 <- h2o.deeplearning(
  model_id="dl_model_first",
  training_frame=train,
  validation_frame=valid,   ## validation dataset: used for scoring and early stopping
  x=predictors,
  y=response,
  #activation="Rectifier",  ## default
  #hidden=c(200,200),       ## default: 2 hidden layers with 200 neurons each
  epochs=1,
  variable_importances=T    ## not enabled by default
)
summary(m1)
#
#Inspect the model in [Flow](http://localhost:54321/) for more information about model building etc. by issuing a cell with the content `ge
#
#### Variable Importances
#Variable importances for Neural Network models are notoriously difficult to compute, and there are many [pitfalls](ftp://ftp.sas.com/pub/n
#
head(as.data.frame(h2o.varimp(m1)))
#
#### Early Stopping
#Now we run another, smaller network, and we let it stop automatically once the misclassification rate converges (specifically, if the movi
#
m2 <- h2o.deeplearning(
  model_id="dl_model_faster",
  training_frame=train,
  validation_frame=valid,
  x=predictors,
  y=response,
  hidden=c(32,32,32),                  ## small network, runs faster
  epochs=1000000,                      ## hopefully converges earlier...
  score_validation_samples=10000,      ## sample the validation dataset (faster)
  stopping_rounds=2,
  stopping_metric="misclassification", ## could be "MSE","logloss","r2"
  stopping_tolerance=0.01
)
summary(m2)
plot(m2)
```

```r
#
#### Adaptive Learning Rate
#By default, H2O Deep Learning uses an adaptive learning rate ([ADADELTA](http://arxiv.org/pdf/1212.5701v1.pdf)) for its stochastic gradien
#
#If `adaptive_rate` is disabled, several manual learning rate parameters become important: `rate`, `rate_annealing`, `rate_decay`, `momentu
#
#### Tuning
#With some tuning, it is possible to obtain less than 10% test set error rate in about one minute. Error rates of below 5% are possible wit
#
m3 <- h2o.deeplearning(
  model_id="dl_model_tuned",
  training_frame=train,
  validation_frame=valid,
  x=predictors,
  y=response,
  overwrite_with_best_model=F,    ## Return the final model after 10 epochs, even if not the best
  hidden=c(128,128,128),          ## more hidden layers -> more complex interactions
  epochs=10,                      ## to keep it short enough
  score_validation_samples=10000, ## downsample validation set for faster scoring
  score_duty_cycle=0.025,         ## don't score more than 2.5% of the wall time
  adaptive_rate=F,                ## manually tuned learning rate
  rate=0.01,
  rate_annealing=2e-6,
  momentum_start=0.2,             ## manually tuned momentum
  momentum_stable=0.4,
  momentum_ramp=1e7,
  l1=1e-5,                        ## add some L1/L2 regularization
  l2=1e-5,
  max_w2=10                       ## helps stability for Rectifier
)
summary(m3)
#
#Let's compare the training error with the validation and test set errors
#
h2o.performance(m3, train=T)          ## sampled training data (from model building)
h2o.performance(m3, valid=T)          ## sampled validation data (from model building)
h2o.performance(m3, newdata=train)    ## full training data
h2o.performance(m3, newdata=valid)    ## full validation data
h2o.performance(m3, newdata=test)     ## full test data
#
#To confirm that the reported confusion matrix on the validation set (here, the test set) was correct, we make a prediction on the test set
#
pred <- h2o.predict(m3, test)
pred
test$Accuracy <- pred$predict == test$Cover_Type
1-mean(test$Accuracy)
#
#### Hyper-parameter Tuning with Grid Search
#Since there are a lot of parameters that can impact model accuracy, hyper-parameter tuning is especially important for Deep Learning:
#
#For speed, we will only train on the first 10,000 rows of the training dataset:
#
sampled_train=train[1:10000,]
#
#The simplest hyperparameter search method is a brute-force scan of the full Cartesian product of all combinations specified by a grid sear
#
hyper_params <- list(
  hidden=list(c(32,32,32),c(64,64)),
  input_dropout_ratio=c(0,0.05),
  rate=c(0.01,0.02),
  rate_annealing=c(1e-8,1e-7,1e-6)
)
hyper_params
grid <- h2o.grid(
  algorithm="deeplearning",
  grid_id="dl_grid",
  training_frame=sampled_train,
```

```
  validation_frame=valid,
  x=predictors,
  y=response,
  epochs=10,
  stopping_metric="misclassification",
  stopping_tolerance=1e-2,        ## stop when misclassification does not improve by >=1% for 2 scoring events
  stopping_rounds=2,
  score_validation_samples=10000, ## downsample validation set for faster scoring
  score_duty_cycle=0.025,         ## don't score more than 2.5% of the wall time
  adaptive_rate=F,                ## manually tuned learning rate
  momentum_start=0.5,             ## manually tuned momentum
  momentum_stable=0.9,
  momentum_ramp=1e7,
  l1=1e-5,
  l2=1e-5,
  activation=c("Rectifier"),
  max_w2=10,                      ## can help improve stability for Rectifier
  hyper_params=hyper_params
)
grid
#
#Let's see which model had the lowest validation error:
#
grid <- h2o.getGrid("dl_grid",sort_by="err",decreasing=FALSE)
grid

## To see what other "sort_by" criteria are allowed
#grid <- h2o.getGrid("dl_grid",sort_by="wrong_thing",decreasing=FALSE)

## Sort by logloss
h2o.getGrid("dl_grid",sort_by="logloss",decreasing=FALSE)

## Find the best model and its full set of parameters
grid@summary_table[1,]
best_model <- h2o.getModel(grid@model_ids[[1]])
best_model

print(best_model@allparameters)
print(h2o.performance(best_model, valid=T))
print(h2o.logloss(best_model, valid=T))
#
#### Random Hyper-Parameter Search
#Often, hyper-parameter search for more than 4 parameters can be done more efficiently with random parameter search than with grid search.
#
hyper_params <- list(
  activation=c("Rectifier","Tanh","Maxout","RectifierWithDropout","TanhWithDropout","MaxoutWithDropout"),
  hidden=list(c(20,20),c(50,50),c(30,30,30),c(25,25,25,25)),
  input_dropout_ratio=c(0,0.05),
  l1=seq(0,1e-4,1e-6),
  l2=seq(0,1e-4,1e-6)
)
hyper_params

## Stop once the top 5 models are within 1% of each other (i.e., the windowed average varies less than 1%)
search_criteria = list(strategy = "RandomDiscrete", max_runtime_secs = 360, max_models = 100, seed=1234567, stopping_rounds=5, stopping_tol
dl_random_grid <- h2o.grid(
  algorithm="deeplearning",
  grid_id = "dl_grid_random",
  training_frame=sampled_train,
  validation_frame=valid,
  x=predictors,
  y=response,
  epochs=1,
  stopping_metric="logloss",
  stopping_tolerance=1e-2,        ## stop when logloss does not improve by >=1% for 2 scoring events
  stopping_rounds=2,
  score_validation_samples=10000, ## downsample validation set for faster scoring
```

```
  score_duty_cycle=0.025,         ## don't score more than 2.5% of the wall time
  max_w2=10,                      ## can help improve stability for Rectifier
  hyper_params = hyper_params,
  search_criteria = search_criteria
)
grid <- h2o.getGrid("dl_grid_random",sort_by="logloss",decreasing=FALSE)
grid

grid@summary_table[1,]
best_model <- h2o.getModel(grid@model_ids[[1]]) ## model with lowest logloss
best_model
#
#Let's look at the model with the lowest validation misclassification rate:
#
grid <- h2o.getGrid("dl_grid",sort_by="err",decreasing=FALSE)
best_model <- h2o.getModel(grid@model_ids[[1]]) ## model with lowest classification error (on validation, since it was available during tra
h2o.confusionMatrix(best_model,valid=T)
best_params <- best_model@allparameters
best_params$activation
best_params$hidden
best_params$input_dropout_ratio
best_params$l1
best_params$l2
#
####Checkpointing
#Let's continue training the manually tuned model from before, for 2 more epochs. Note that since many important parameters such as `epochs
#
max_epochs <- 12 ## Add two more epochs
m_cont <- h2o.deeplearning(
  model_id="dl_model_tuned_continued",
  checkpoint="dl_model_tuned",
  training_frame=train,
  validation_frame=valid,
  x=predictors,
  y=response,
  hidden=c(128,128,128),          ## more hidden layers -> more complex interactions
  epochs=max_epochs,              ## hopefully long enough to converge (otherwise restart again)
  stopping_metric="logloss",      ## logloss is directly optimized by Deep Learning
  stopping_tolerance=1e-2,        ## stop when validation logloss does not improve by >=1% for 2 scoring events
  stopping_rounds=2,
  score_validation_samples=10000, ## downsample validation set for faster scoring
  score_duty_cycle=0.025,         ## don't score more than 2.5% of the wall time
  adaptive_rate=F,                ## manually tuned learning rate
  rate=0.01,
  rate_annealing=2e-6,
  momentum_start=0.2,             ## manually tuned momentum
  momentum_stable=0.4,
  momentum_ramp=1e7,
  l1=1e-5,                        ## add some L1/L2 regularization
  l2=1e-5,
  max_w2=10                       ## helps stability for Rectifier
)
summary(m_cont)
plot(m_cont)
#
#Once we are satisfied with the results, we can save the model to disk (on the cluster). In this example, we store the model in a directory
#
path <- h2o.saveModel(m_cont,
          path="./mybest_deeplearning_covtype_model", force=TRUE)
#
#It can be loaded later with the following command:
#
print(path)
m_loaded <- h2o.loadModel(path)
summary(m_loaded)
#
#This model is fully functional and can be inspected, restarted, or used to score a dataset, etc. Note that binary compatibility between H2
```

```
#
####Cross-Validation
#For N-fold cross-validation, specify `nfolds>1` instead of (or in addition to) a validation frame, and `N+1` models will be built: 1 model
#
dlmodel <- h2o.deeplearning(
  x=predictors,
  y=response,
  training_frame=train,
  hidden=c(10,10),
  epochs=1,
  nfolds=5,
  fold_assignment="Modulo" # can be "AUTO", "Modulo", "Random" or "Stratified"
  )
dlmodel
#
#N-fold cross-validation is especially useful with early stopping, as the main model will pick the ideal number of epochs from the converge
#
###Regression and Binary Classification
#Assume we want to turn the multi-class problem above into a binary classification problem. We create a binary response as follows:
#
train$bin_response <- ifelse(train[,response]=="class_1", 0, 1)
#
#Let's build a quick model and inspect the model:
#
dlmodel <- h2o.deeplearning(
  x=predictors,
  y="bin_response",
  training_frame=train,
  hidden=c(10,10),
  epochs=0.1
)
summary(dlmodel)
#
#Instead of a binary classification model, we find a regression model (`H2ORegressionModel`) that contains only 1 output neuron (instead of
#H2O Deep Learning supports regression for distributions other than `Gaussian` such as `Poisson`, `Gamma`, `Tweedie`, `Laplace`. It also su
#
#To perform classification, the response must first be turned into a categorical (factor) feature:
#
train$bin_response <- as.factor(train$bin_response) ##make categorical
dlmodel <- h2o.deeplearning(
  x=predictors,
  y="bin_response",
  training_frame=train,
  hidden=c(10,10),
  epochs=0.1
  #balance_classes=T    ## enable this for high class imbalance
)
summary(dlmodel) ## Now the model metrics contain AUC for binary classification
plot(h2o.performance(dlmodel)) ## display ROC curve
#
#Now the model performs (binary) classification, and has multiple (2) output neurons.
#
###Unsupervised Anomaly detection
#For instructions on how to build unsupervised models with H2O Deep Learning, we refer to our previous [Tutorial on Anomaly Detection with 
#[Unsupervised Pretraining with an AutoEncoder R code example](https://github.com/h2oai/h2o-3/blob/master/h2o-r/tests/testdir_algos/deeplea
#
#
###H2O Deep Learning Tips & Tricks
#
#####Performance Tuning
#The [Definitive H2O Deep Learning Performance Tuning](http://blog.h2o.ai/2015/08/deep-learning-performance-august/) blog post covers many 
#
#####Activation Functions
#While sigmoids have been used historically for neural networks, H2O Deep Learning implements `Tanh`, a scaled and shifted variant of the s
#
#####Generalization Techniques
#L1 and L2 penalties can be applied by specifying the `l1` and `l2` parameters. Intuition: L1 lets only strong weights survive (constant pu
```

```
#
#####Early stopping and optimizing for lowest validation error
#By default, Deep Learning training stops when the `stopping_metric` does not improve by at least `stopping_tolerance` (0.01 means 1% impro
#
#####Training Samples per MapReduce Iteration
#The parameter `train_samples_per_iteration` matters especially in multi-node operation. It controls the number of rows trained on for each
#
#####Categorical Data
#For categorical data, a feature with K factor levels is automatically one-hot encoded (horizontalized) into K-1 input neurons. Hence, the
#
#####Sparse Data
#If the input data is sparse (many zeros), then it might make sense to enable the `sparse` option. This will result in the input not being
#
#####Missing Values
#H2O Deep Learning automatically does mean imputation for missing values during training (leaving the input layer activation at 0 after sta
#
#####Loss functions, Distributions, Offsets, Observation Weights
#H2O Deep Learning supports advanced statistical features such as multiple loss functions, non-Gaussian distributions, per-row offsets and
#In addition to `Gaussian` distributions and `Squared` loss, H2O Deep Learning supports `Poisson`, `Gamma`, `Tweedie` and `Laplace` distrib
#
#We refer to our [H2O Deep Learning R test code examples](https://github.com/h2oai/h2o-3/tree/master/h2o-r/tests/testdir_algos/deeplearning
#
#####Exporting Weights and Biases
#The model parameters (weights connecting two adjacent layers and per-neuron bias terms) can be stored as H2O Frames (like a dataset) by en
#
iris_dl <- h2o.deeplearning(1:4,5,as.h2o(iris),
             export_weights_and_biases=T)
h2o.weights(iris_dl, matrix_id=1)
h2o.weights(iris_dl, matrix_id=2)
h2o.weights(iris_dl, matrix_id=3)
h2o.biases(iris_dl,  vector_id=1)
h2o.biases(iris_dl,  vector_id=2)
h2o.biases(iris_dl,  vector_id=3)
#plot weights connecting `Sepal.Length` to first hidden neurons
plot(as.data.frame(h2o.weights(iris_dl,  matrix_id=1))[,1])
#
#####Reproducibility
#Every run of DeepLearning results in different results since multithreading is done via [Hogwild!](http://www.eecs.berkeley.edu/~brecht/pa
#
#####Scoring on Training/Validation Sets During Training
#The training and/or validation set errors *can* be based on a subset of the training or validation data, depending on the values for `scor
#
#Note that the default value of `score_duty_cycle=0.1` limits the amount of time spent in scoring to 10%, so a large number of scoring samp
#
#Stratified sampling of the validation dataset can help with scoring on datasets with class imbalance.  Note that this option also requires
#
#### More information can be found in the [H2O Deep Learning booklet](http://h2o.ai/resources/), in our [H2O SlideShare Presentations](http
#
#### All done, shutdown H2O
h2o.shutdown(prompt=FALSE)
```