A Deeper Dive into Experiments with R

David Smith

1/26/2020

A Deeper Dive into Experiments with R

In this vignette, we will go into more details about experiments.

- Work with multiple experiments
- Control experiments with command line arguments
- Track multiple metrics
- Use caret to train models and deploy to Kubernetes

We will use the `accidents` data, which was saved to your shared data store in the vignette "Train and deploy your first model with Azure ML". We will also use the cluster `rcluster` created in that same vignette. Run that vignette first (at least through the section "Upload data to the datastore"), or the following commands will not work.

Load the data

```
library(azuremlsdk)
ws <- load_workspace_from_config()
ds <- get_default_datastore(ws)
target_path <- "accidentdata"

#download_from_datastore(ds, target_path=".",prefix="accidentdata")

## Find the compute target
cluster_name <- "rcluster"
compute_target <- get_compute(ws, cluster_name = cluster_name)
if(is.null(compute_target)) stop("Training cluster not found")
```

Try out several models

We have provided three different experiment files: `accident-glm.R`, `accident-knn.R`, `accident-glmnet.R`. Each uses the `caret` package to fit a predictive model to the accidents data (GLM, KNN and GLMNET respectively). Here are the parts of `accident-glm.R` that load the data and fit the model.

The script loads the data `accidents` from the shared datastore, and then creates a training partition `accident_trn`. This data is then passed to the caret `train` function to fit a generalized linear model (in this case, a logistic regression).

```r
### FROM FILE: accident-glm.R - do not run this code

## Caret GLM model on training set with 5-fold cross validation
accident_glm_mod <- train(
  form = dead ~ .,
  data = accident_trn,
  trControl = trainControl(method = "cv", number = 5),
  method = "glm",
  family = "binomial"
)
summary(accident_glm_mod)
```

The `summary` command at the end generates output, which you can view in the experiment logs saved to Azure ML.

We will make several training runs, and save and track the results in a new experiment, called `accident`.

```r
exp <- experiment(ws, "accident")
```

Now, train the GLM model by submitting the script `accident-glm.R` to the experiment.

```r
est <- estimator(source_directory=".",
                 entry_script = "accident-glm.R",
                 script_params = list("--data_folder" = ds$path(target_path)),
                 compute_target = compute_target)
run <- submit_experiment(exp, est)
```

As usual, we can track the progress of the run in Azure ML by clicking on the "Web View" link displayed by this widget.

```r
view_run_details(run)
```

Now let's submit scripts fitting K-nearest-neighbors and GLMnet models to the data. We'll submit them to the same experiment, which will allow us to track and compare the accuracy of each model. In each script, the accuracy statistic is tracked with this line of code:

```r
### DO NOT RUN: tracking code from accident-XXX.R scripts
log_metric_to_run("Accuracy",
                  calc_acc(actual = accident_tst$dead,
                           predicted = predict(accident_glmnet_mod, newdata = accident_tst))
)
log_metric_to_run("Method","GLMNET")
log_metric_to_run("TrainPCT",train.pct)
```

We also track the algorithm used with the "Method" metric. (It's not really a metric, but it's useful to track in the Experiment view.) We also track the percentage of data used for the training set (the remainder is used for the test set, where accuracy is calculated). By default it is set to 75%, and we'll see how to change that in the next session.

For now, submit expermiments for the KNN and GLMnet models:

```
est <- estimator(source_directory=".",
                 entry_script = "accident-knn.R",
                 script_params = list("--data_folder" = ds$path(target_path)),
                 compute_target = compute_target)
run <- submit_experiment(exp, est)

est <- estimator(source_directory=".",
                 entry_script = "accident-glmnet.R",
                 script_params = list("--data_folder" = ds$path(target_path)),
                 compute_target = compute_target)
run <- submit_experiment(exp, est)
```

At this point, it's worth visiting `ml.azure.com` and taking a look at your training cluster: click on "Compute", then "Training Clusters", then "rcluster". Note that the experiments are queued until a node in the cluster is available, and the cluster may even spin up a new node if the maximum nodes setting allows.

Also take a look at your experiment, by clicking on "Experiments" and then "accident". You will see all of the runs you have submitted so far, along with their status (Completed or Running) and the Accuracy statistic you tracked. It's likely that the GLM experiment has the highest accuracy, but this depends on the random split between the training and test partitions.

Speaking of the training percentage, the experiment scripts have been designed to accept a command-line argument to specify the proportion used for the training set. The code, which makes use of the `optparse` package, looks like this:

```
## DO NOT RUN: options code from experiment script
options <- list(
  make_option(c("-d", "--data_folder")),
  make_option(c("-p", "--percent_train"))
)

opt_parser <- OptionParser(option_list = options)
opt <- parse_args(opt_parser)

train.pct <- as.numeric(opt$percent_train)
```

We can use this option to pass in a value for the training percentage when we submit the experiment. Let's submit three more experiments, setting the training percentage to 80%:

```
train_pct_exp <- 0.80

## GLM model
est <- estimator(source_directory = ".",
                 entry_script = "accident-glm.R",
                 script_params = list("--data_folder" = ds$path(target_path),
                                      "--percent_train" = train_pct_exp),
                 compute_target = compute_target
)
run.glm <- submit_experiment(exp, est)

## KNN model
exp <- experiment(ws, "accident")
est <- estimator(source_directory = ".",
                 entry_script = "accident-knn.R",
                 script_params = list("--data_folder" = ds$path(target_path),
                                      "--percent_train" = train_pct_exp),
                 compute_target = compute_target
)
run.knn <- submit_experiment(exp, est)

## GLMNET model
exp <- experiment(ws, "accident")
est <- estimator(source_directory = ".",
                 entry_script = "accident-glmnet.R",
                 script_params = list("--data_folder" = ds$path(target_path),
                                      "--percent_train" = train_pct_exp),
                 compute_target = compute_target
)
run.glmnet <- submit_experiment(exp, est)
```

We can check the accuracy for our runs at `ml.azure.com`, or by querying the service directly:

```
get_run_metrics(run.glm)$Accuracy
get_run_metrics(run.knn)$Accuracy
get_run_metrics(run.glmnet)$Accuracy
```

Select a model for deployment

There's not a lot of differences between the accuracy of the models. We will deploy the GLMnet model, but you can deploy any of the models in the same way. The `caret` prediction interface is consistent across models, so you can use the same scoring script for all of them. We will deploy the model object along with the prediction script `accident_predict_caret.R`.

```
download_files_from_run(run.glmnet, prefix="outputs/")
accident_model <- readRDS("outputs/model.rds")

model <- register_model(ws,
                        model_path = "outputs/model.rds",
                        model_name = "accidents_model_caret",
                        description = "Predict probability of auto accident using caret")

r_env <- r_environment(name = "basic_env")

inference_config <- inference_config(
  entry_script = "accident_predict_caret.R",
  source_directory = ".",
  environment = r_env)
```

Deploy a model to Azure Container Instances

Now we cam deploy our model as a container, to Azure Container Instances.

```
aci_config <- aci_webservice_deployment_config(cpu_cores = 1, memory_gb = 0.5)

aci_service <- deploy_model(ws,
```

```r
                              'accident-pred-caret',
                              list(model),
                              inference_config,
                              aci_config)

wait_for_deployment(aci_service, show_output = TRUE)
```

If you wanted to deploy to Kubernetes, you would use something like this instead:

```r
## DO NOT RUN: sample code for Kubernetes deployment
aks_target <- create_aks_compute(ws,
                                   cluster_name = 'caretkluster',
                                   vm_size='Standard_D2_v2',
                                   agent_count=12)

wait_for_provisioning_completion(aks_target, show_output = TRUE)
```

Use the prediction service in a Shiny application

We have provided the server and UI for a shiny application in the `accident-app` folder. The app uses the global variable `accident-endpoint` to find the endpoint of the prediction service to call, so set it here:

```r
accident.endpoint <- get_webservice(ws,   "accident-predict-caret")$scoring_uri
```

You can run the app by opening `app.R` in the `accident-app` folder in RStudio and clicking "Run App", or by running the code below. Try out different values of the input variables to see how they affect the predicted probability of a fatal accident.

```r
shiny::runApp("accident-app")
```