Attempting Predictive Models for Average CPU Usage

Brian Kichler

May 18, 2019

Setting Up the Data

Logistic regression is an appropriate method to predict the probability of an event by defining a dichotomous/dummy variable. In this case, we want to see if we can develop a model that predicts whether a VM will be underutilized and a candidate for reduction in number of CPUs.

We can start by defining a new dummy variable, which will be on AvgCpuUsage in wide_df. We'll create the variable "BelowQ1AvgCpuUsage", which is defined as true (1) when an observation falls below the first quartile value for Average CPU Usage:

```
load("Itility - Global ENV.RData")
library(tidyverse)
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
## -- Attaching packages ---------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.1       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
wide_df_model <- wide_df %>%
  mutate(BelowQ1AvgCpuUsage = as.numeric(AvgCpuUsage < 168236689))
```

There will obviously be bias due to the definition of this variable. Here's what it looks like:

```
table(wide_df_model$BelowQ1AvgCpuUsage)
```

```
## 
##      0      1 
## 165220  55074
```

To mitigate this, this code creates a data frame called “trainingData,” which equally samples ones and zeroes with a set seed, then folds the leftover data into another data frame called “testData”:

```
# Training Data
input_ones <- wide_df_model[which(wide_df_model$BelowQ1AvgCpuUsage == 1), ]
input_zeros <- wide_df_model[which(wide_df_model$BelowQ1AvgCpuUsage == 0), ]
set.seed(100)
input_ones_training_rows <- sample(1:nrow(input_ones), 0.8*nrow(input_ones))
input_zeros_training_rows <- sample(1:nrow(input_zeros), 0.8*nrow(input_ones))
training_ones <- input_ones[input_ones_training_rows, ]
training_zeros <- input_zeros[input_zeros_training_rows, ]
trainingData <- rbind(training_ones, training_zeros)

# Create Test Data
test_ones <- input_ones[-input_ones_training_rows, ]
test_zeros <- input_zeros[-input_zeros_training_rows, ]
testData <- rbind(test_ones, test_zeros)
```

Next, the variables “Cluster” and “NumCpu” should be converted to factors, then we can calculate the IVs for the included variables in our wide_df_model data frame. “MemoryMB” and “CpuMHz” are treated as continuous. The “smbinning” package provides functions to calculate WOE by variable, from which we can grab the IV and render it in a table:

```
iv_df
```

```
##           VARS     IV
## 9  MinCpuUsage 1.0657
## 10 MaxCpuUsage 0.2802
## 5  AvgMemUsage 0.0916
## 6  MinMemUsage 0.0539
## 3       CpuMHz 0.0479
## 4       NumCpu 0.0427
## 2     MemoryMB 0.0342
## 7  MaxMemUsage 0.0133
## 1      Cluster 0.0061
## 8  AvgCpuUsage 0.0000
```

The only variables above 0.1 are min and max CPU usage. “MinCpuUsage” has a very high IV, and would naturally track with low average CPU usage.

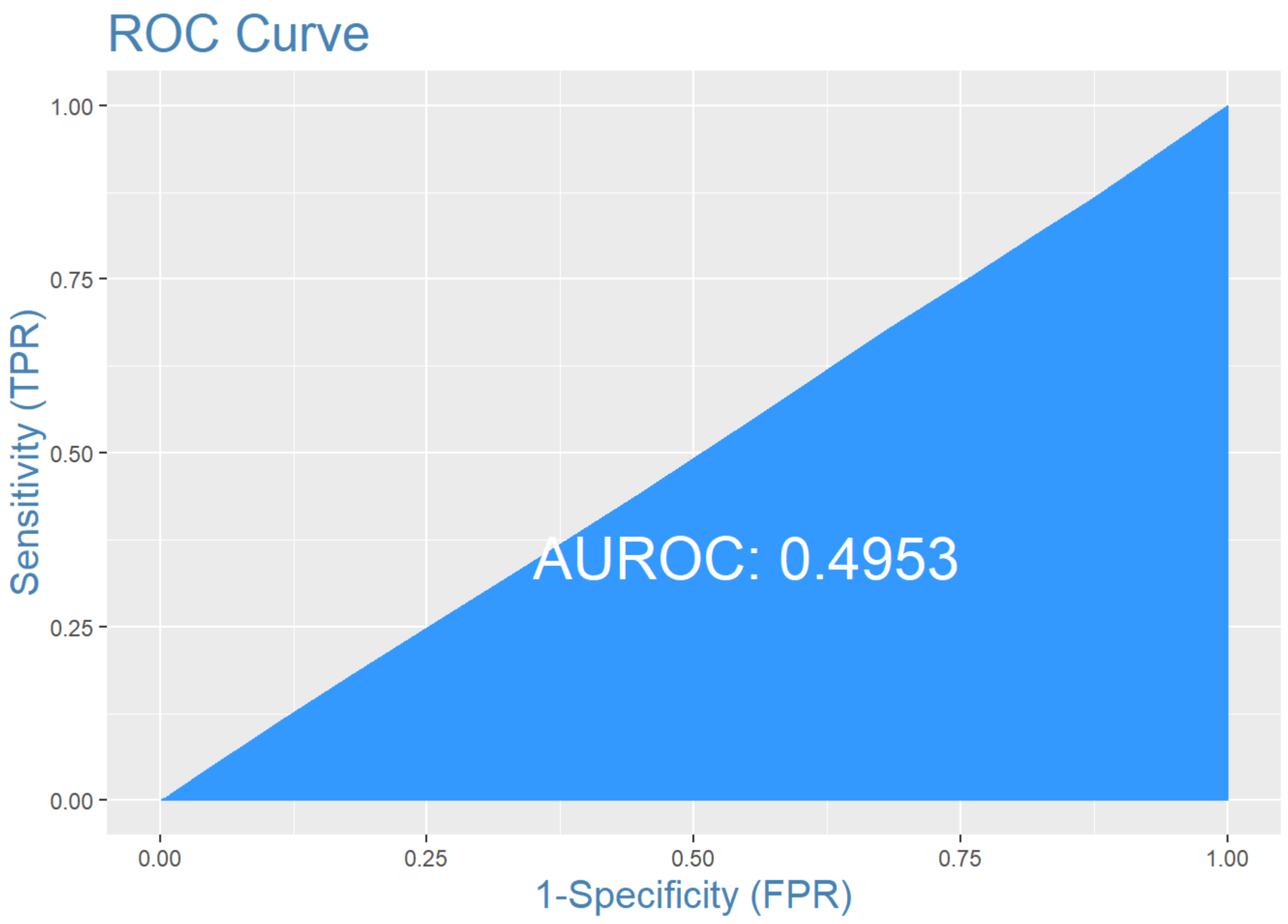
Building the Logit Model

Next step is to build the model and predicted average CPU usage:

```
logitMod <- glm(BelowQ1AvgCpuUsage ~ MinCpuUsage + MaxCpuUsage + AvgMemUsage + MinMemUsage + Cpu
MHz + NumCpu + MemoryMB + MaxMemUsage, data=trainingData, family=binomial(link="logit"))
predicted <- predict(logitMod, testData, type="response")
summary(logitMod)
```

```
##
## Call:
## glm(formula = BelowQ1AvgCpuUsage ~ MinCpuUsage + MaxCpuUsage +
##     AvgMemUsage + MinMemUsage + CpuMHz + NumCpu + MemoryMB +
##     MaxMemUsage, family = binomial(link = "logit"), data = trainingData)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1799  -1.1591  -0.8623   1.1688   1.5851
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.942e-02  2.644e-02  -1.869   0.0616 .
## MinCpuUsage  6.761e-10  2.612e-11  25.887  < 2e-16 ***
## MaxCpuUsage -5.978e-10  2.556e-11 -23.385  < 2e-16 ***
## AvgMemUsage -3.551e-11  2.410e-11  -1.474   0.1406
## MinMemUsage -3.461e-10  2.750e-11 -12.586  < 2e-16 ***
## CpuMHz       4.464e-05  7.432e-06   6.006 1.90e-09 ***
## NumCpu      -1.059e-01  1.763e-02  -6.008 1.88e-09 ***
## MemoryMB     2.003e-06  2.670e-07   7.499 6.41e-14 ***
## MaxMemUsage  2.027e-10  2.808e-11   7.221 5.18e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 122012  on 88012  degrees of freedom
## Residual deviance: 120022  on 88004  degrees of freedom
##   (105 observations deleted due to missingness)
## AIC: 120040
##
## Number of Fisher Scoring iterations: 4
```

The outcome is really poor, with very little predictive value. The true positive rate basically matches the false positive rate:



What about modeling summary data?

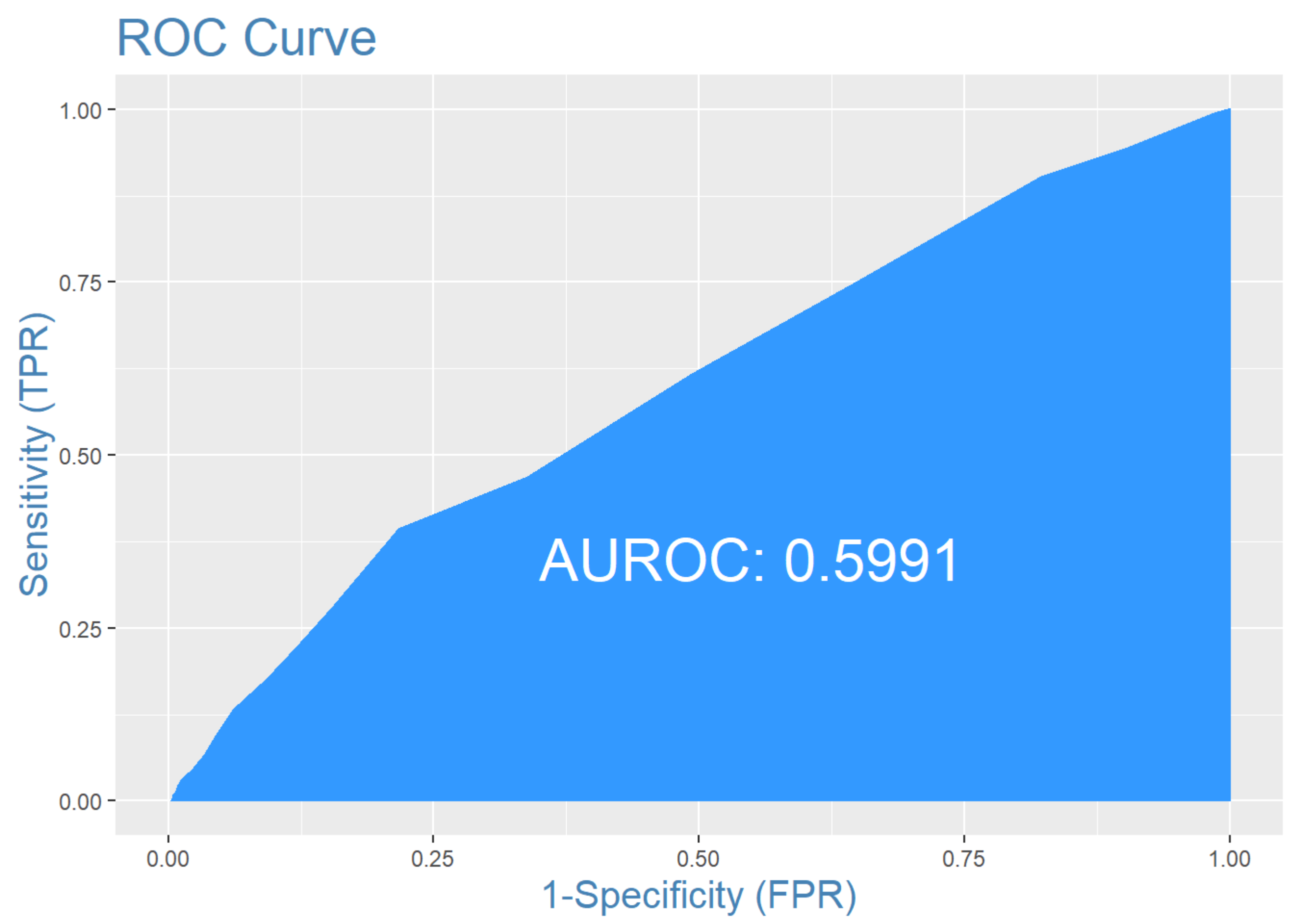
The number of observations is relatively small (n=4261), so this time we will run the analysis without training data. The results are slightly more favorable with summary data:

```
##
## Call:
## glm(formula = BelowQ1AvgCpuUsage ~ MedianCpuUsageRange + CpuMHz +
##     NumCpu + MedianMemUsageRange + Cluster + ObservationCount,
##     family = binomial(link = "logit"), data = summary_df_model)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2703  -0.7768  -0.7005   1.1028   2.0241
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -1.107e+00  3.080e-01  -3.593 0.000327 ***
## MedianCpuUsageRange -1.814e-10  1.123e-10  -1.615 0.106212
## CpuMHz               7.362e-05  3.998e-05   1.842 0.065547 .
## NumCpu2             -9.758e-02  2.634e-01  -0.370 0.711028
## NumCpu4             -9.457e-02  3.851e-01  -0.246 0.806021
## NumCpu6             -1.300e+01  3.771e+02  -0.034 0.972512
## NumCpu8             -2.349e-01  7.605e-01  -0.309 0.757397
## NumCpu10            -1.683e+00  9.655e-01  -1.743 0.081272 .
## NumCpu12            -1.494e+01  3.785e+02  -0.039 0.968506
## NumCpu16            -2.770e+00  1.501e+00  -1.846 0.064935 .
## NumCpu20            -3.198e+00  1.849e+00  -1.729 0.083798 .
## NumCpu32            -4.384e+00  3.070e+00  -1.428 0.153229
## MedianMemUsageRange  3.773e-10  1.198e-10   3.149 0.001640 **
## Cluster2            -3.442e-01  1.125e-01  -3.060 0.002215 **
## Cluster3            -4.230e-01  1.023e-01  -4.134 3.57e-05 ***
## Cluster4            -4.117e-01  1.176e-01  -3.501 0.000464 ***
## Cluster5            -4.033e-01  1.327e-01  -3.039 0.002376 **
## ObservationCount    -5.123e-03  3.436e-03  -1.491 0.136003
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4785.9  on 4250  degrees of freedom
## Residual deviance: 4674.1  on 4233  degrees of freedom
##   (10 observations deleted due to missingness)
## AIC: 4710.1
##
## Number of Fisher Scoring iterations: 12
```

```
## Waiting for profiling to be done...
##                             2.5 %        97.5 %
## (Intercept)         -1.725780e+00 -5.155724e-01
## MedianCpuUsageRange -4.017389e-10  3.863297e-11
## CpuMHz              -5.044892e-06  1.519452e-04
## NumCpu2             -5.990160e-01  4.371227e-01
## NumCpu4             -8.426070e-01  6.696051e-01
## NumCpu6                        NA  4.598299e+01
## NumCpu8             -1.725040e+00  1.260786e+00
## NumCpu10            -3.582511e+00  2.075034e-01
## NumCpu12                       NA  4.268941e+01
## NumCpu16            -5.732793e+00  1.586937e-01
## NumCpu20            -6.821358e+00  4.397360e-01
## NumCpu32            -1.040236e+01  1.651579e+00
## MedianMemUsageRange  1.422581e-10  6.121265e-10
## Cluster2            -5.654309e-01 -1.242838e-01
## Cluster3            -6.237152e-01 -2.225230e-01
## Cluster4            -6.435596e-01 -1.823360e-01
## Cluster5            -6.659441e-01 -1.453793e-01
## ObservationCount    -1.177178e-02  1.715473e-03
```

```
##                          GVIF Df GVIF^(1/(2*Df))
## MedianCpuUsageRange  1.048421  1        1.023924
## CpuMHz              81.178013  1        9.009884
## NumCpu              96.669149  9        1.289121
## MedianMemUsageRange  1.092356  1        1.045159
## Cluster              1.283913  4        1.031732
## ObservationCount     1.068625  1        1.033743
```

Here’s the ROC plot for summary data:

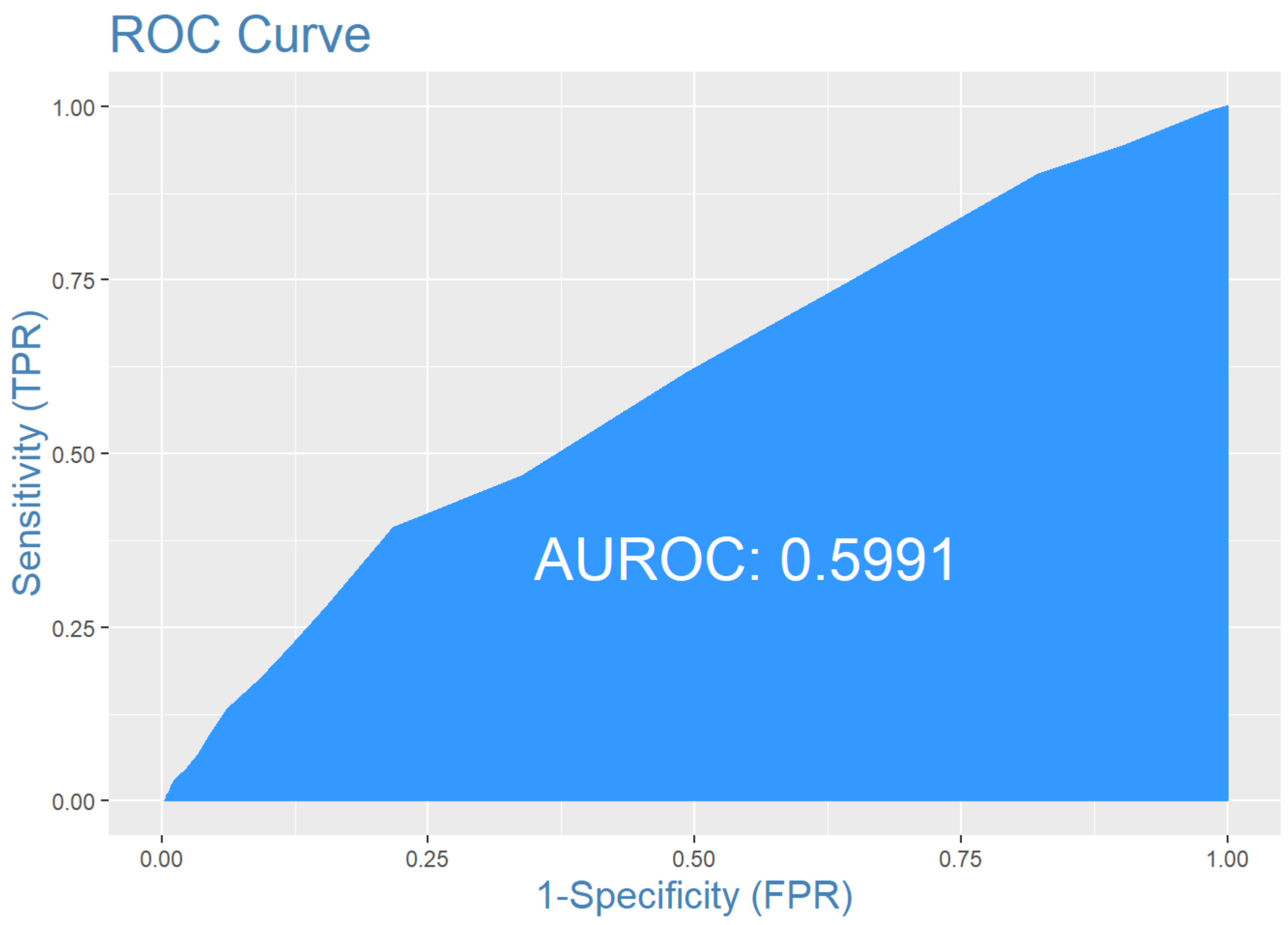


The predictive value of this model is still low. “MedianMemUsageRange” and “CpuMHz” both have weak positive estimates, meaning a 1-unit change in either of those variables slightly increases the odds of a VM being in the first quartile for average CPU usage. However, the p-value for CpuMHz is slightly above 0.05 at 0.0656, and more to the point, its high variance inflation factor points to collinearity. If we drop CpuMHz and re-run the analysis, the results look slightly better:

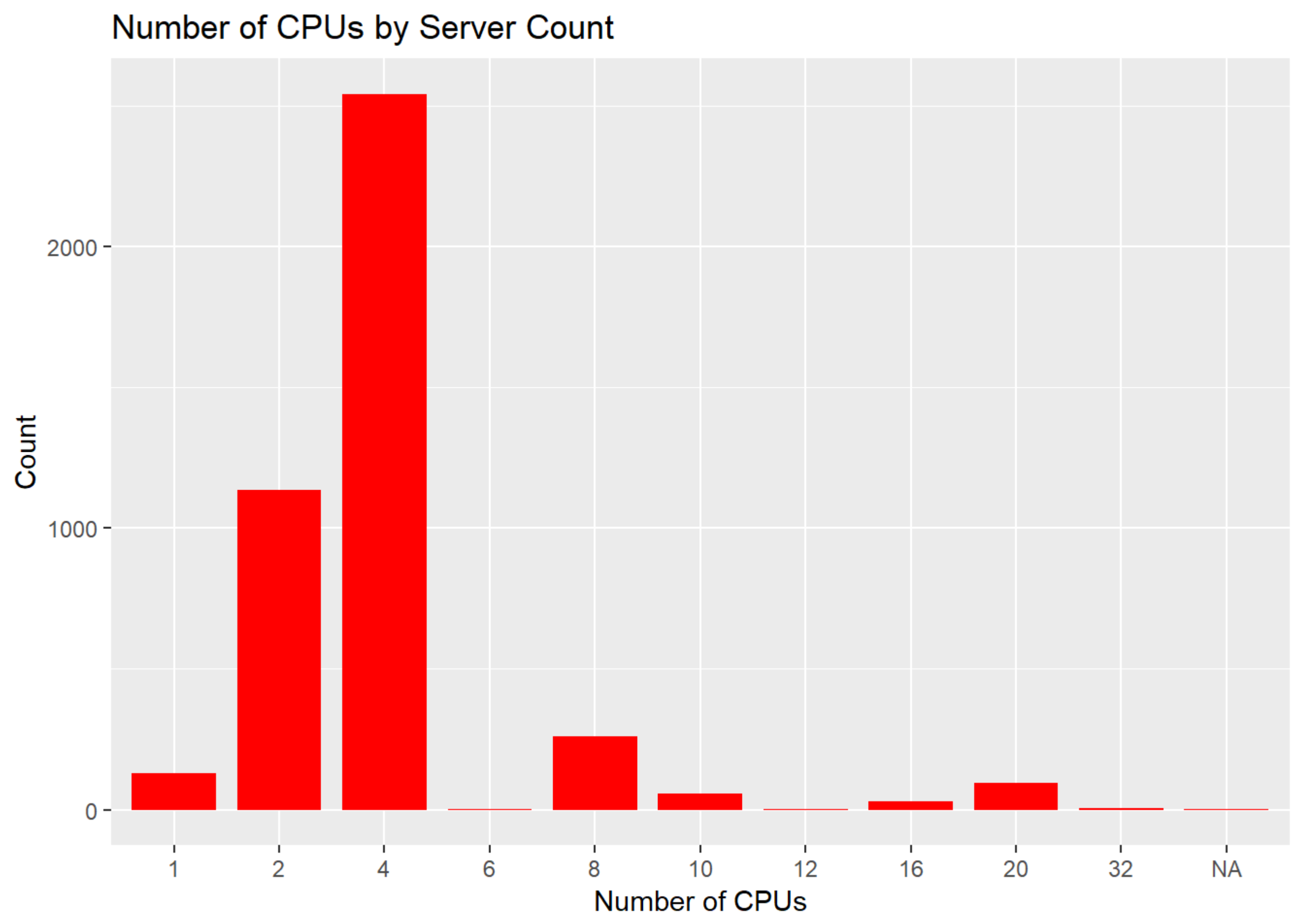
```
##
## Call:
## glm(formula = BelowQ1AvgCpuUsage ~ MedianCpuUsageRange + NumCpu +
##     MedianMemUsageRange + Cluster + ObservationCount, family = binomial(link = "logit"),
##     data = summary_df_model)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2635  -0.7749  -0.7027   0.5632   2.0208
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -9.653e-01  2.962e-01  -3.259 0.001117 **
## MedianCpuUsageRange -1.967e-10  1.120e-10  -1.756 0.079065 .
## NumCpu2              8.281e-02  2.443e-01   0.339 0.734663
## NumCpu4              4.586e-01  2.375e-01   1.931 0.053463 .
## NumCpu6             -1.206e+01  3.089e+02  -0.039 0.968850
## NumCpu8              1.075e+00  2.663e-01   4.035 5.47e-05 ***
## NumCpu10            -6.542e-02  3.964e-01  -0.165 0.868916
## NumCpu12            -1.213e+01  3.784e+02  -0.032 0.974427
## NumCpu16            -1.987e-01  5.192e-01  -0.383 0.701981
## NumCpu20             1.507e-01  3.368e-01   0.447 0.654583
## NumCpu32             1.077e+00  8.057e-01   1.337 0.181228
## MedianMemUsageRange  3.803e-10  1.197e-10   3.177 0.001486 **
## Cluster2            -3.060e-01  1.101e-01  -2.780 0.005441 **
## Cluster3            -4.259e-01  1.023e-01  -4.163 3.14e-05 ***
## Cluster4            -4.309e-01  1.175e-01  -3.668 0.000244 ***
## Cluster5            -4.061e-01  1.326e-01  -3.062 0.002199 **
## ObservationCount    -4.274e-03  3.429e-03  -1.246 0.212634
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4790.5  on 4258  degrees of freedom
## Residual deviance: 4681.5  on 4242  degrees of freedom
##   (2 observations deleted due to missingness)
## AIC: 4715.5
##
## Number of Fisher Scoring iterations: 12
```

```
##                         GVIF Df GVIF^(1/(2*Df))
## MedianCpuUsageRange 1.047308  1        1.023381
## NumCpu              1.292740  9        1.014367
## MedianMemUsageRange 1.092603  1        1.045277
## Cluster             1.196374  4        1.022665
## ObservationCount    1.062525  1        1.030789
```

```
## Waiting for profiling to be done...
##                          2.5 %        97.5 %
## (Intercept)         -1.562854e+00 -3.989527e-01
## MedianCpuUsageRange -4.163897e-10  2.275410e-11
## NumCpu2             -3.788476e-01  5.830032e-01
## NumCpu4              1.154178e-02  9.467052e-01
## NumCpu6                        NA  2.819941e+01
## NumCpu8              5.666121e-01  1.614326e+00
## NumCpu10            -8.627300e-01  7.012326e-01
## NumCpu12                       NA  4.723363e+01
## NumCpu16            -1.290883e+00  7.748607e-01
## NumCpu20            -5.107311e-01  8.148858e-01
## NumCpu32            -6.150132e-01  2.668222e+00
## MedianMemUsageRange  1.455366e-10  6.148575e-10
## Cluster2            -5.223981e-01 -9.072268e-02
## Cluster3            -6.266580e-01 -2.254790e-01
## Cluster4            -6.625472e-01 -2.017891e-01
## Cluster5            -6.685584e-01 -1.483518e-01
## ObservationCount    -1.090618e-02  2.552896e-03
```



Our collinearity problem is gone, and there are some interesting results based on number of CPU cores. Now we see the log odds of a VM falling into the first quartile increasing when the number of CPUs is 4 or 8. Here's what the distribution of CPUs by VM looks like:



```
## # A tibble: 11 x 2
##    NumCpu CpuCount
##    <fct>     <int>
##  1 1           130
##  2 2          1134
##  3 4          2542
##  4 6             3
##  5 8           260
##  6 10           58
##  7 12            2
##  8 16           29
##  9 20           94
## 10 32            7
## 11 <NA>          2
```

Concluding thoughts

- If there were far more observations per VM and consistent POSIX timestamps, it would be easier to train a machine learning model and better understand the underlying trends that are driving behavior
- There's more analysis that could be done on this data set, but as it stands now, we could advise the client based on the summary analysis at https://slides.com/brian-kichler/itility?token=cLnU1kTL (https://slides.com/brian-kichler/itility?token=cLnU1kTL) that shows ~ 6% of the multi-CPU VMs underutilized

- The last log regression yielded some more interesting results, although it’s still not a strong predictor for low CPU usage by VM. However, with relatively higher log odds increasing for 8-CPU VMs, we could advise the client to study reduction for these units first. The more impactful result if it holds is the weaker positive log odds increase for 4-CPU VMs, as these represent a high proportion (60%) of the overall count. It would be very useful to plot 4-core and 8-core over time for trend analysis to see if this holds up.