# Scott's Thesis Images

## Contents

### Figure 1

*Figure 1* shows the complex set of relationships between the data in the Neotoma Paleoecological Database. The development of a dedicated database to manage these relationshios implicates that the complexity of the data exceeded the ability of traditional data management techniques.

### Figure 2

**A.**

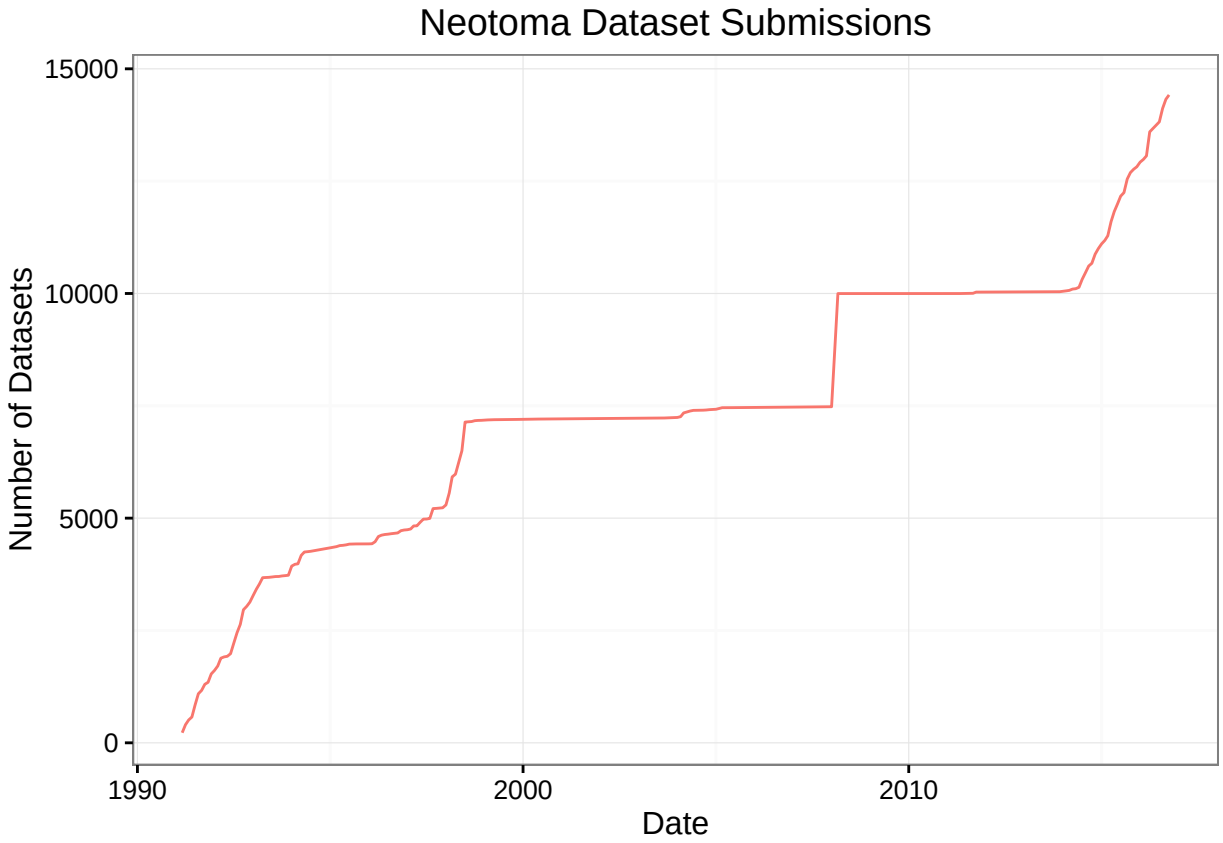



*Figure 2A* shows the steady increase in datasets in the Neotoma Paleoecological Database.

**B.**

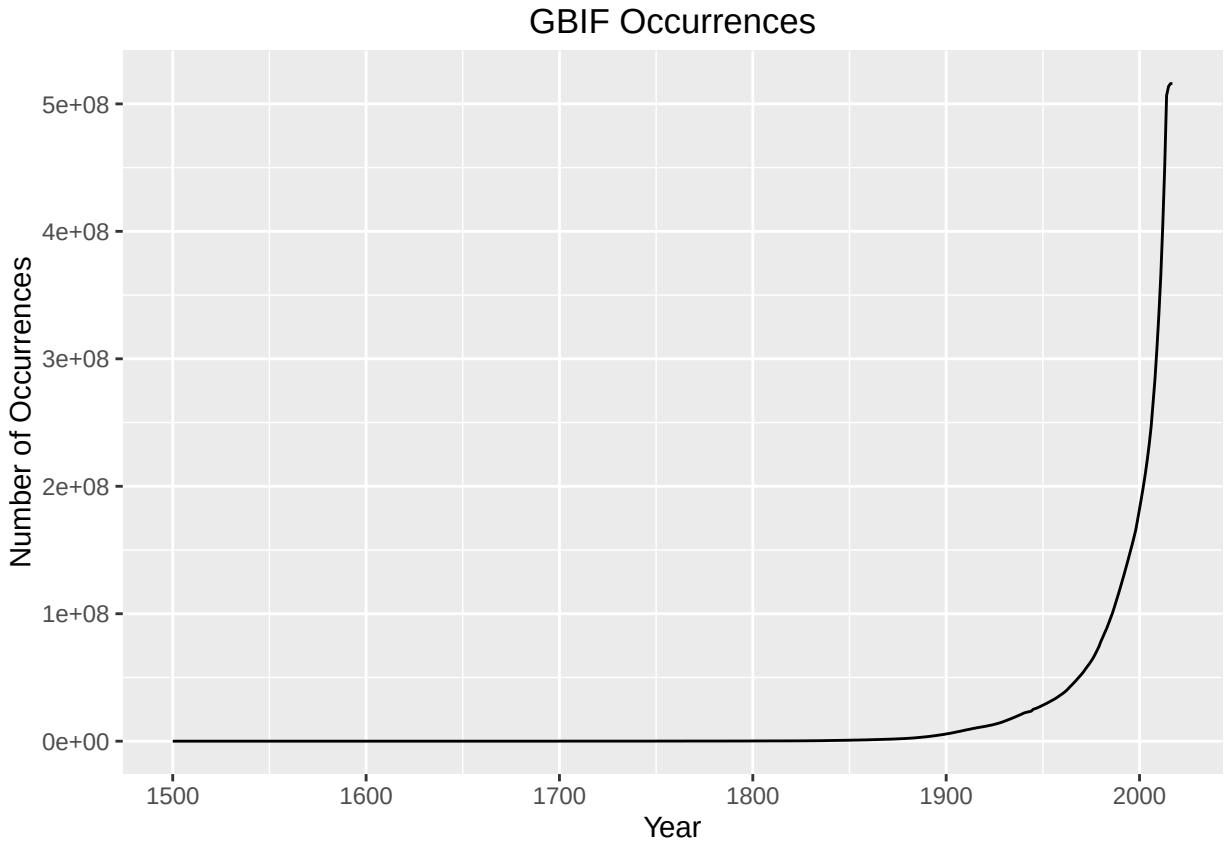



*Figure 2A* shows the massive influx of occurrence records in the Global Biodiversity Information Facility. Note that digitization of existing records allows GBIF's holdings to preceed it's organization in 2001.

**Figure 3**

**A.**

```
df$type <- as.factor(df$type)
ggplot(df, aes(df$type)) + geom_bar() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ggtitle("Dataset Types in Neotoma") +
  xlab("") + ylab("Count")
```

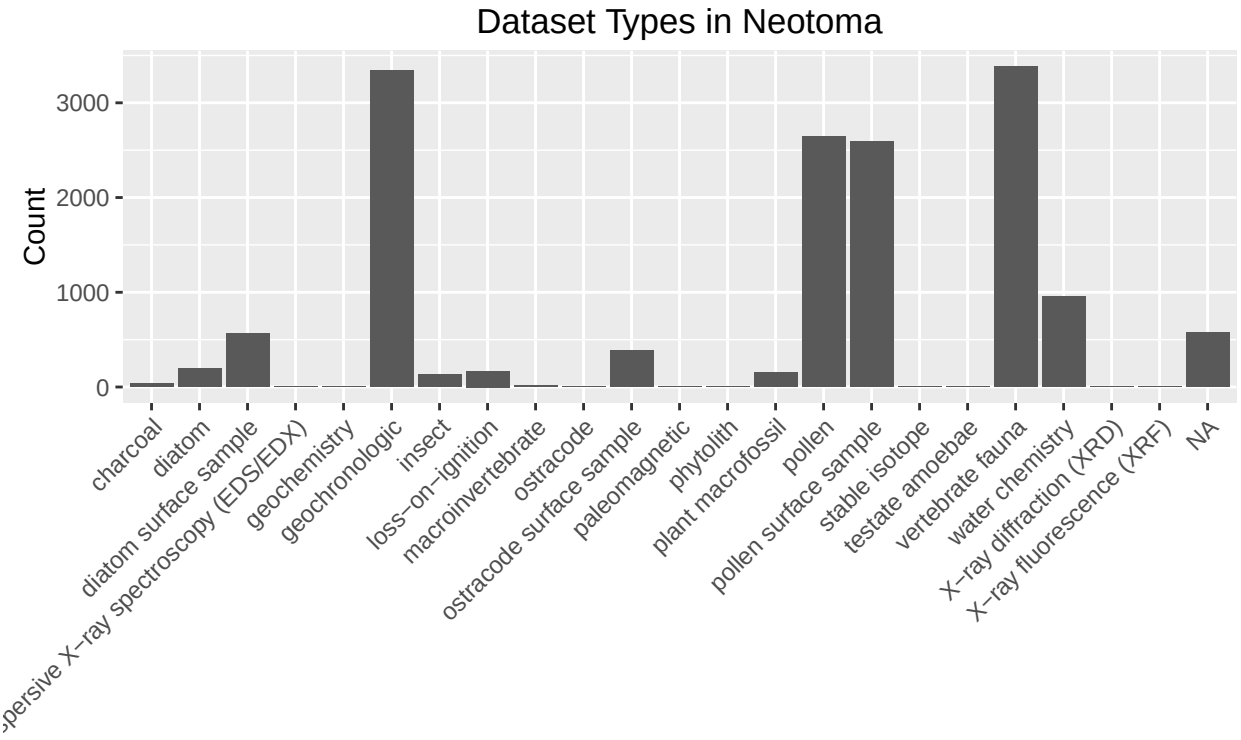



*Figure 3A* shows the relative proportion of each of the 23 dataset types in the Neotoma Paleoecological Database.

**B.**

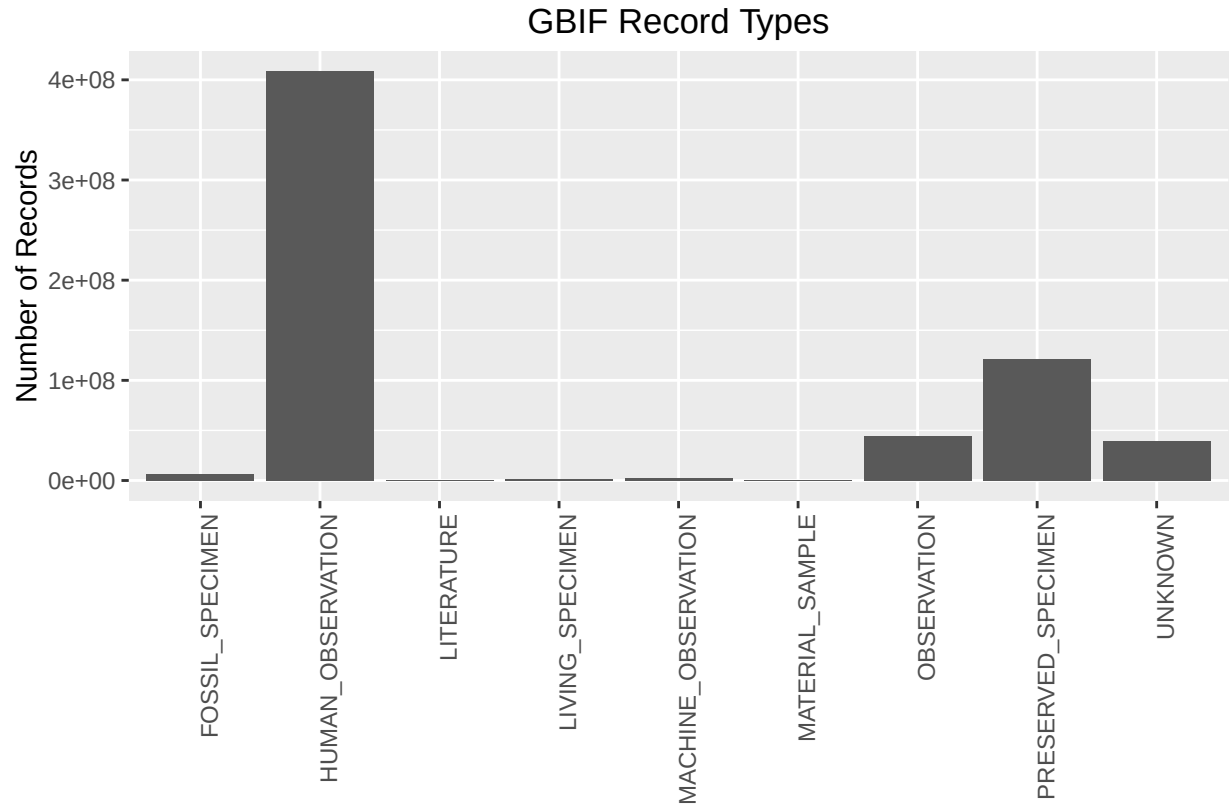


*Figure 3B* shows the relative proportion of each of the eight record types in the GBIF dataset.

**Figure 4**

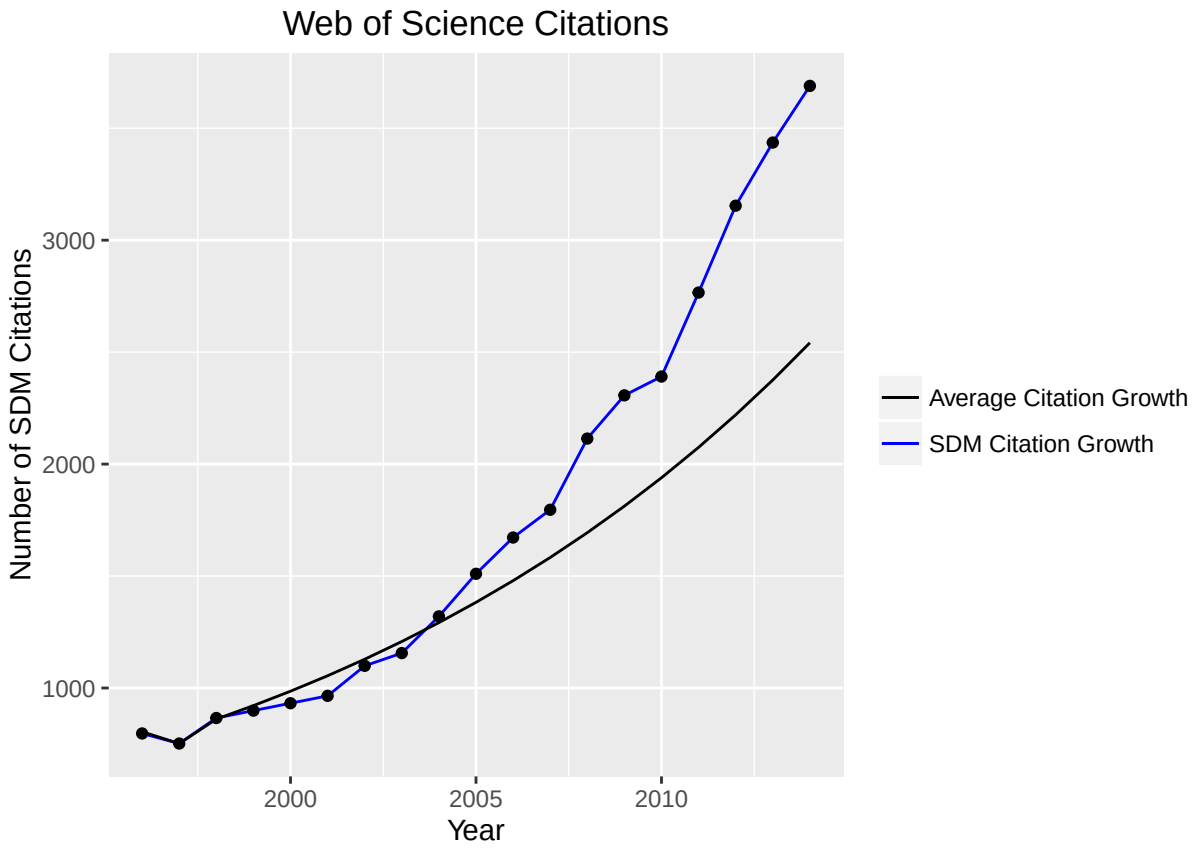



*Figure 4* shows that the recent growth in citations for ecological forecasting models far outpaces the average citation growth in all of STEM fields. SDM citation growth was established from a Web of Science query for ("Ecolgical Niche Model" OR "Species Distribution Model" OR "Habitat Suitability Model") and average citation growth was derived from the National Science Board report on Science and Engineering indicators (2014).

**Figure 5**

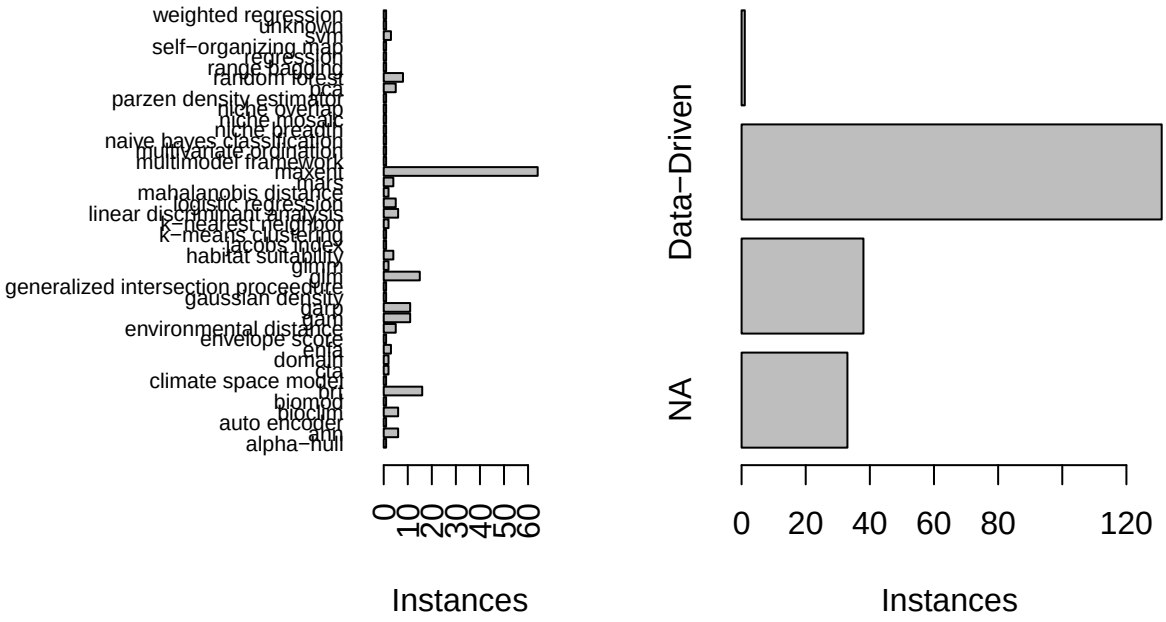



*Figure 5* reports the relative proportions of algorithms used in 100 randomly sampled modeling studies. Instances were classified according to their classification in the data-driven/model-drive/Bayesian framework. In total, 203 model instances were reviewed in 100 papers. 42 unique algorithms were employed.

**Figure 6**

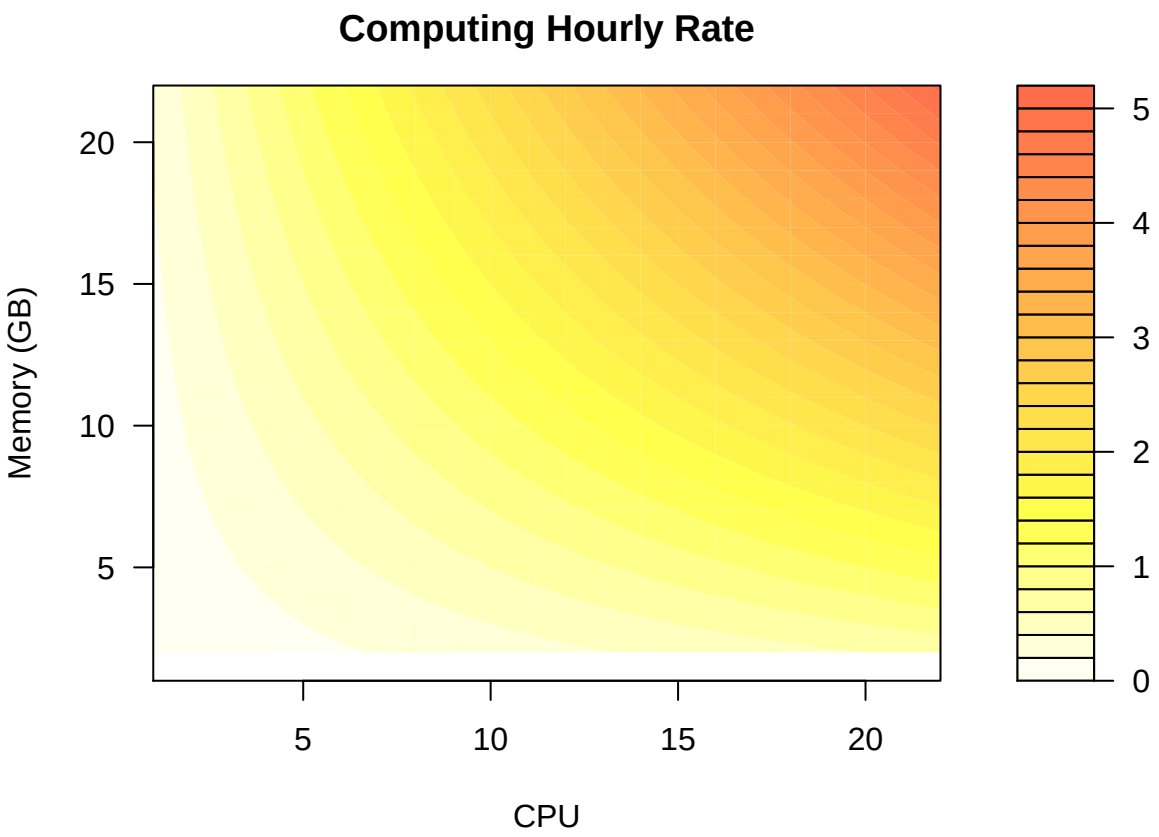
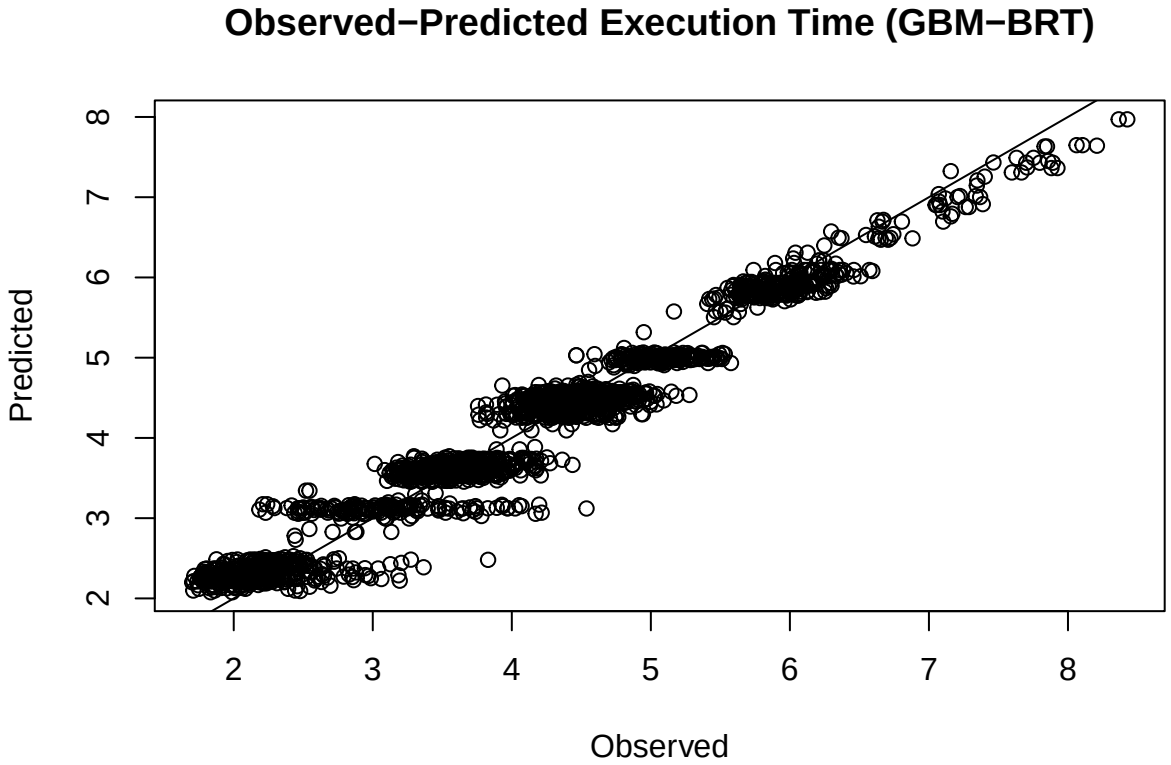



*Figure 6* demonstrates the cost surface faced by consumers of Google's Cloud Computing Engine. Rates are in $/hr. Note the tradeoff in relative increases in one of the computing components for the same total rate.

***Figures from this point onwards are not referenced in the thesis yet***

**Figure 7**

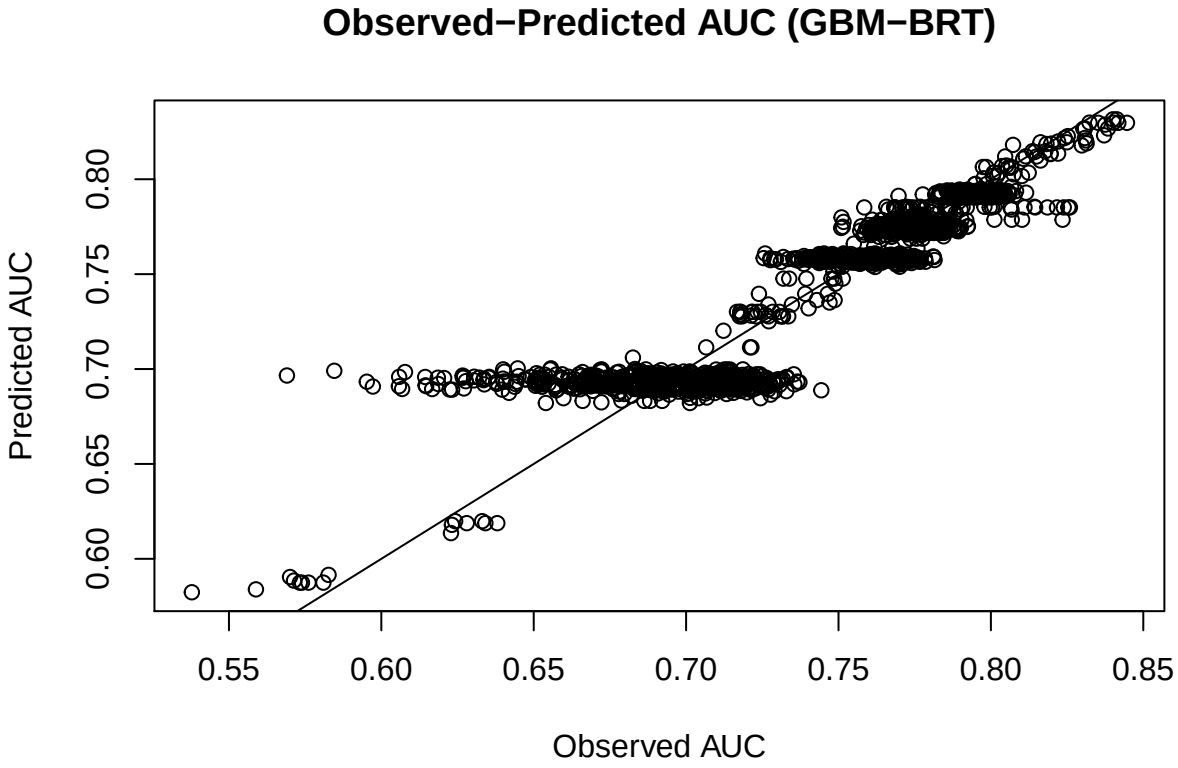



```
## [1] "Runtime Model Mean Squared Error:  0.0724812628107845"
```

```
## [1] "Runtime Model Percent Variance Explained:  0.956158761045389 %"
```

*Figure 7* shows the model-data comparison for the GBM-BRT model running time as predicted by a random forest ensemble of 100 trees.

**Figure 8**




```
## [1] "Accuracy Model Mean Squared Error:  0.000236331688872181"
```

```
## [1] "Accuracy Model Percent Variance Explained:  0.873307862053914 %"
```

*Figure 8* shows the model-data comparison for the GBM-BRT accuracy as predicted by a random forest ensemble of 100 trees.

**Figure 9**

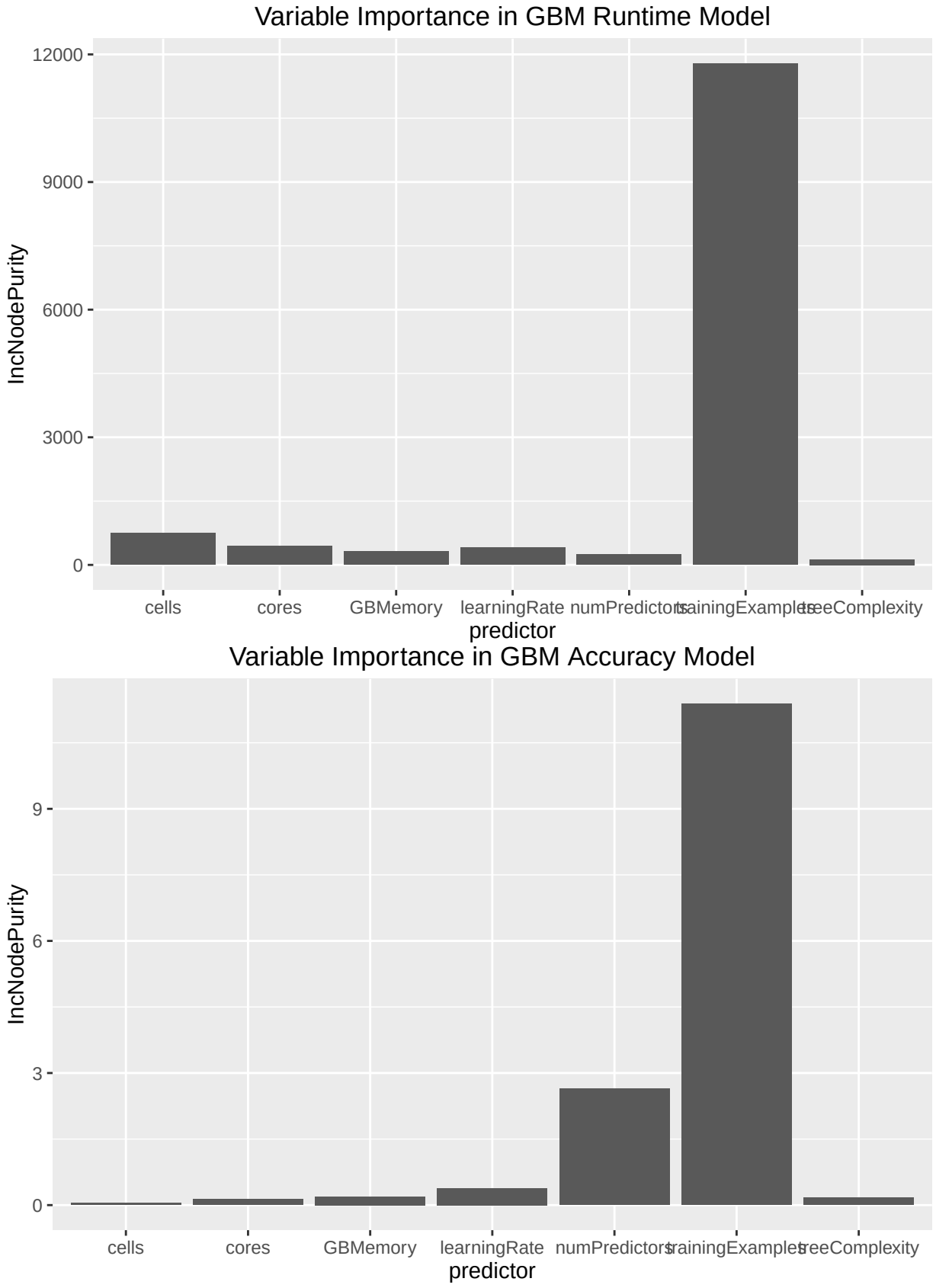
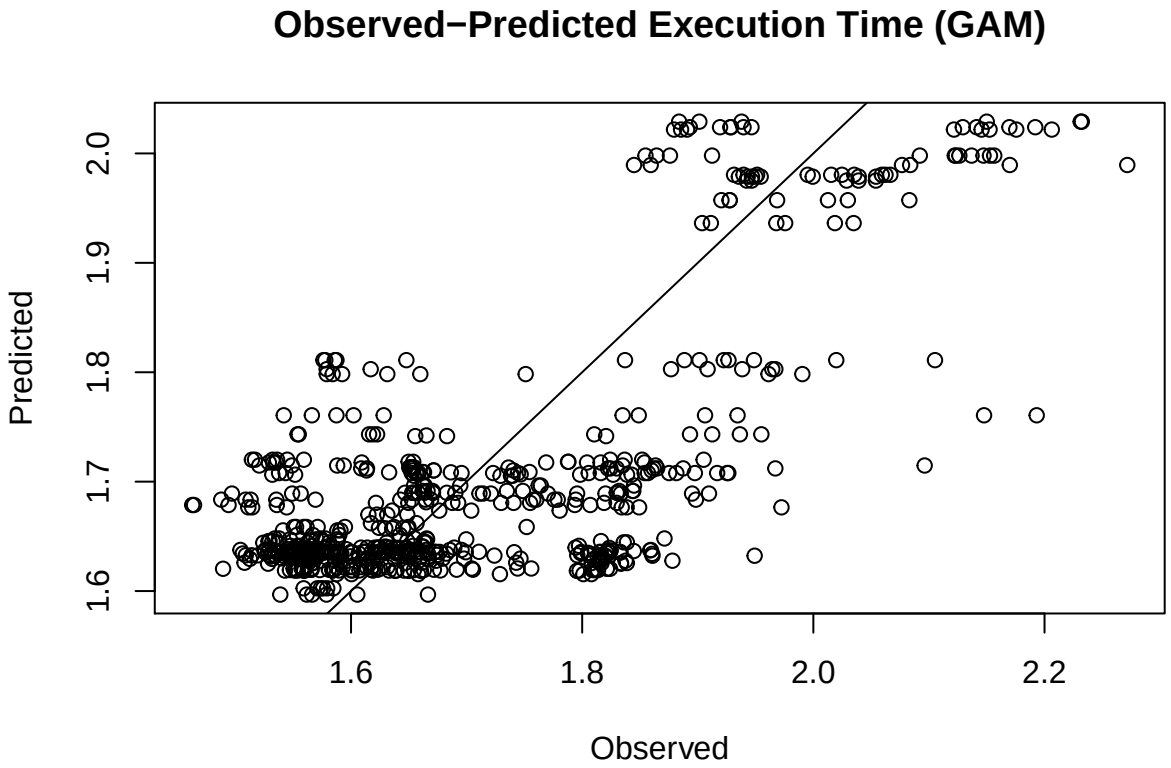



*Figure 9A* shows the relative importance of variables in the model of runtime for the GBM-BRT. *Figure 9B*

shows the relative importance of variables in the model of accuracy for the GBM-BRT.

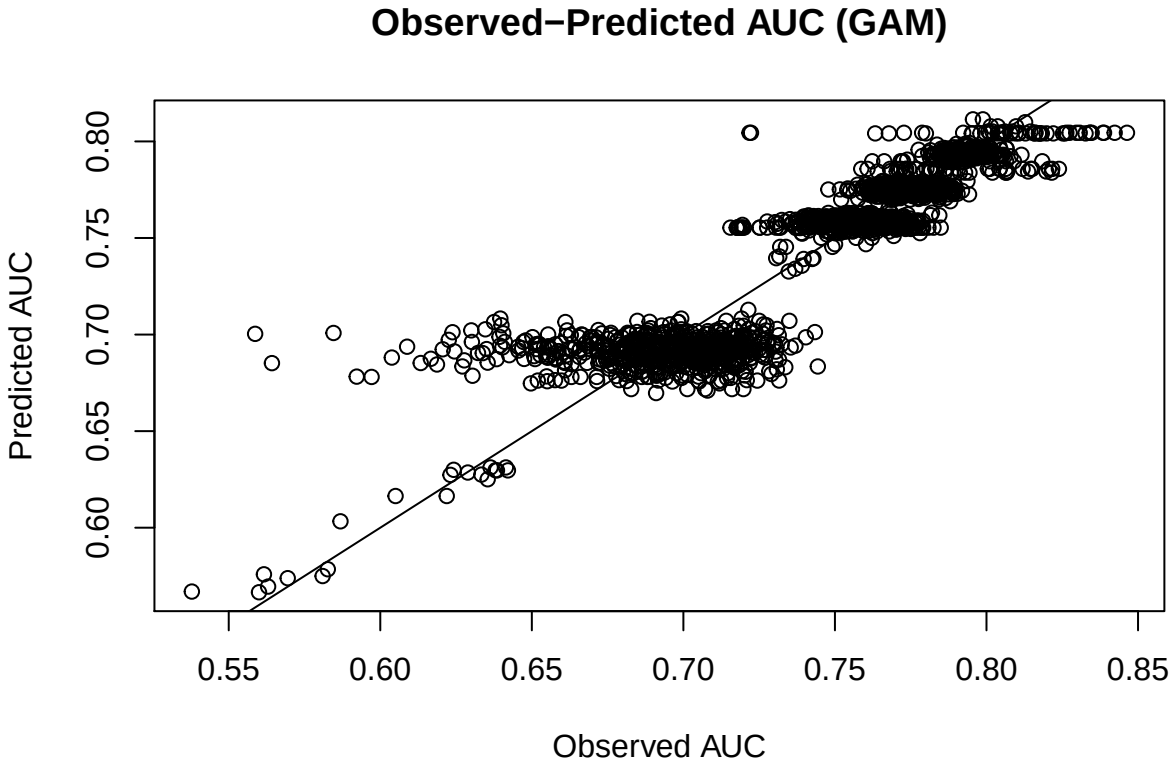
**Figure 10**




```
## [1] "Runtime Model Mean Squared Error:  0.0127041994990109"
## [1] "Runtime Model Percent Variance Explained:  0.511636084359816 %"
```

*Figure 10* shows the model-data comparison for the runtime of the generalized additive model (GAM) SDM.

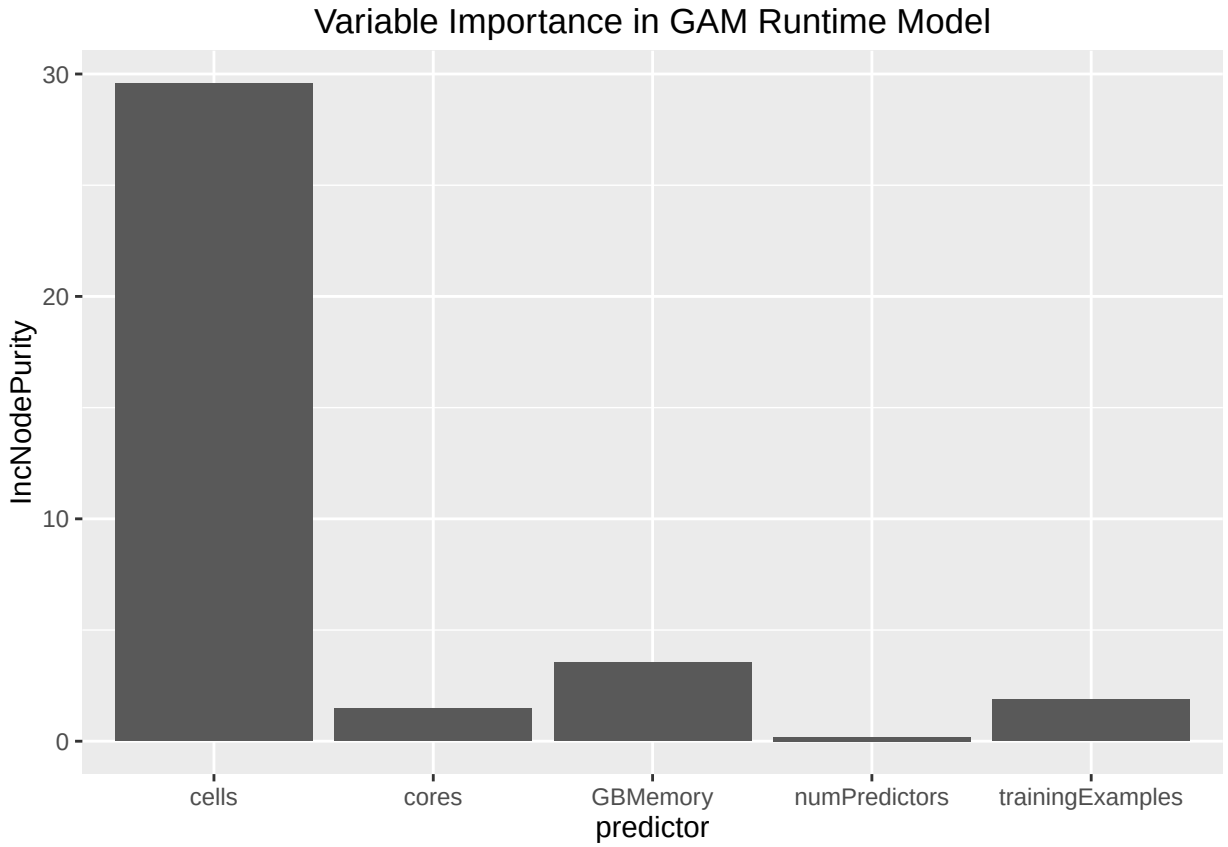
**Figure 11**




```
## [1] "Accuracy Model Mean Squared Error:  0.000274156074036223"
## [1] "Accuracy Model Percent Variance Explained:  0.854307852575571 %"
```

*Figure 11* shows the model-data comaprison for the model of GAM accuracy.

**Figure 12**



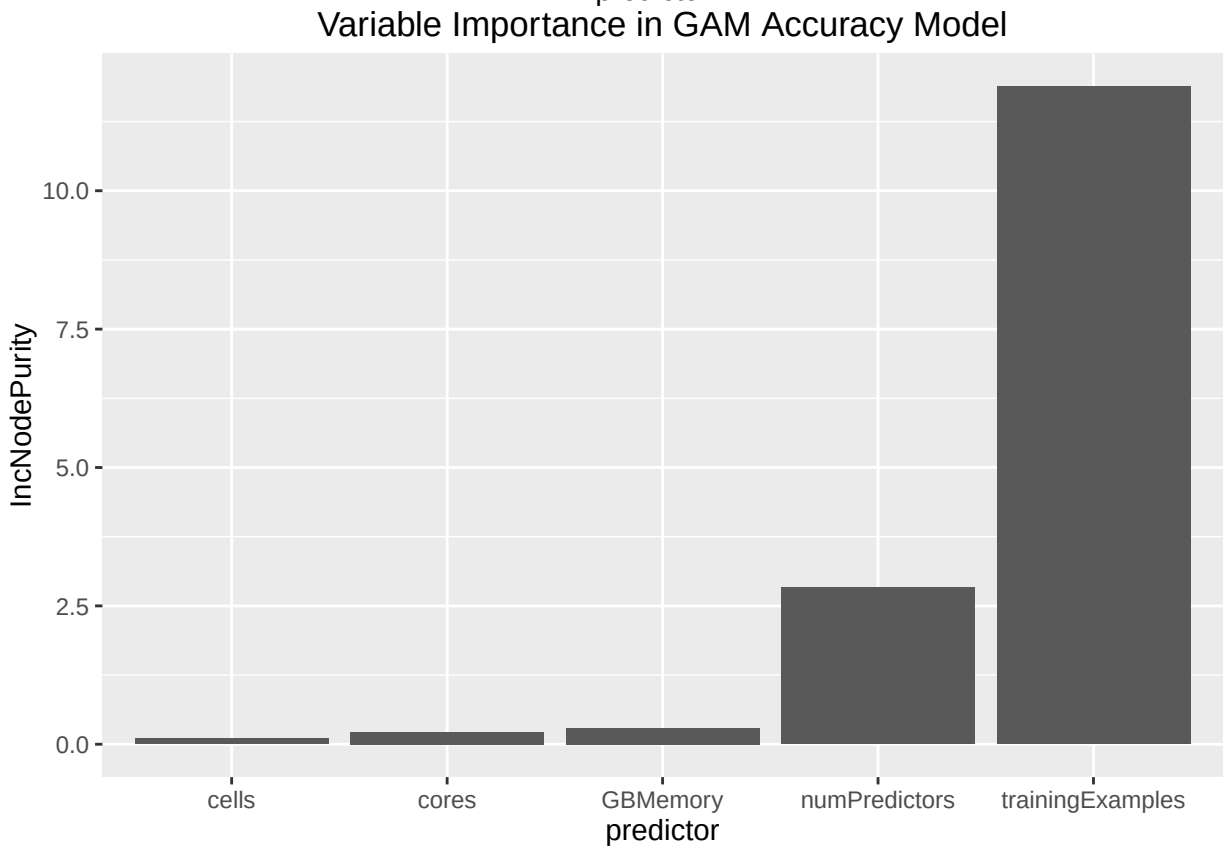
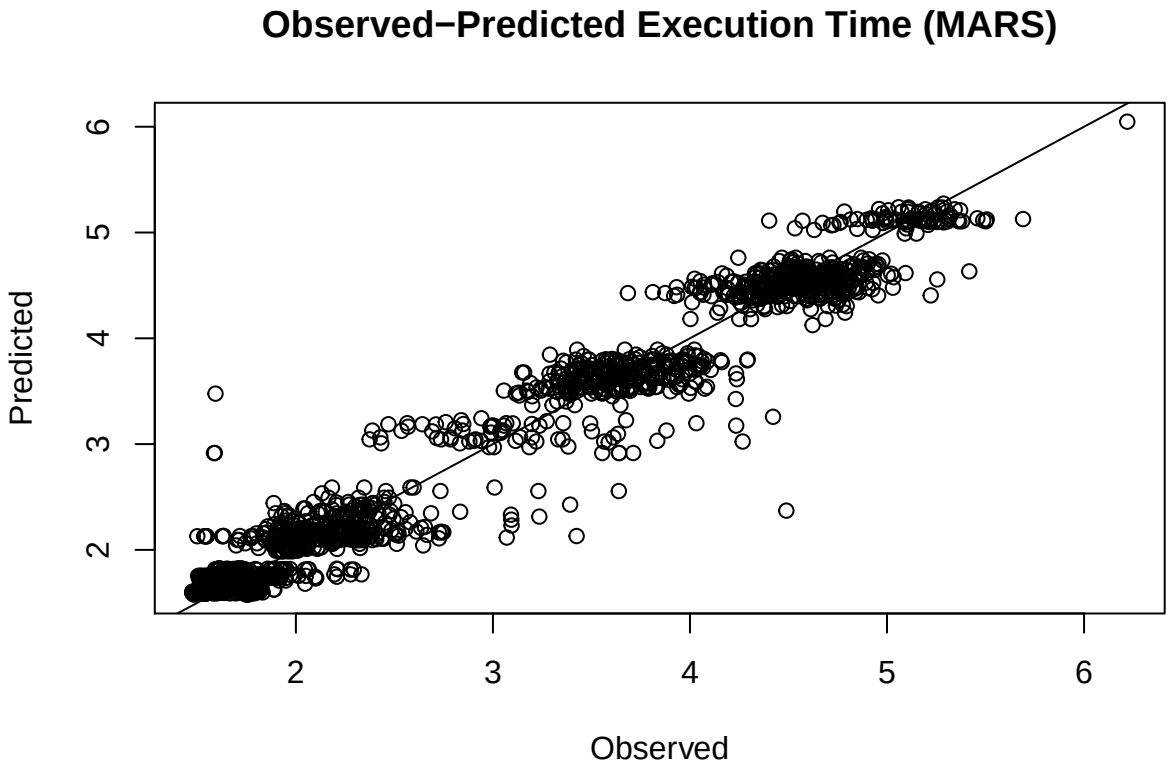




*Figure 12A* shows the relative importance of each variable in the GAM runtime model. *Figure 12B* shows the

relative importance of each variable in the GAM accuracy model.

**Figure 13**

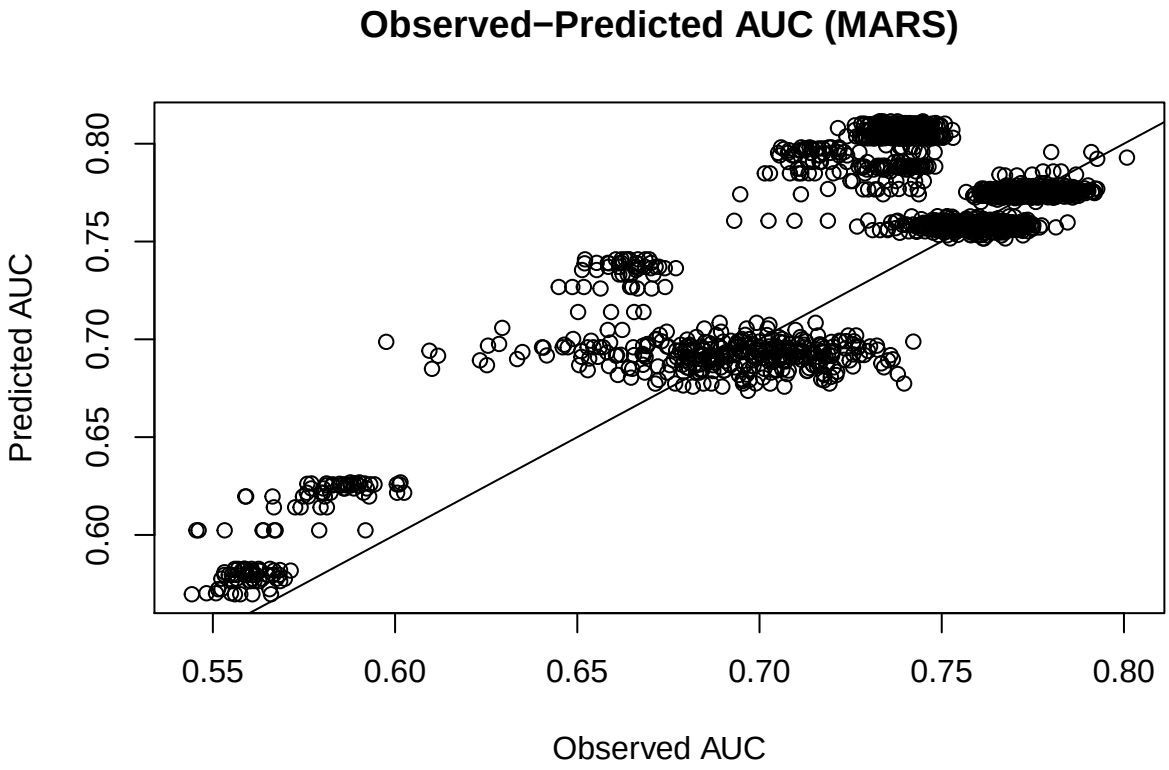



```
## [1] "Runtime Model Mean Squared Error:  0.0629078417271952"
```

```
## [1] "Runtime Model Percent Variance Explained:  0.960660810056395 %"
```

*Figure 13* shows the model-data comparison of the runtime model for the multivariate adaptive regression splines (MARS) SDM.

**Figure 14**




```
## [1] "Accuracy Model Mean Squared Error:  0.000274760883657695"
## [1] "Accuracy Model Percent Variance Explained:  0.853986444367327 %"
```

*Figure 14* shows the model-data comparison for the model of MARS accuracy.

**Figure 15**

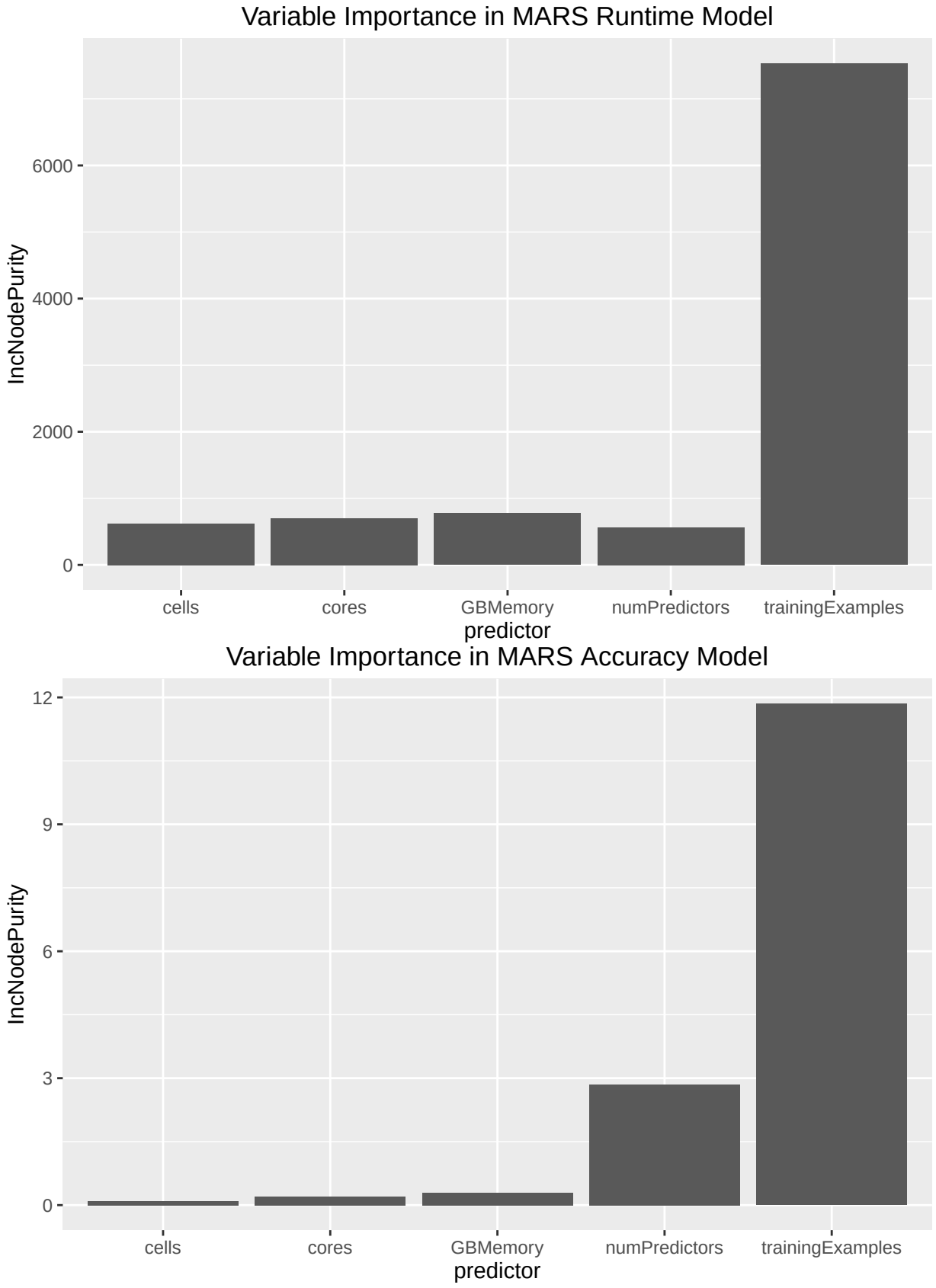
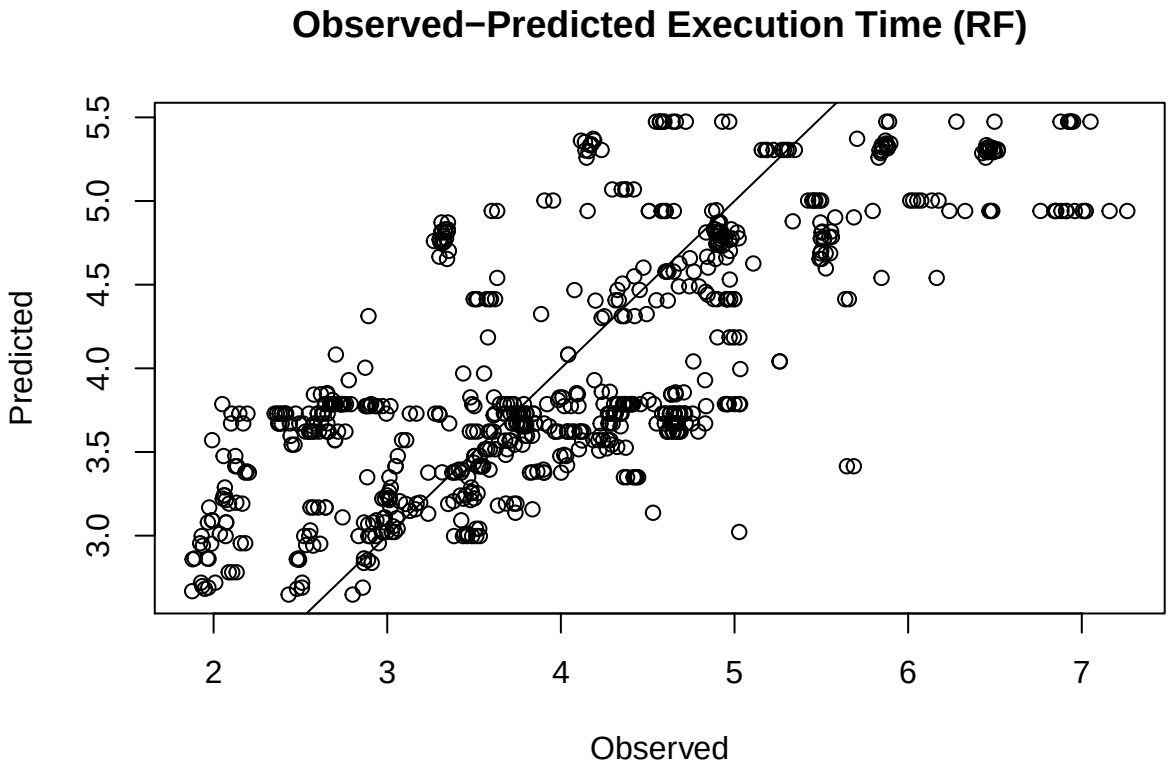



*Figure 15A* shows the relative importance of variables in the MARS runtime model. *Figure 15B* shows the

relative importance of variables in the MARS accuracy model.

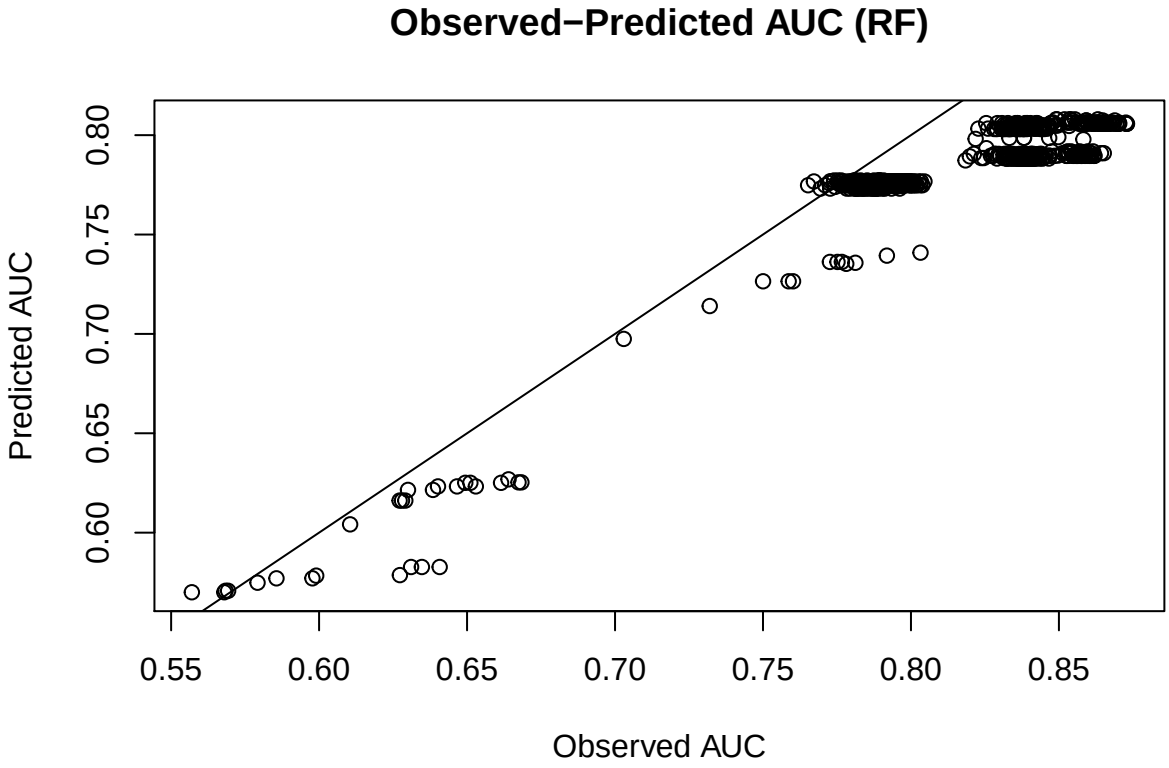
**Figure 16**




```
## [1] "Runtime Model Mean Squared Error:  0.667586429571664"
## [1] "Runtime Model Percent Variance Explained:  0.539064359980998 %"
```

*Figure 16* shows the model-data comparison for the random forest (RF) model of runtime.

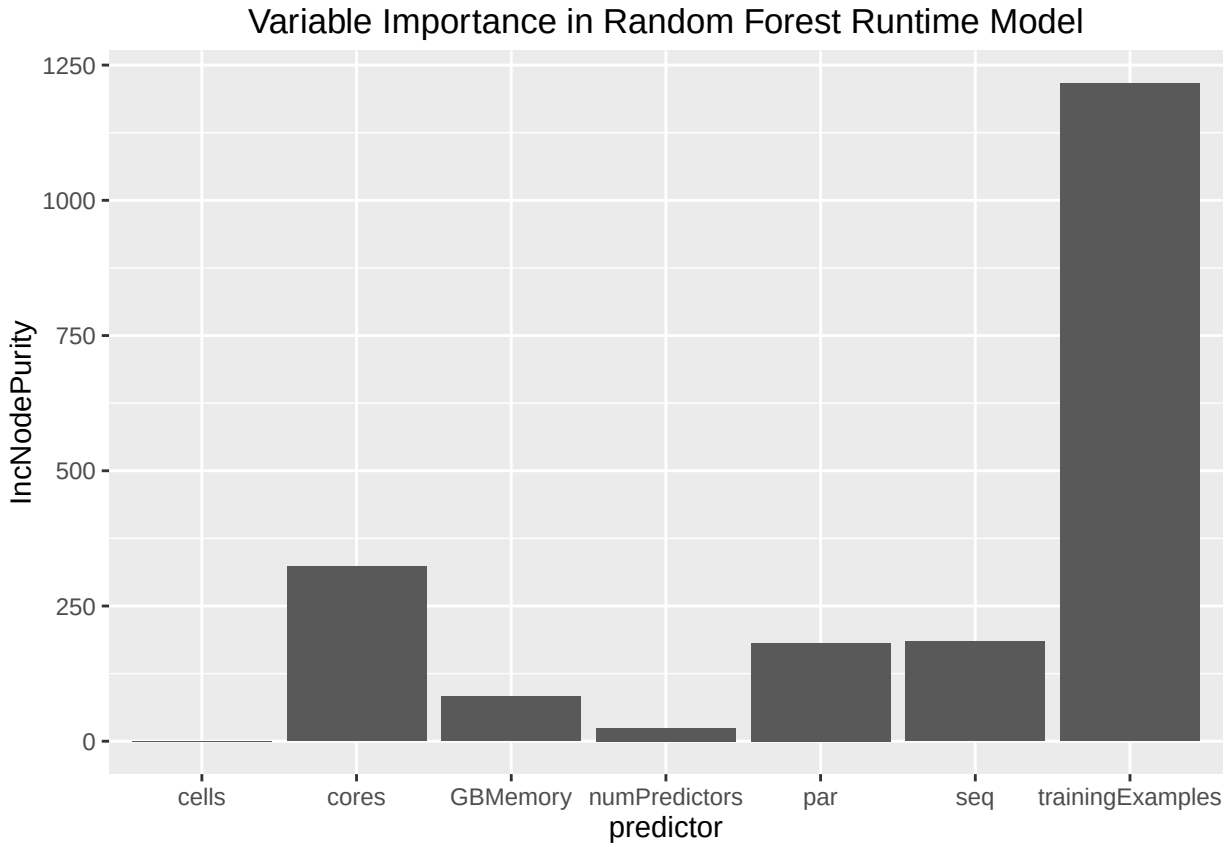
**Figure 17**




```
## [1] "Accuracy Model Mean Squared Error:  0.000275733380026873"
## [1] "Accuracy Model Percent Variance Explained:  0.85346963989788 %"
```

*Figure 17* shows the model-data comparison for the RF model of accruacy.

**Figure 18**



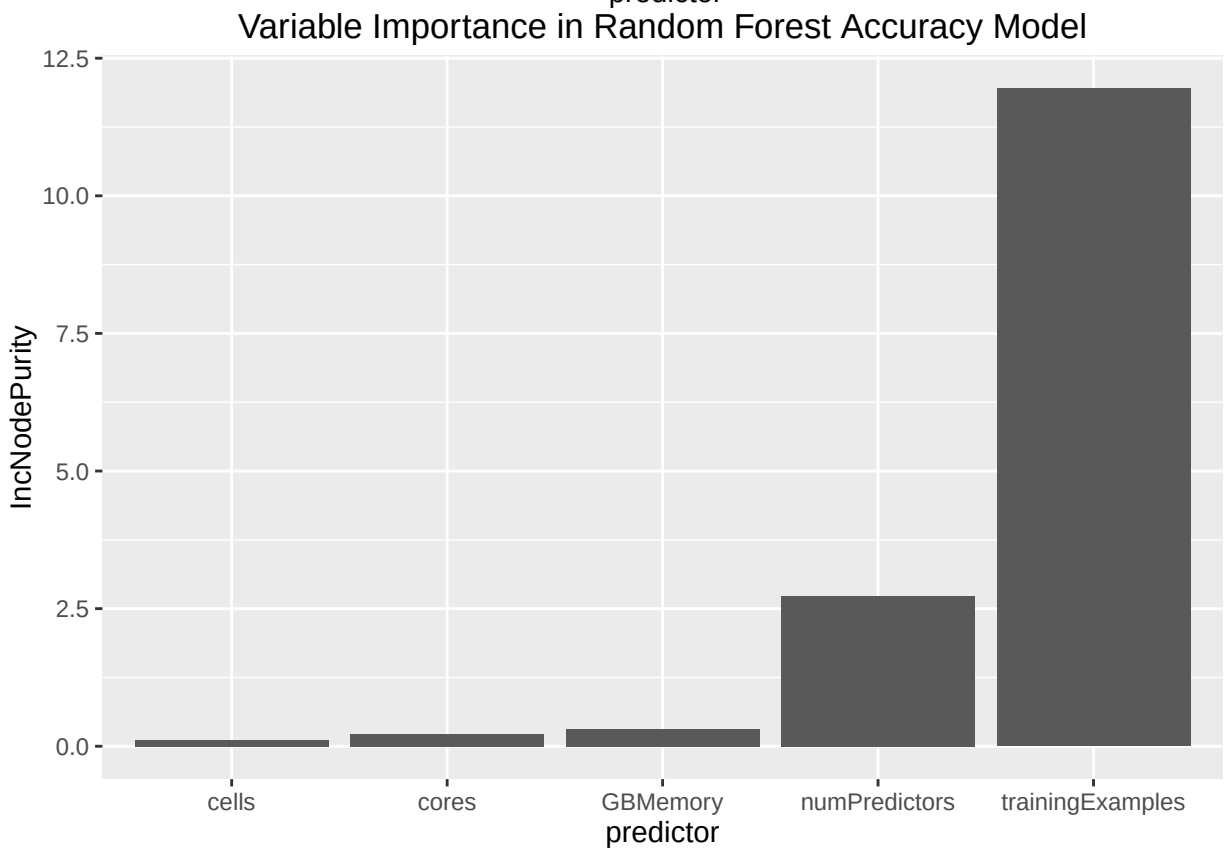




*Figure 18A* shows the relative importance of each of the variables used in the RF runtime model. *Figure 18B*

shows the relative importance of each of the variables used in the RF accuracy model.

**Figure 19**

```
##
## Attaching package: 'plyr'
## The following object is masked from 'package:lubridate':
##
##     here
## The following object is masked from 'package:maps':
##
##     ozone
```

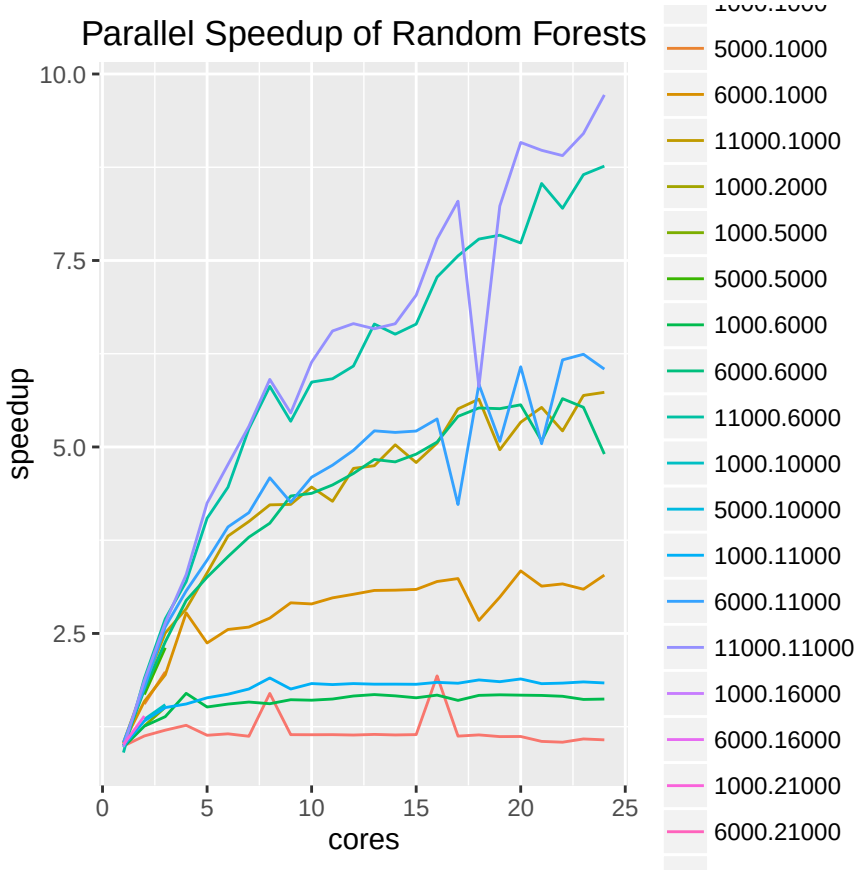



*Figure 19* shows that more expensive workloads benefit more from additional cores than simple modeling routines.

**Figure 20**

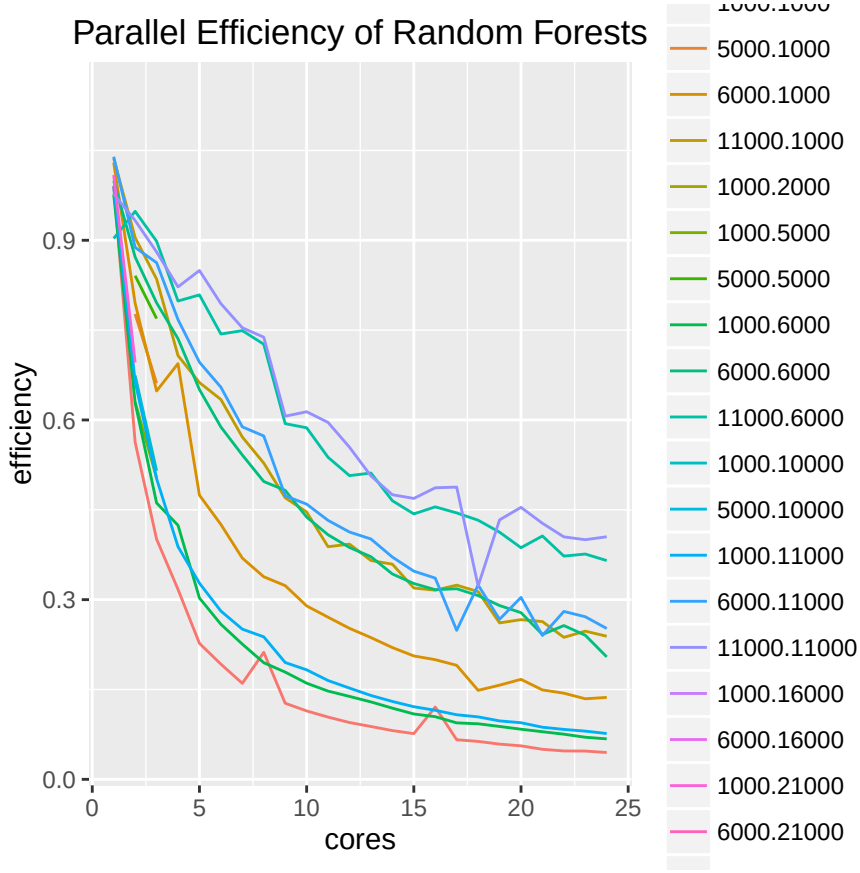
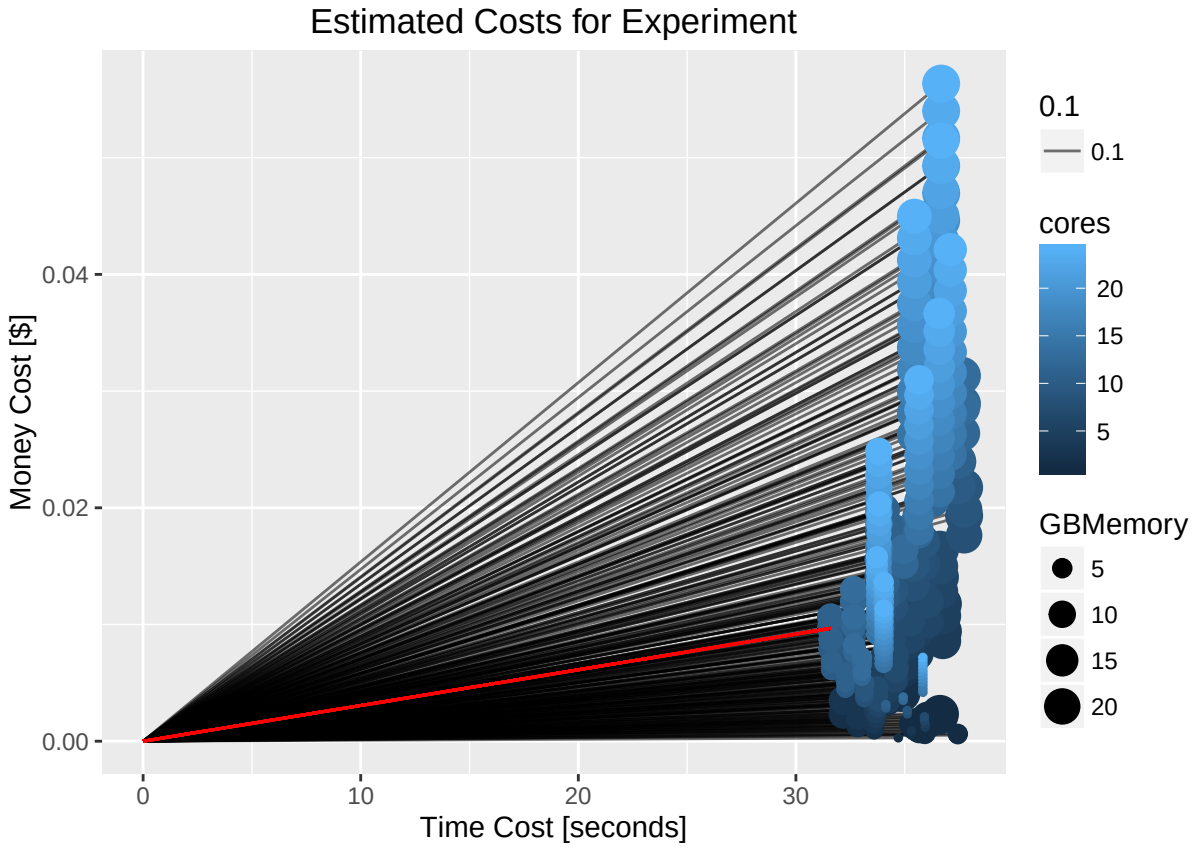


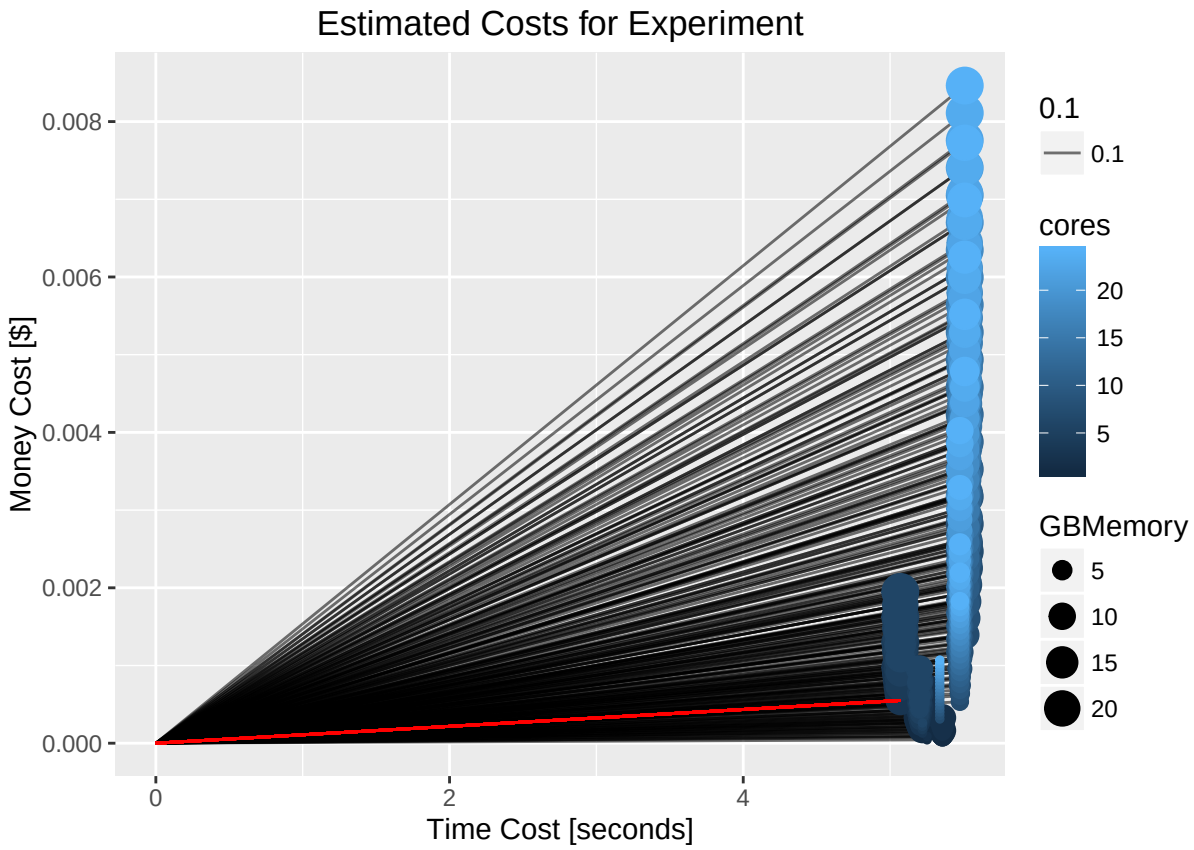

*Figure 20* shows the diminishing marginal returns of using additional cores. Note that simple workflows, though benefiting from additional cores, drop off steeply, while complex workloads decline nearly linearly.

**Figure 22**

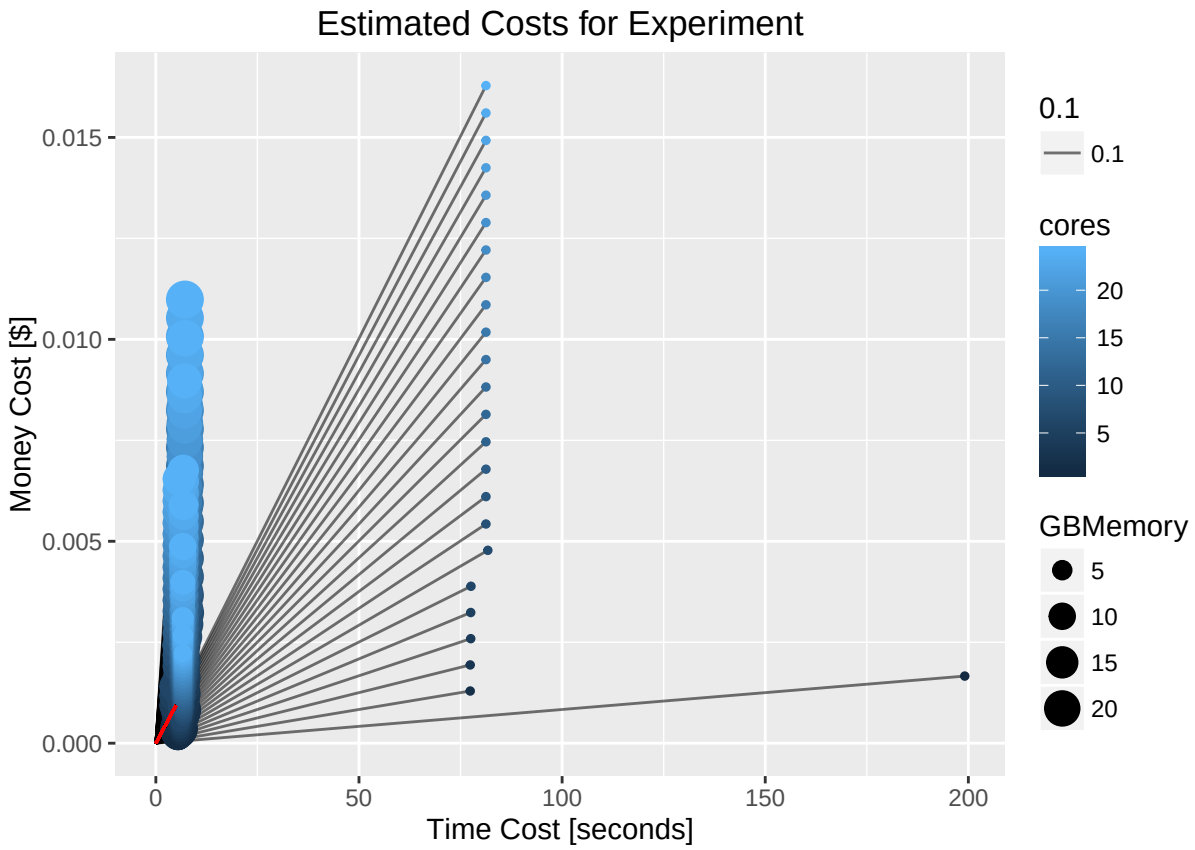
```
## [1] "Model:  GBM-BRT  using n= 500 and p = 5"
```



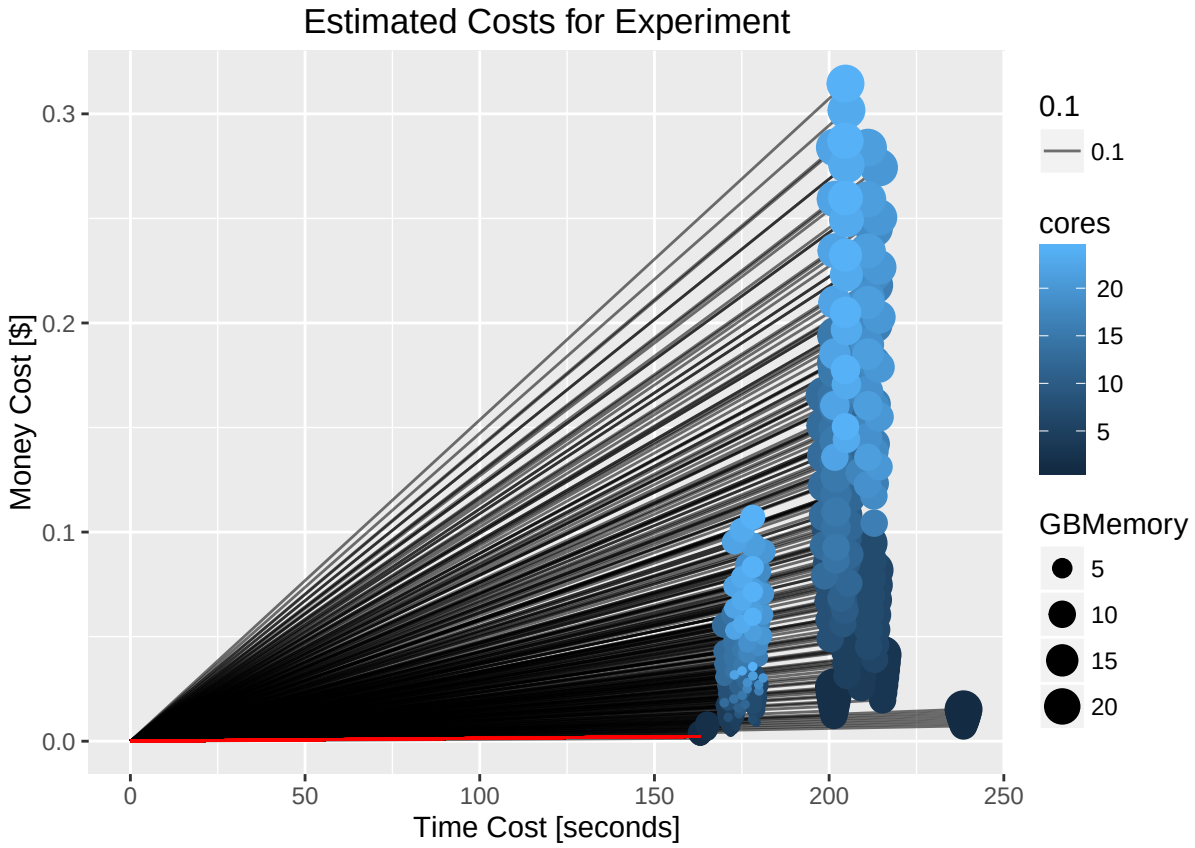

```
## [1] "Estimated Cost ($):  0.0096740474205723"
## [1] "Estimated Cost (seconds):  31.5973241825987"
## [1] "Optimal # Cores:  10"
## [1] "Optimal RAM:  10"
## [1] "Model:  GAM  using n= 11000 and p = 2"
```




```
## [1] "Estimated Cost ($):  0.000550310759479872"
## [1] "Estimated Cost (seconds):  5.0696523213254"
## [1] "Optimal # Cores:  3"
## [1] "Optimal RAM:  12"
## [1] "Model:  MARS  using n= 8000 and p = 2"
```




```
## [1] "Estimated Cost ($):  0.000935808532670277"
## [1] "Estimated Cost (seconds):  4.94439168371052"
## [1] "Optimal # Cores:  4"
## [1] "Optimal RAM:  16"
## [1] "Model:  RF  using n= 11000 and p = 5"
```




```
## [1] "Estimated Cost ($):  0.00226993803576235"
## [1] "Estimated Cost (seconds):  163.109319935019"
## [1] "Optimal # Cores:  1"
## [1] "Optimal RAM:  4"
```

*Figure 22* demonstrates calculating the optimal computing configuration for five randomly drawn experiments. I first calculate the predicted execution time and cost under 200+ configuration types, using the Google Cloud Engine cost curves. Then, I plot them in cartesian space, with time cost and dollar cost on orthogonal axes. Then, I find the euclidean distance between each point and the origin (0,0). Finally, I find the minimum distance between the origin and a point, and call that point the optimal.

**Figure 21**

Move from a machine-learning to a Bayesian approach. Run a simple linear model, estimating coefficients with MCMC simulations, then predict the execution time for a testing set of n=50.

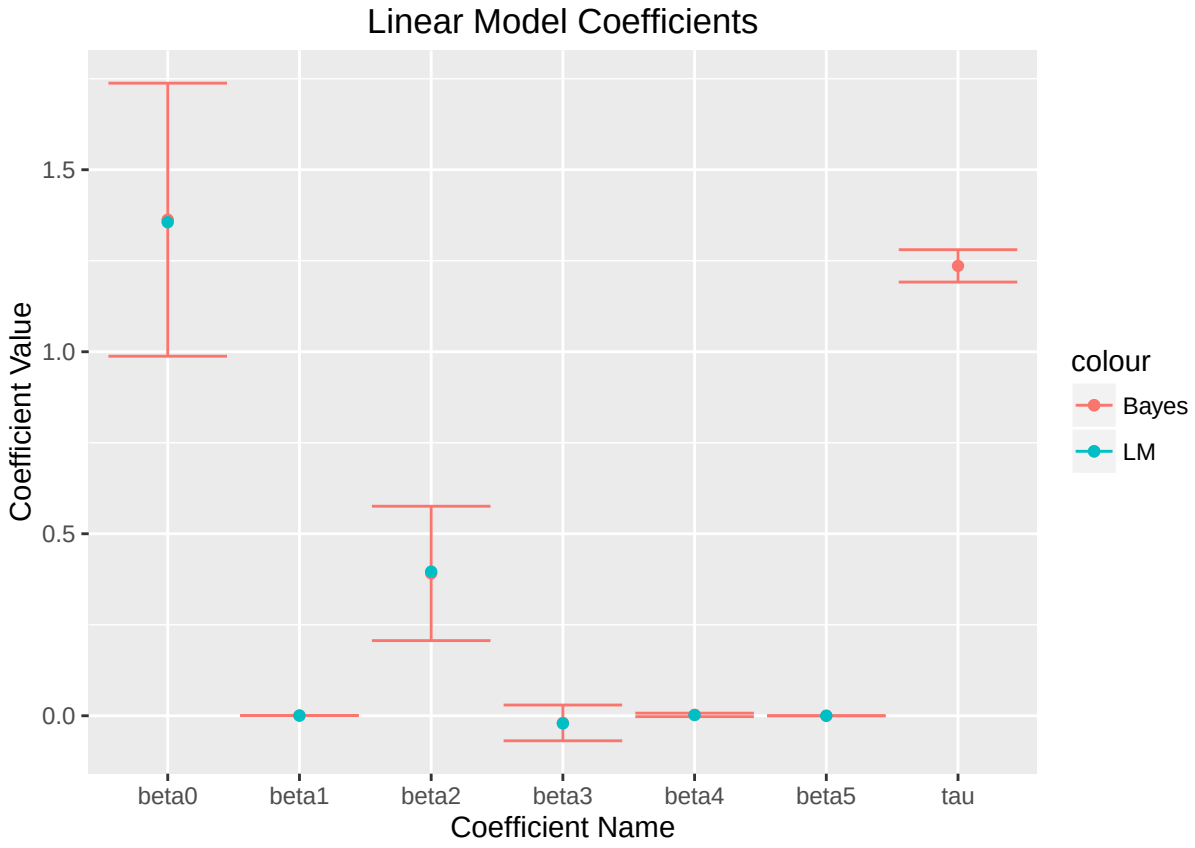
```
## Loading required package: rjags
## Loading required package: coda
## Linked to JAGS 4.2.0
## Loaded modules: basemod,bugs
##
## Attaching package: 'R2jags'
## The following object is masked from 'package:coda':
##
##     traceplot
```

```
## module glm loaded
```

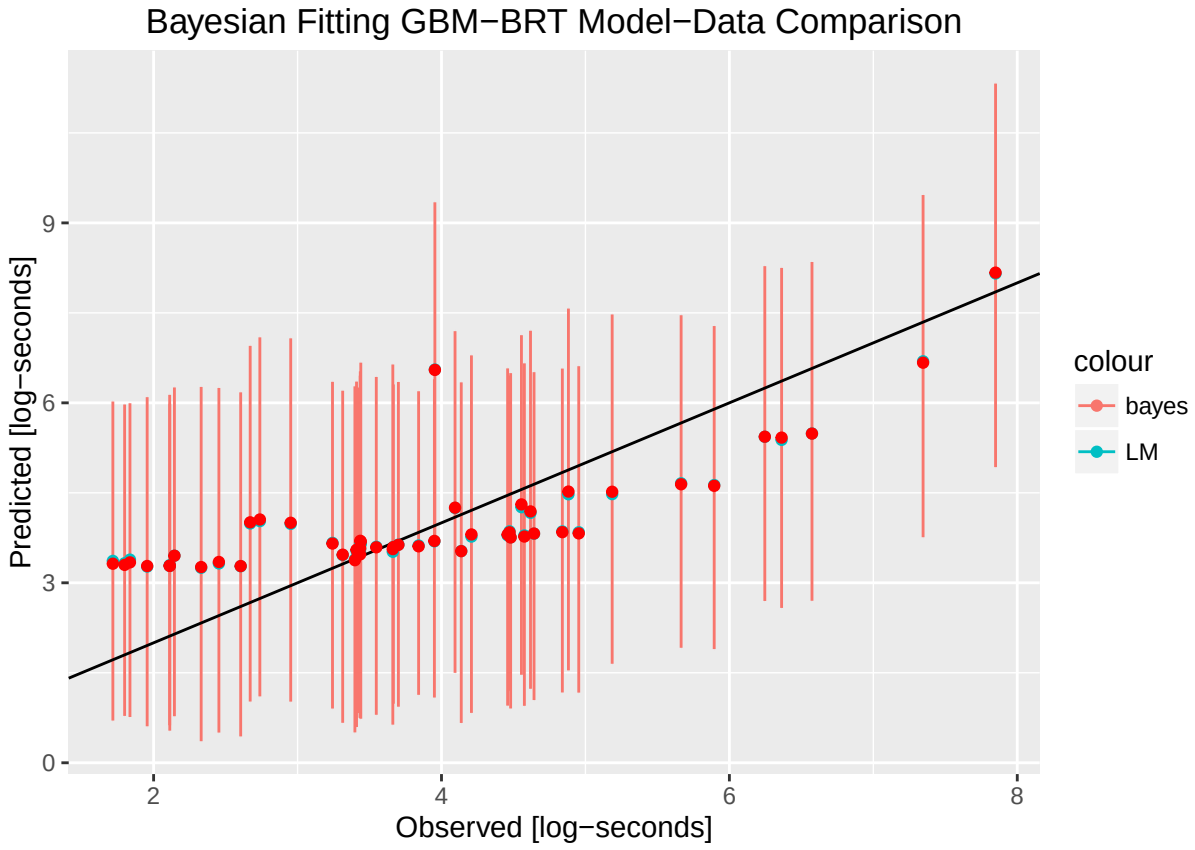
```
## module dic loaded
```

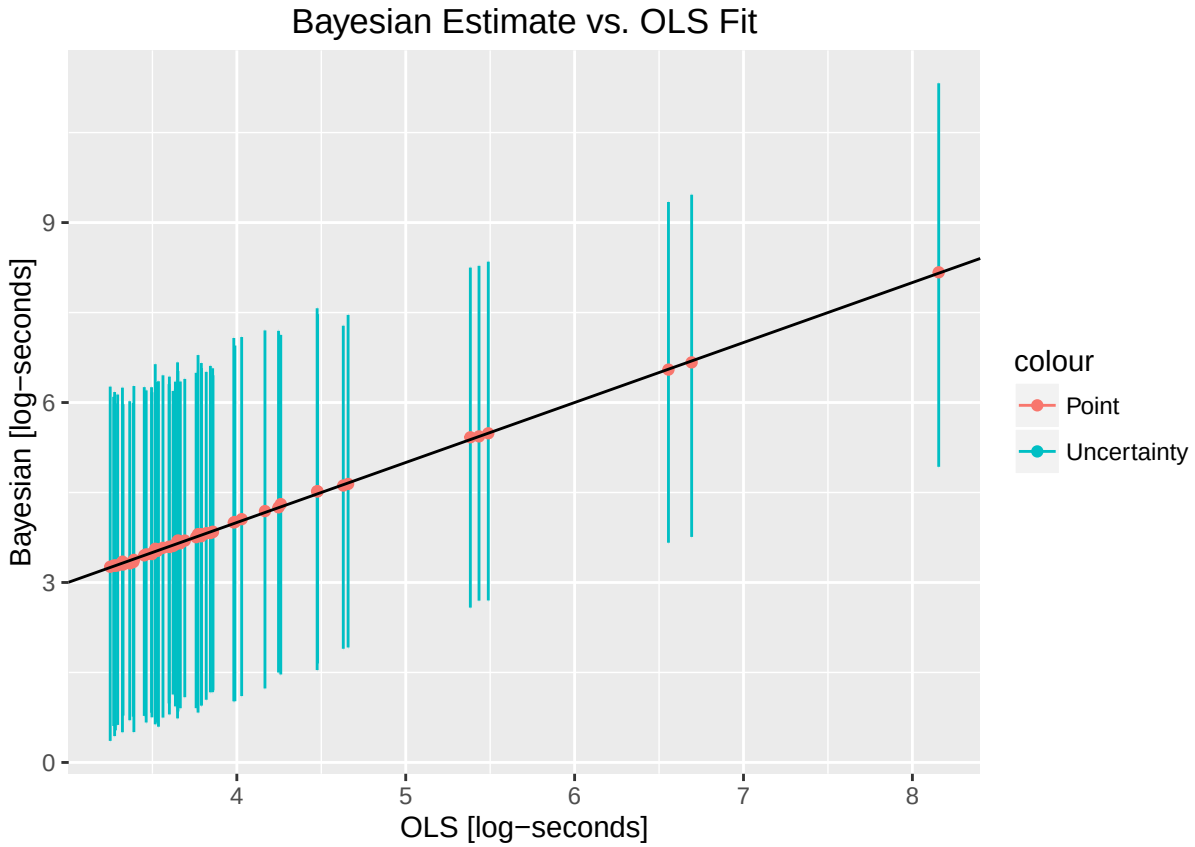
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 11570
##    Unobserved stochastic nodes: 7
##    Total graph size: 70844
##
## Initializing model
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```




```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 0
##    Unobserved stochastic nodes: 57
##    Total graph size: 444
##
## Initializing model
```


Bayesian Fitting GBM−BRT Model−Data Comparison
Predicted [log−seconds]
Observed [log−seconds]
0
3
6
9
2
4
6
8
colour
bayes
LM


Bayesian Estimate vs. OLS Fit
Bayesian [log−seconds]
OLS [log−seconds]
0
3
6
9
4
5
6
7
8
colour
Point
Uncertainty

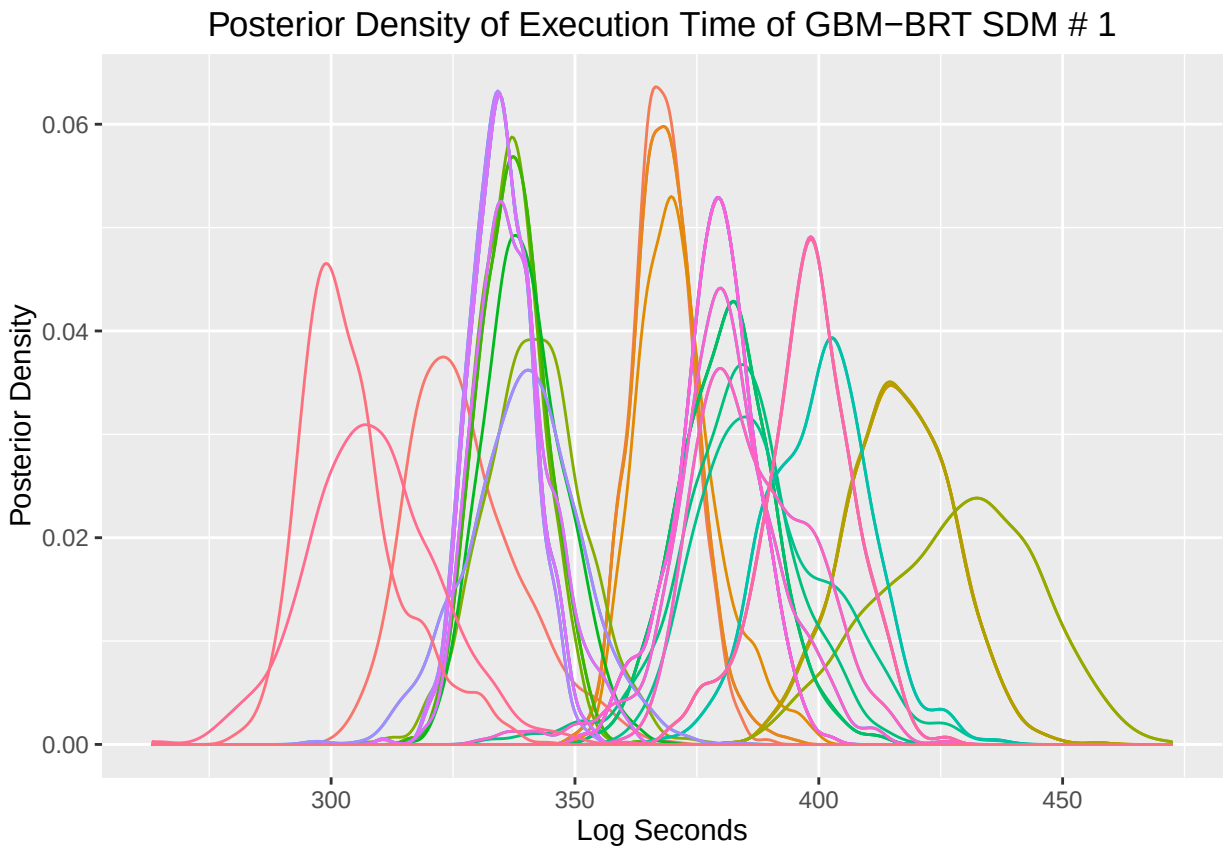
```
## [1] "Model Explains  0.622382105137496 % of data variance."
```

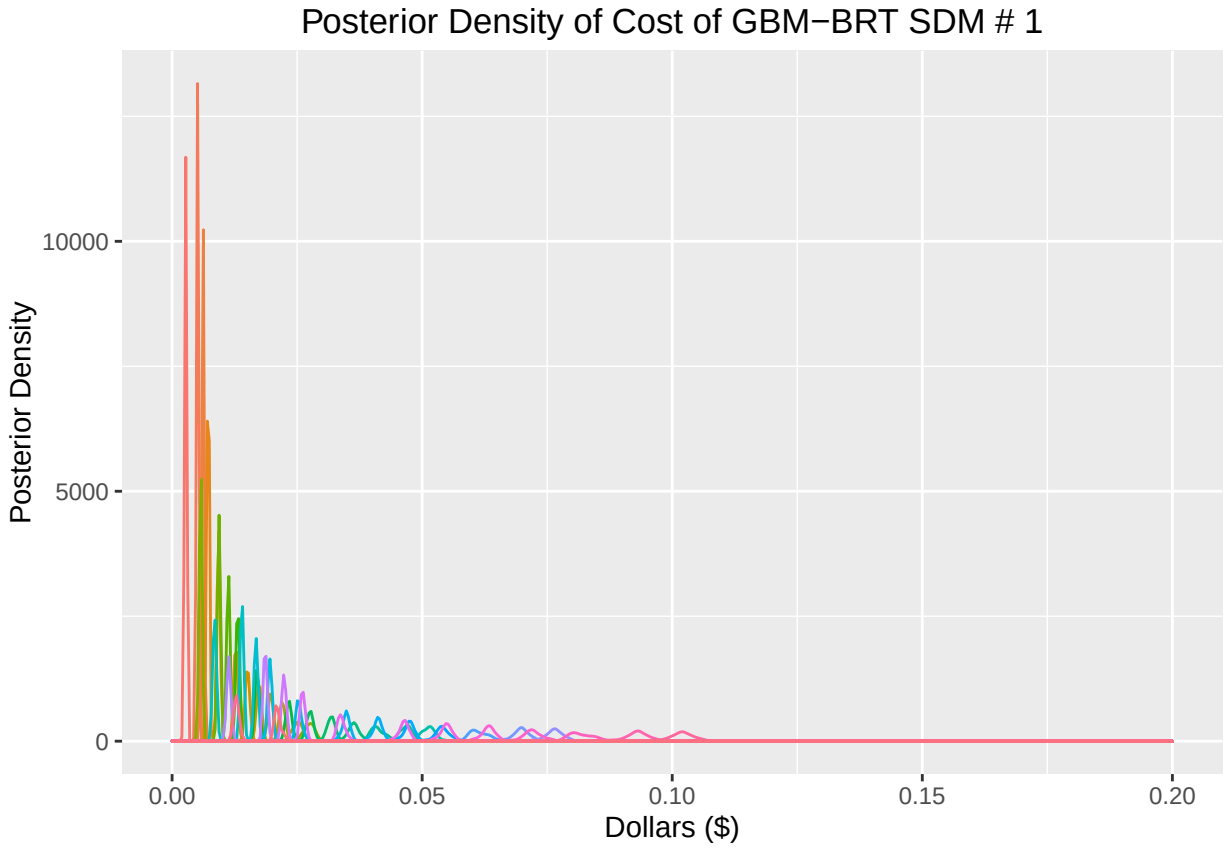
*Figure 21* shows the results of fitting a model using draws from an MCMC instead of OLS. Notice the close agreement between OLS and posterior means. Also, we now have uncertainty estimates on all of our results.

**Figure 23**

```
## Warning: replacing previous import 'lme4::sigma' by 'stats::sigma' when
## loading 'pbkrtest'
```

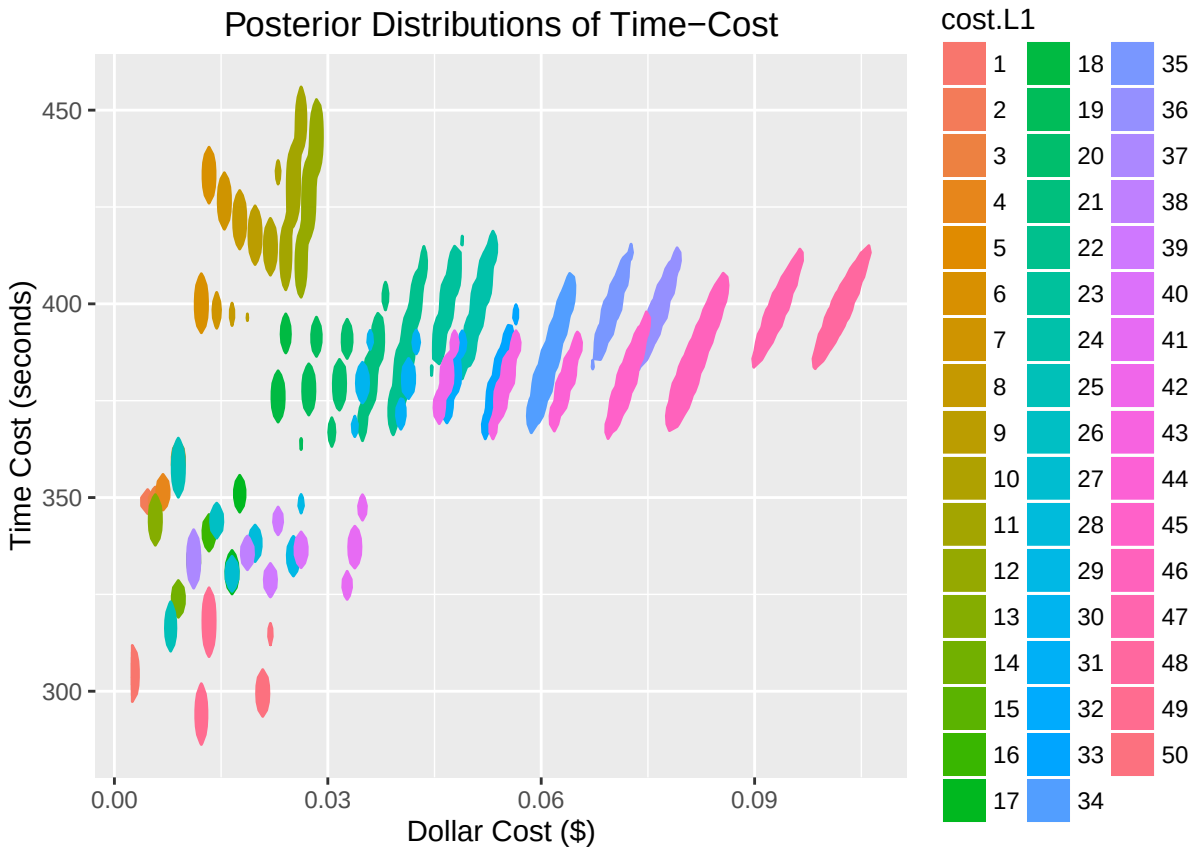
```
## bartMachine initializing with 50 trees...
## bartMachine vars checked...
## bartMachine java init...
## bartMachine factors created...
## bartMachine before preprocess...
## bartMachine after preprocess... 8 total features...
## bartMachine sigsq estimated...
## bartMachine training data finalized...
## Now building bartMachine for regression ...
## evaluating in sample data...done
## serializing in order to be saved for future R sessions...done
```






```
c.d <- melt(cost.dist)
c.d <- na.omit(c.d)
c.d$L1 <- as.factor(c.d$L1)
t.d <- melt(time.dist)
t.d <- na.omit(t.d)
t.d$L1 <- as.factor(t.d$L1)
pots <- data.frame(time = t.d, cost = c.d)
ggplot(pots) +
  # geom_point(aes(x = cost.value, y = time.value, group=cost.L1, col=cost.L1), alpha=0.25) +
  scale_fill_gradient() +
  guides(col=F, group=F) +
  stat_density2d(aes(y = time.value, x = cost.value, group=cost.L1, col=cost.L1, fill=cost.L1),
                 geom="polygon", bins=5) +
  xlab("Dollar Cost ($)") +
  ylab("Time Cost (seconds)") +
  scale_fill_discrete() +
  ggtitle("Posterior Distributions of Time-Cost")
```

```
## Scale for 'fill' is already present. Adding another scale for 'fill',
## which will replace the existing scale.
```




```
# c.d <- melt(cost.dist)
# c.d <- na.omit(c.d)
# c.d$L1 <- as.factor(c.d$L1)
# t.d <- melt(time.dist)
# t.d <- na.omit(t.d)
# t.d$L1 <- as.factor(t.d$L1)
# pots <- data.frame(time = t.d, cost = c.d)
# ggplot(pots) +
#   # geom_point(aes(x = cost.value, y = time.value, group=cost.L1, col=cost.L1), alpha=0.25) +
#   geom_density(aes(y = time.value, group=cost.L1, col=cost.L1)) +
#   geom_density(aes(x = cost.value, group=cost.L1, col-cost.L1)) +
#   scale_x_continuous()
```

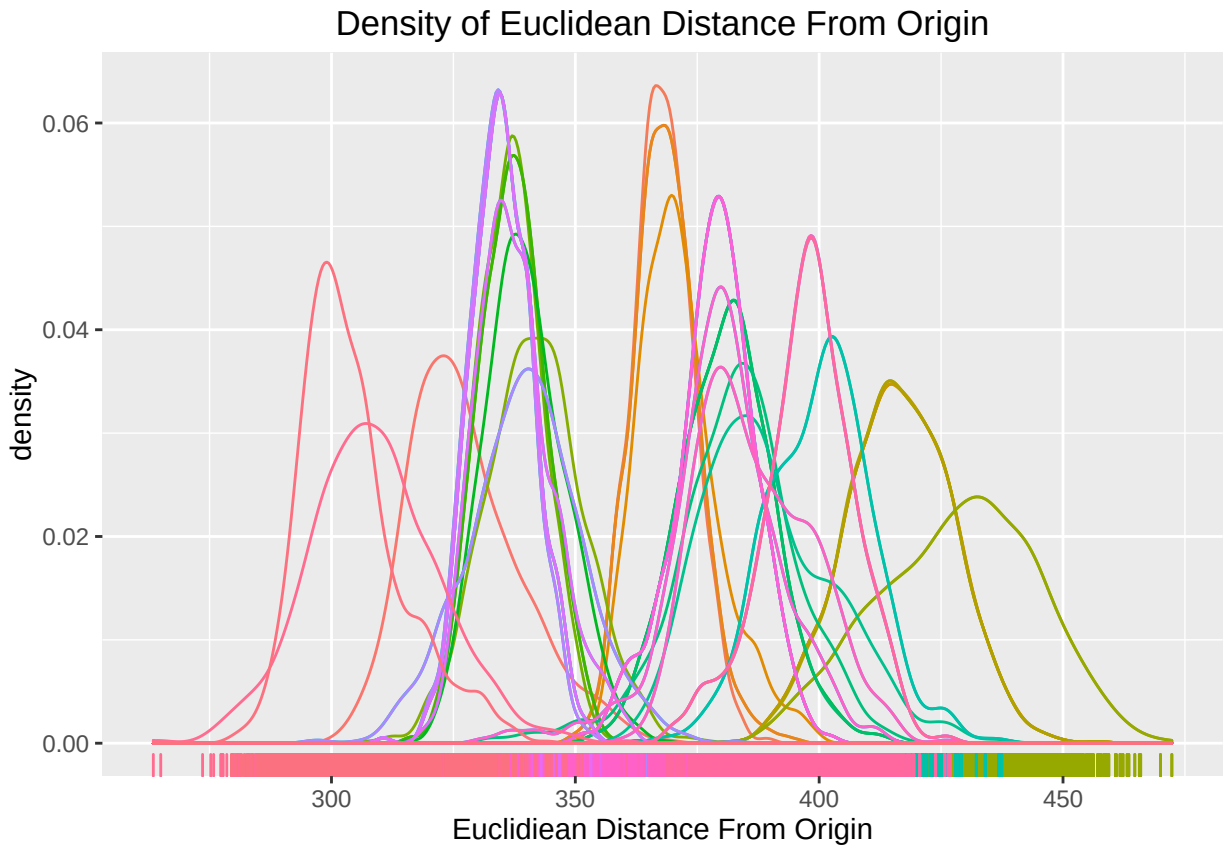



*Figure 23* is something I'm still in the middle of working through, where I use a bayesian framework to fit the execution time model, then propagate the uncertainty through the optimal prediction process. First, I fit an additive tree model (a la GBM), but using the Bayesian framework, with priors. I used the `bartMachine` package in R to do this. I computed a 1000 iteration MCMC, where each iteration was a full tree model. Then using the posterior draws, I calculated the uncertainty on the execution time prediction. Then, for each time, I calculated the corresponding dollar cost. Then, I calculate the euclidean distance between all posterior draws of execution time/dollar cost and the origin for all possible computing configurations. As before, I then take the minimum euclidean distance, but this time, with a range of possible solutions.