Project 5

Matt Wakefield

May 05, 2021

Contents

Introduction.

Modeling climate change is an extraordinarily complex endevor that require large disparate teams to contribute to the project according to their area of expertise.

This model is developed via several parameters (columns 3 through 20). Due to the complexity of this model it can crash at times. This is represented by the outcome variable and contains the values 0 for crashed or 1 for ran successfully.

The samples were gathered via a method known as latin hybercube sampling. Which samples out of a cumulative density function where each variable is it's own dimension.

Our goal will be to identify the parameters that are causing this failure, and be able to accurately predict whether or not a series of parameters will lead to a failure.

Data Preperation and Exploratory Data Analysis

```
library(tidyverse)
library(skimr )
library(corrplot)
library(factoextra)
library(ggfortify)
library(tidymodels)
library(keras)
library(vip)
```

Read file and present basic summary data.

```
df<-read.table('http://archive.ics.uci.edu/ml/machine-learning-databases/00252/pop_failures.dat

skim_without_charts(df)
```

Table 1: Data summary

Name	df
Number of rows	540
Number of columns	21
Column type frequency:	
numeric	21
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
Study	0	1	2.00	0.82	1	1.00	2.0	3.00	3
Run	0	1	90.50	52.01	1	45.75	90.5	135.25	180
vconst_corr	0	1	0.50	0.29	0	0.25	0.5	0.75	1
vconst_2	0	1	0.50	0.29	0	0.25	0.5	0.75	1
vconst_3	0	1	0.50	0.29	0	0.25	0.5	0.75	1
vconst_4	0	1	0.50	0.29	0	0.25	0.5	0.75	1
vconst_5	0	1	0.50	0.29	0	0.25	0.5	0.75	1
vconst_7	0	1	0.50	0.29	0	0.25	0.5	0.75	1
ah_corr	0	1	0.50	0.29	0	0.25	0.5	0.75	1
ah_bolus	0	1	0.50	0.29	0	0.25	0.5	0.75	1
slm_corr	0	1	0.50	0.29	0	0.25	0.5	0.75	1
efficiency_factor	0	1	0.50	0.29	0	0.25	0.5	0.75	1

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
tidal_mix_max	0	1	0.50	0.29	0	0.25	0.5	0.75	1
vertical_decay_scale	0	1	0.50	0.29	0	0.25	0.5	0.75	1
convect_corr	0	1	0.50	0.29	0	0.25	0.5	0.75	1
bckgrnd_vdc1	0	1	0.50	0.29	0	0.25	0.5	0.75	1
bckgrnd_vdc_ban	0	1	0.50	0.29	0	0.25	0.5	0.75	1
bckgrnd_vdc_eq	0	1	0.50	0.29	0	0.25	0.5	0.75	1
bckgrnd_vdc_psim	0	1	0.50	0.29	0	0.25	0.5	0.75	1
Prandtl	0	1	0.50	0.29	0	0.25	0.5	0.75	1
outcome	0	1	0.91	0.28	0	1.00	1.0	1.00	1

Firstly we observe that there are no missing values, thus there's no need for imputation.

What we can observe is that each of the variables has been normalized at a scale of 0 to 1, with each ntile corresponding the value (e.g. .25 corresponds to the 25th percentile)

The first and second variables, Study and Run, are likely not relevant to the outcome. Through EDA we can at least gauge whether or not Study is equally representative.

Most of the focus of the EDA will be on how the variables relate to one another, and if it's immediately apparent that these variables appear to be associated with failures.

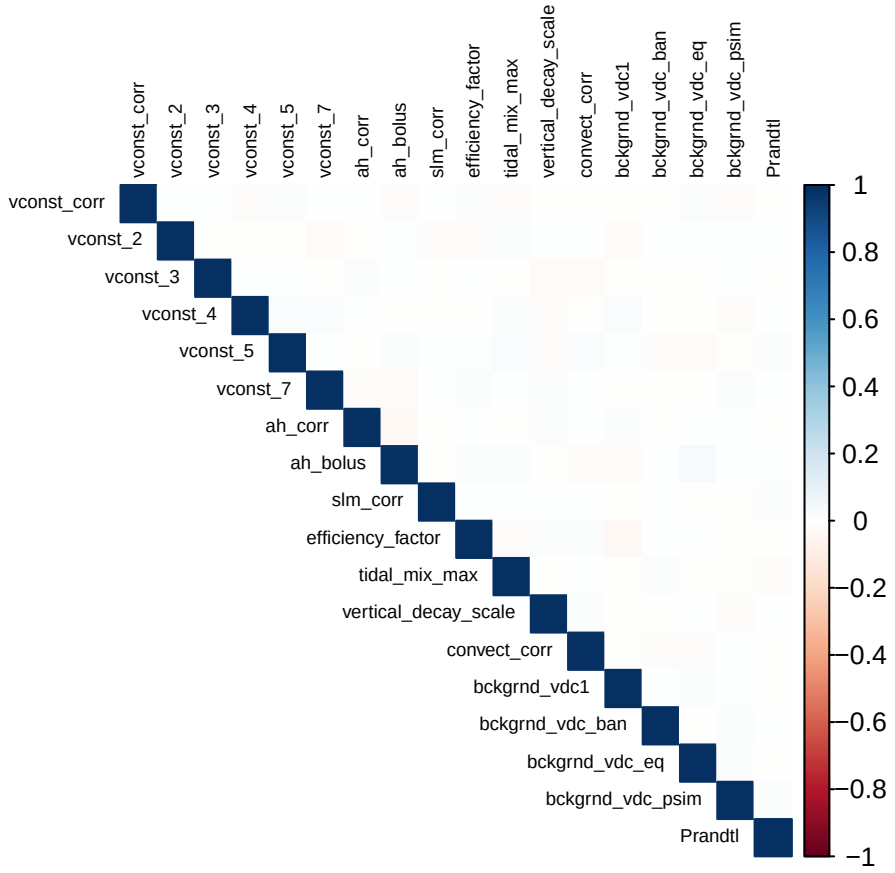
Correlation Among Variables

```
df_corr<-df%>%select(-c(Study, Run, outcome))%>%cor()
corrplot(df_corr, type="upper",method = "color", add.coef.col = 'black',tl.col = 'black',numbe
```

```
## Warning in text.default(pos.xlabel[, 1], pos.xlabel[, 2], newcolnames, srt =
## tl.srt, : "add.coef.col" is not a graphical parameter
```

```
## Warning in text.default(pos.ylabel[, 1], pos.ylabel[, 2], newrownames, col =
## tl.col, : "add.coef.col" is not a graphical parameter
```

```
## Warning in title(title, ...): "add.coef.col" is not a graphical parameter
```



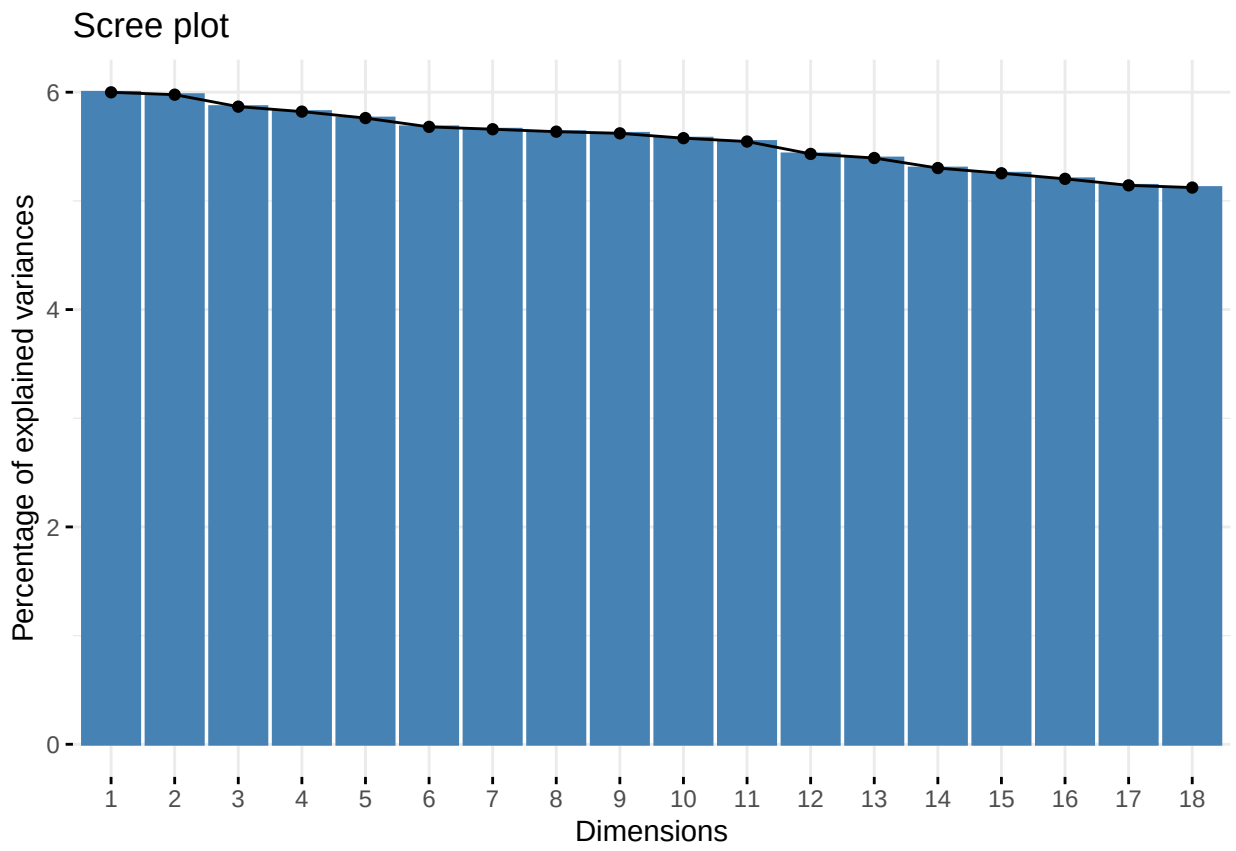

```
paste('The Average Correlation is', df_corr[df_corr !=1]%>%mean()%>%round(.,5))
```

```
## [1] "The Average Correlation is 0.00025"
```

Perhaps as a result of the normalization effort somehow leads to 0 correlation. Regardless, we can assume that our variables are about as independent as possible.

Principal Component Analysis

```
par(mar = c(0, 0, 0, 0))
df_pca<-df%>%select(-c(Study, Run, outcome))%>%as.matrix()
df_pca <- prcomp(df_pca, scale = TRUE, center = TRUE)
pca_viz<-fviz_eig(df_pca, ncp = 18)
pca_viz
```



As we can see, each additional PCA dimension consistently adds between 5 and 6 percent variance. Typically want we want to see is that one particular dimension is clearly in the lead, so we can list those variables to see what's really influencing the shape of this data.

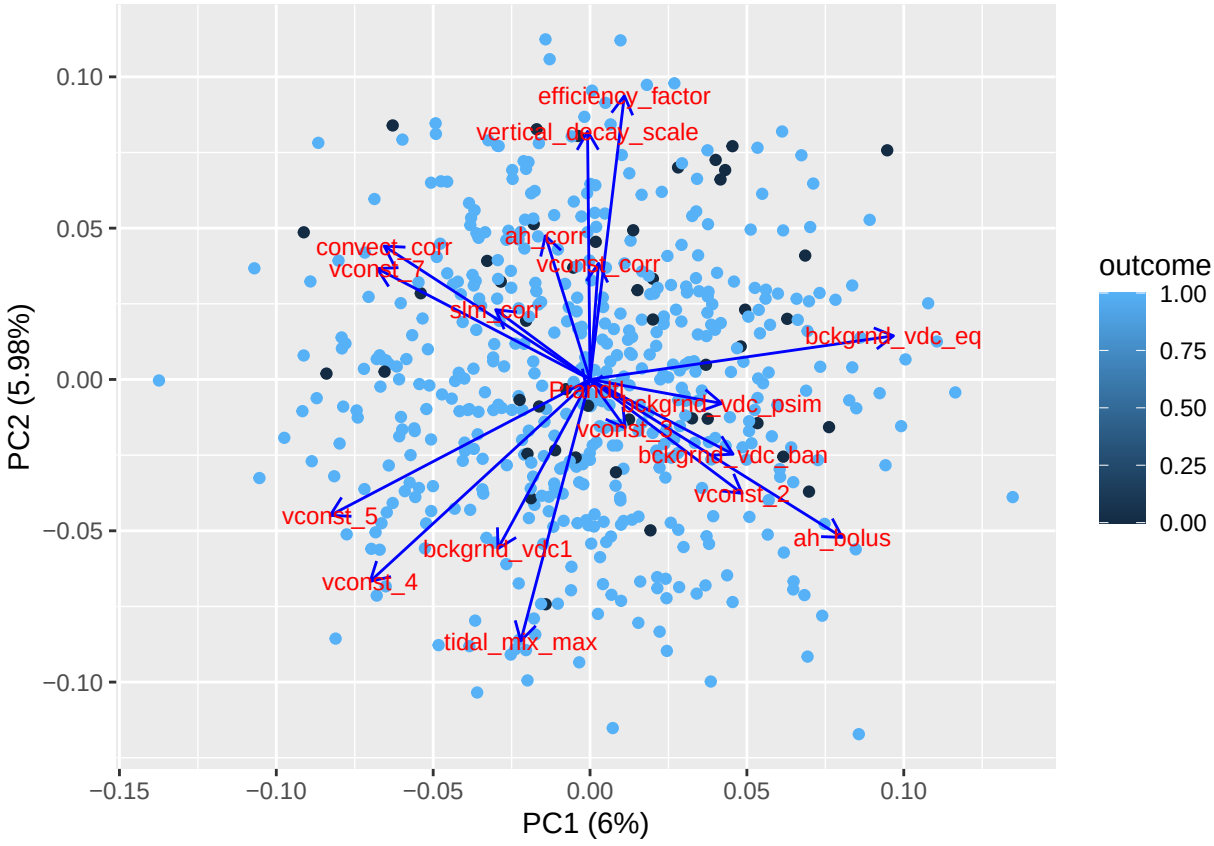
Regardless the 1st PCA is listed below:

```
df_pca$rotation[,1]%>%abs()%>%sort(., decreasing = T)%>%head(4)
```

```
## bckgrnd_vdc_eq       vconst_5       ah_bolus       vconst_4
##      0.4540322      0.3886978      0.3774293      0.3288626
```

```
autoplot(df_pca, data = df, colour = 'outcome',
         loadings = TRUE, loadings.colour = 'blue',
         loadings.label = TRUE, loadings.label.size = 3)
```

```
## Warning: `select_()` was deprecated in dplyr 0.7.0.
## Please use `select()` instead.
```



As we can see the variables are all over the place, as is the relation of those variables to the outcome. One possible trend is in the lower right hand side of the plot, that indicates ah_bolus, vconst_2, v_const3, backgrnd_vdc_ban, and backgrnd_vdc_psim all vary together somewhat. Unfortunately the scattering of outcomes is just as random as it is across the rest of the plot, and the overall contribution to variance is still very small.

The PCA does not give us a clear idea of variables we can discard yet to encourage a parsimonious model.

Means of values related to failure.

```
df_means<-df%>%select(-c(Study, Run))
df_means<-df_means%>%pivot_longer(cols = -c(outcome))%>%group_by(outcome, name)%>%summarise(mea
df_means
```

```
## # A tibble: 18 x 3
## # Groups:   outcome [1]
##    outcome name                 mean
##      <int> <chr>               <dbl>
##  1       0 vconst_corr         0.788
##  2       0 vconst_2            0.786
##  3       0 convect_corr        0.682
```

```
##  4       0 vertical_decay_scale 0.543
##  5       0 efficiency_factor    0.531
##  6       0 bckgrnd_vdc_ban      0.527
##  7       0 Prandtl              0.526
##  8       0 tidal_mix_max        0.514
##  9       0 vconst_3             0.500
## 10       0 ah_bolus             0.496
## 11       0 ah_corr              0.484
## 12       0 vconst_7             0.454
## 13       0 slm_corr             0.454
## 14       0 vconst_5             0.449
## 15       0 bckgrnd_vdc_psim     0.445
## 16       0 vconst_4             0.432
## 17       0 bckgrnd_vdc_eq       0.426
## 18       0 bckgrnd_vdc1         0.326
```

We finally have some hints about how this data may be related to failure.

Because the data has been normalized, we can compare the mean of one variable to another. Note that the mean value of vconst_corr is much higher for failures (.78) then it is bckgrnd_vdc1 (.33)

Perhaps the model that creates this variable should be looked at to see why higher values would cause a crash.

```
table(df$outcome)
```

```
##
##   0   1
##  46 494
```

Finally we can see that we are dealing with a very imbalanced target variable. With the ratio of the majority being over 10x that of the minority.

Data Partition

```
set.seed(123)

df<-df%>%select(-c(Study,Run))

df$outcome<-df$outcome%>%factor()
n <- nrow(df)
split_data <- sample(x=1:2, size = n, replace=TRUE, prob=c(0.67, 0.33))
D1 <- df[split_data == 1, ]
D2 <- df[split_data == 2, ]
y.train <- D1$outcome
yobs <- D2$outcome

data.frame('Split'=c('Train','Test'),'Observations'=c(nrow(D1), nrow(D2)))
```

```
##   Split Observations
## 1 Train          367
## 2  Test          173
```

Due to class imbalance of the target variable, let's go ahead and upsample the training set.

```
D1_upsample <- recipe( ~ ., data = D1) %>%
  step_upsample(outcome, over_ratio = (1/3))%>%
  prep(training = D1)
```

```
## Warning: `step_upsample()` was deprecated in recipes 0.1.13.
## Please use `themis::step_upsample()` instead.
```

```
D1 <- bake(D1_upsample, new_data = NULL)

table(D1$outcome)
```

```
##
##   0   1
## 112 336
```

Models

Logistic Regression

```
log_mod<-logistic_reg(penalty = tune(), mixture = tune())%>%
  set_engine('glmnet')%>%
  set_mode('classification')

log_grid<-grid_regular(penalty(),mixture(),levels = 20)

D1_folds<-vfold_cv(D1)

log_wf<-workflow()%>%
  add_model(log_mod)%>%
  add_formula(outcome ~ .)

log_res<-log_wf%>%tune_grid(resamples = D1_folds,
                            grid = log_grid)
```

```
## Warning: package 'rlang' was built under R version 4.0.5
```

```
## Warning: package 'vctrs' was built under R version 4.0.5
```

```
## Warning: package 'glmnet' was built under R version 4.0.4
```

```
log_best<-log_res%>%select_best("roc_auc")

final_log_wf<-log_wf%>%finalize_workflow(log_best)

final_log<-final_log_wf%>%fit(data = D1)

final_log$fit$fit$spec
```

```
## Logistic Regression Model Specification (classification)
##
## Main Arguments:
##   penalty = 0.00233572146909012
##   mixture = 0.684210526315789
##
## Computational engine: glmnet
##
## Model fit template:
## glmnet::glmnet(x = missing_arg(), y = missing_arg(), weights = missing_arg(),
##     alpha = 0.684210526315789, family = "binomial")
```

```
 final_log%>%
  pull_workflow_fit() %>%
  tidy()
```

```
## # A tibble: 19 x 3
##    term                 estimate penalty
##    <chr>                   <dbl>   <dbl>
##  1 (Intercept)           21.4    0.00234
##  2 vconst_corr          -13.0    0.00234
##  3 vconst_2             -11.8    0.00234
##  4 vconst_3              -1.45   0.00234
##  5 vconst_4              -0.204  0.00234
##  6 vconst_5               1.03   0.00234
##  7 vconst_7               0      0.00234
##  8 ah_corr                0.479  0.00234
##  9 ah_bolus              -2.39   0.00234
## 10 slm_corr              -0.0961 0.00234
## 11 efficiency_factor     -1.67   0.00234
## 12 tidal_mix_max         -1.61   0.00234
## 13 vertical_decay_scale  -1.34   0.00234
## 14 convect_corr          -5.72   0.00234
## 15 bckgrnd_vdc1           5.88   0.00234
## 16 bckgrnd_vdc_ban        1.22   0.00234
## 17 bckgrnd_vdc_eq         1.17   0.00234
```

```
## 18 bckgrnd_vdc_psim       1.13   0.00234
## 19 Prandtl                0.410  0.00234
```

```
prediction_metrics_log<-data.frame(D2$outcome,predict(final_log,D2),predict(final_log,D2, type

metrics_log<-metrics(prediction_metrics_log, D2.outcome,.pred_class, .pred_0)

metrics<-data.frame(Model_Type='Logistic_Model',metrics_log)
```

Here we are using an elastic net that is closer to ridge regression then it is lasso regression. These parameters were chosen via a grid search of 20 samples using 10 fold cross validation.

```
final_log%>%
    pull_workflow_fit() %>%
    tidy()%>%arrange(desc(abs(estimate)))
```

```
## # A tibble: 19 x 3
##    term                 estimate penalty
##    <chr>                   <dbl>   <dbl>
##  1 (Intercept)           21.4    0.00234
##  2 vconst_corr          -13.0    0.00234
##  3 vconst_2             -11.8    0.00234
##  4 bckgrnd_vdc1           5.88   0.00234
##  5 convect_corr          -5.72   0.00234
##  6 ah_bolus              -2.39   0.00234
##  7 efficiency_factor     -1.67   0.00234
##  8 tidal_mix_max         -1.61   0.00234
##  9 vconst_3              -1.45   0.00234
## 10 vertical_decay_scale  -1.34   0.00234
## 11 bckgrnd_vdc_ban        1.22   0.00234
## 12 bckgrnd_vdc_eq         1.17   0.00234
## 13 bckgrnd_vdc_psim       1.13   0.00234
## 14 vconst_5               1.03   0.00234
## 15 ah_corr                0.479  0.00234
## 16 Prandtl                0.410  0.00234
## 17 vconst_4              -0.204  0.00234
## 18 slm_corr              -0.0961 0.00234
## 19 vconst_7               0      0.00234
```

Above indicates the magnitude of the coefficients. Because all variables were normalized beforehand, we can safely assume that the largest absolute value of the coefficient is a pretty good indicator of variable importance.

Random Forest

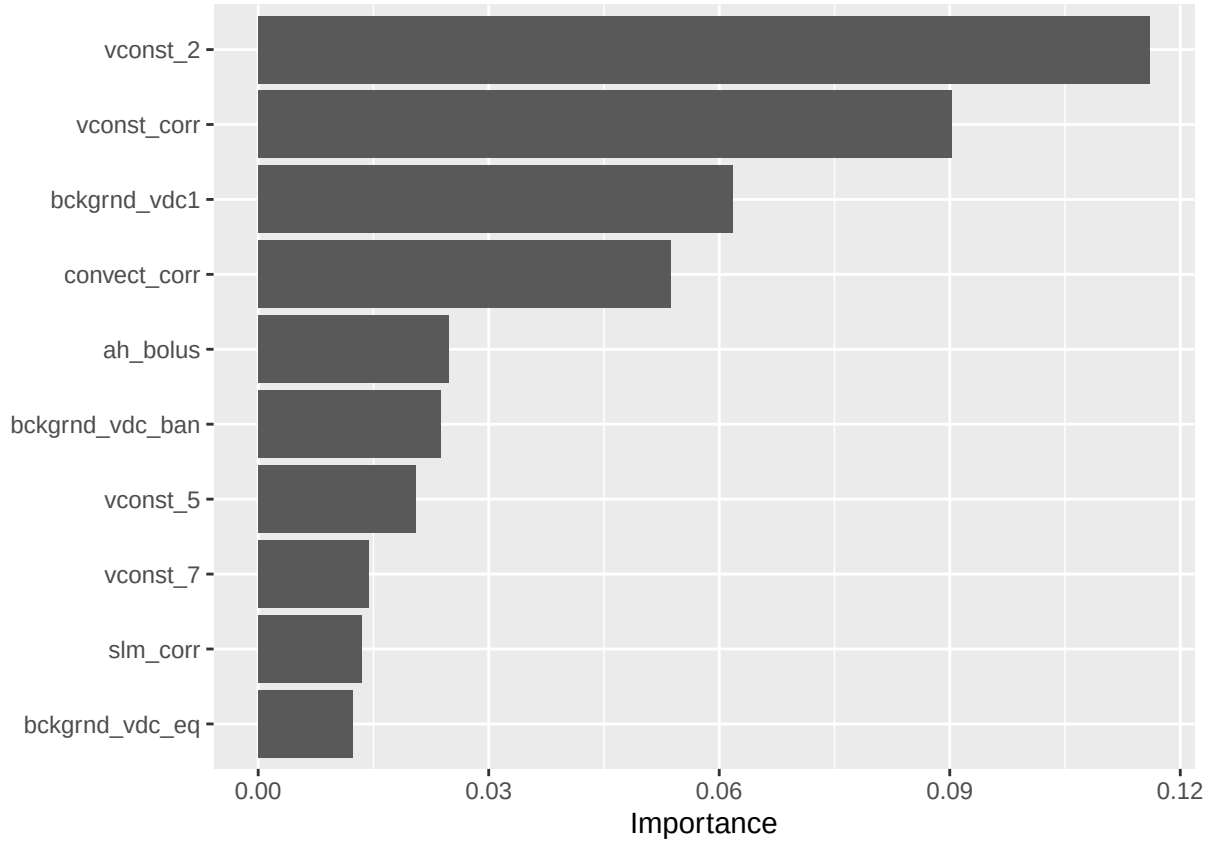
```
rf_mod<-rand_forest(mtry =tune(),trees = tune(), min_n = tune())%>%
  set_engine('ranger',importance = "permutation")%>%
#%>%
  set_mode('classification')


rf_wf<-workflow()%>%
  add_model(rf_mod)%>%
  add_formula(outcome ~ .)

#rf_res<-rf_wf%>%tune_grid(resamples = D1_folds,
#                                grid = rf_grid)

rf_res<-rf_wf%>%tune_grid(resamples = D1_folds,
                                grid = 20)
```

```
## i Creating pre-processing data to finalize unknown parameter: mtry

## Warning: package 'ranger' was built under R version 4.0.5
```

```
rf_best<-rf_res%>%select_best("roc_auc")

final_rf_wf<-rf_wf%>%finalize_workflow(rf_best)

final_rf<-final_rf_wf%>%fit(data = D1)
```

Similar to our log model process, we run a grid pattern of 20 values for trees and min_n.

```
final_rf%>%
    pull_workflow_fit() %>%
    vip(geom = "col")
```




```
prediction_metrics_rf<-data.frame(D2$outcome,predict(final_rf,D2),predict(final_rf,D2, type = '

metrics_rf<-metrics(prediction_metrics_rf, D2.outcome,.pred_class, .pred_0)

metrics<-rbind(metrics,data.frame(Model_Type='Random_Forest',metrics_rf))
```

As we can see, the importance of the factors is similar to the log model as well with vconst_2 and vconst_corr pulling well ahead of the others.

Neural Nets

```
D1_x<-D1%>%select(-outcome)%>%as.matrix()
D1_y<-D1%>%select(outcome)%>%as.matrix()%>%to_categorical()

D2_x<-D2%>%select(-outcome)%>%as.matrix()
D2_y<-D2%>%select(outcome)%>%as.matrix()%>%to_categorical()

nn_mod<-mlp(hidden_units = c(4,2))%>%
  set_engine('nnet')%>%
  set_mode('classification')
```

```
nn_mod2 <- keras_model_sequential() %>%
  layer_dense(units = 10, activation = "relu", input_shape = 18) %>%
  layer_dense(units = 30, activation = "relu") %>%
  layer_dense(units = 2, activation = "sigmoid")%>%
  compile(
    loss = 'categorical_crossentropy',
    optimizer = optimizer_rmsprop(),
    metrics = c('accuracy')
  )


nn_mod4 <- keras_model_sequential() %>%
  layer_dense(units = 10, activation = "relu", input_shape = 18) %>%
  layer_dense(units = 30, activation = "relu") %>%
    layer_dense(units = 75, activation = "relu") %>%
    layer_dense(units = 40, activation = "relu") %>%
  layer_dense(units = 2, activation = "sigmoid")%>%
  compile(
    loss = 'categorical_crossentropy',
    optimizer = optimizer_rmsprop(),
    metrics = c('accuracy')
  )

nn_mod5 <- keras_model_sequential() %>%
  layer_dense(units = 10, activation = "relu", input_shape = 18) %>%
  layer_dense(units = 120, activation = "relu") %>%
    layer_dense(units = 75, activation = "relu") %>%
    layer_dense(units = 100, activation = "relu") %>%
      layer_dense(units = 40, activation = "relu") %>%
  layer_dense(units = 2, activation = "sigmoid")%>%
  compile(
    loss = 'categorical_crossentropy',
    optimizer = optimizer_rmsprop(),
    metrics = c('accuracy')
  )


nn_fi2 <- nn_mod2 %>%
  fit(
    D1_x,
    D1_y,
    epochs = 500,
    validation_split = 0.2,
    verbose = FALSE
  )

nn_fit4 <- nn_mod4 %>%
```

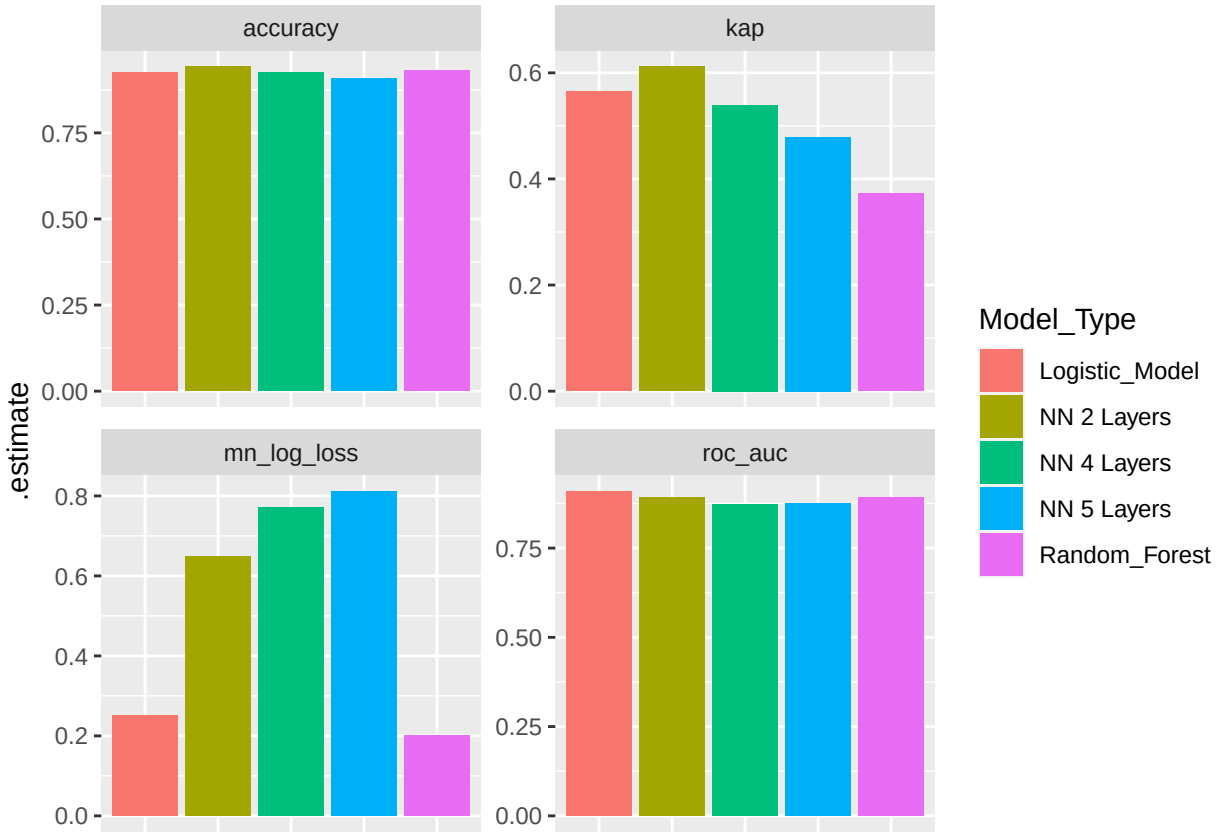
```
  fit(
    D1_x,
    D1_y,
    epochs = 500,
    validation_split = 0.2,
    verbose = FALSE
  )

nn_fit5 <- nn_mod5 %>%
  fit(
    D1_x,
    D1_y,
    epochs = 500,
    validation_split = 0.2,
    verbose = FALSE
  )



nn2_fit <- predict_classes(object = nn_mod2, x = D2_x)%>%factor()
nn2_fit_p <- predict_proba(object = nn_mod2, x = D2_x)
nn2_fits<-data.frame(outcome = D2$outcome, .pred_class=nn2_fit,.pred_0 = nn2_fit_p[,1])
metrics_nn2<-metrics(nn2_fits, outcome,.pred_class, .pred_0)


nn4_fit <- predict_classes(object = nn_mod4, x = D2_x)%>%factor()
nn4_fit_p <- predict_proba(object = nn_mod4, x = D2_x)
nn4_fits<-data.frame(outcome = D2$outcome, .pred_class=nn4_fit,.pred_0 = nn4_fit_p[,1])
metrics_nn4<-metrics(nn4_fits, outcome,.pred_class, .pred_0)


nn5_fit <- predict_classes(object = nn_mod5, x = D2_x)%>%factor()
nn5_fit_p <- predict_proba(object = nn_mod5, x = D2_x)
nn5_fits<-data.frame(outcome = D2$outcome, .pred_class=nn5_fit,.pred_0 = nn5_fit_p[,1])
metrics_nn5<-metrics(nn5_fits, outcome,.pred_class, .pred_0)


metrics<-rbind(metrics,data.frame(Model_Type='NN 2 Layers',metrics_nn2))
metrics<-rbind(metrics,data.frame(Model_Type='NN 4 Layers',metrics_nn4))
metrics<-rbind(metrics,data.frame(Model_Type='NN 5 Layers',metrics_nn5))
```

A variety of layers and node counts were used in the creation of the three models. One thing that wasn't anticipated, is that the training time for these models was much quicker then that of the other models.

This may be because we didn't tune our hyperparameters here via a grid search like we did for

Model Comparison

```
ggplot(data = metrics, aes(x = Model_Type, y = .estimate, fill = Model_Type))+
  geom_col()+
  facet_wrap(.~.metric, scales = 'free')+
  theme(axis.title.x=element_blank(),
        axis.text.x=element_blank(),
        axis.ticks.x=element_blank())
```



As we can see, the two main metrics, accuracy and auc don't vary greatly, but they do vary. Interestingly the logistic and the random forest models performed the best, with all three neural nets performing somewhat worse.

This is likely due to the massive amount of options that are available when tuning a neural net. Looking at efficient parameter tuning (such as the methods used for the other two models) may be a worthwhile effort.

Another possibility is that the upsampling led to decreased accuracy only because the test data set was so imbalanced. An analysis of precision vs recall could be worthwhile. Perhaps we are willing to sacrifice some misclassification of the majority class if it means we more accurately predict the minority class.