STAT 325 Model 1 SVM

THEDION V. DIAM JR.

2022-12-14

Load Data Sets

The data contains 197 rows and 431 columns with *Failure.binary* binary output.

```
rawd <- read.csv("C:/Users/redee/OneDrive/Desktop/STAT 325 Final Project/FP DATA.csv")
```

#========== Reprocessing the Raw Data ==========#

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
```

```
## Warning: package 'ggplot2' was built under R version 4.2.2
```

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

```
library(bestNormalize)
```

```
## Warning: package 'bestNormalize' was built under R version 4.2.2
```

Check for null and missing values

Using *anyNA()* function, We can determine if any missing values in our data.

```
anyNA(rawd)
```

```
## [1] FALSE
```

```
#The result shows either *True* or *False*. If True, omit the missing values using *na.omit()*

#[1] FALSE

#Thus, our data has no missing values.
```

Check for Normality of the Data

We used *Shapiro-Wilk's Test* to check the normality of the data.

```
rd <- rawd%>%select_if(is.numeric)
rd <- rd[,-1]
test <- apply(rd,2,function(x){shapiro.test(x)})
```

To have the list of p-value of all variables, the *unlist()* function is used and convert a list to vector.

```
pvalue_list <- unlist(lapply(test, function(x) x$p.value))
```

```
sum(pvalue_list<0.05)  # not normally distributed
```

```
## [1] 428
```

```
sum(pvalue_list>0.05)  # normally distributed
```

```
## [1] 1
```

```
test$Entropy_cooc.W.ADC
```

```
##
##  Shapiro-Wilk normality test
##
## data:  x
## W = 0.98903, p-value = 0.135
```

```
# [1] 428
# [1] 1

#  Thus, we have 428 variables that are not normally distributed and Entropy_cooc.W.ADC is normally dis
```

We use *orderNorm()* function, the *x.t* is the elements of orderNorm() function transformed original data.Using the *Shapiro-Wilk's Test*

```
TRDrawd=rawd[,c(3,5:length(names(rawd)))]

TRDrawd=apply(TRDrawd,2,orderNorm)
TRDrawd=lapply(TRDrawd, function(x) x$x.t)
TRDrawd=TRDrawd%>%as.data.frame()
test=apply(TRDrawd,2,shapiro.test)
test=unlist(lapply(test, function(x) x$p.value))
```

#Testing Data

```
sum(test <0.05)  # not normally distributed
```

```
## [1] 0
```

```
sum(test >0.05)  # normally distributed
```

```
## [1] 428
```

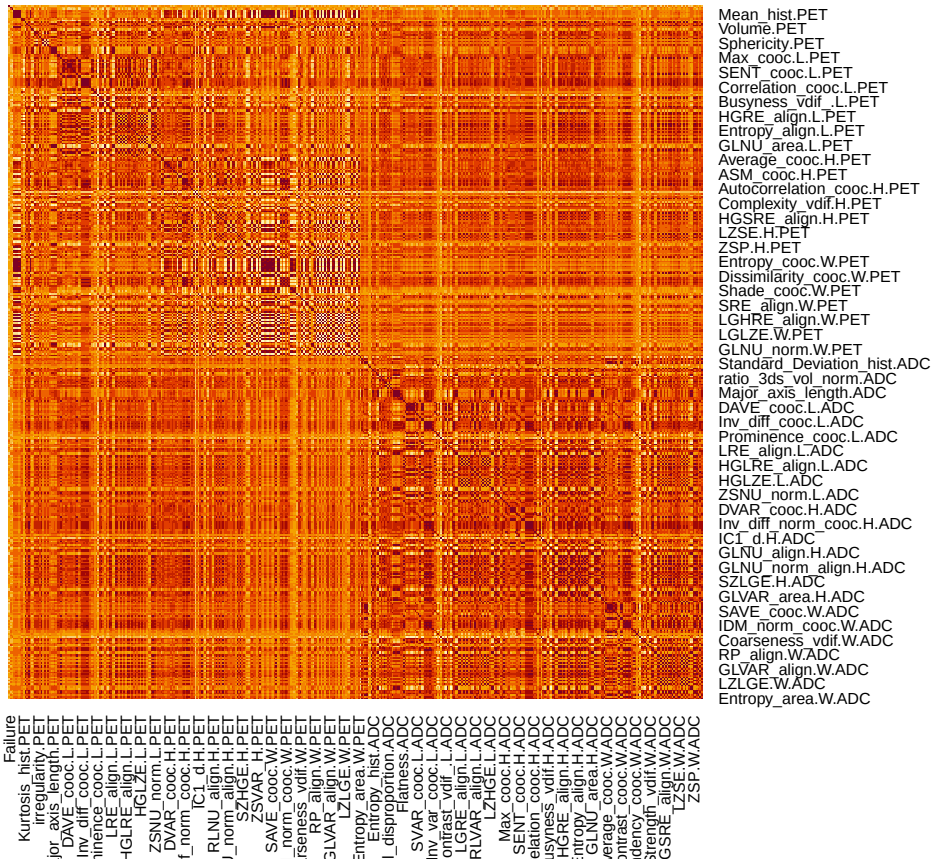
```
#[1] 0
#[1] 428

# Thus, our data is normally distributed.
```

```
rawd[,c(3,5:length(names(rawd)))]=TRDrawd
```

Get the correlation of the whole data expect the categorical variables

```
CorMatrix=cor(rawd[,-c(1,2)])
heatmap(CorMatrix,Rowv=NA,Colv=NA,scale="none",revC = T)
```



#Splitting the Data Split the data into training (80%) and testing (20%).

```
rawd$Institution=as.factor(rawd$Institution)
rawd$Failure.binary=as.factor(rawd$Failure.binary)
```

```
splitter <- sample(1:nrow(rawd), round(nrow(rawd) * 0.8))
trainND <- rawd[splitter, ]
testND  <- rawd[-splitter, ]
```

The data frame output of data reprocessing will be converted into to "csv", which will be used for entire project.

```
# Helper packages
library(dplyr)    # for data wrangling
library(ggplot2)  # for awesome graphics
library(rsample)  # for data splitting

# Modeling packages
library(caret)    # for classification and regression training
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
```

```
##
##     lift
```

```
library(kernlab)  # for fitting SVMs
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:purrr':
##
##     cross
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
library(modeldata) #for Failure.binary data
library(forcats)

# Model interpretability packages
library(pdp)      # for partial dependence plots, etc.
```

```
##
## Attaching package: 'pdp'
```

```
## The following object is masked from 'package:purrr':
##
##     partial
```

```
library(vip)      # for variable importance plots
```

```
##
## Attaching package: 'vip'
```

```
## The following object is masked from 'package:utils':
##
##     vi
```

```
# DATA
final<- read.csv("C:/Users/redee/OneDrive/Desktop/STAT 325 Final Project/newdat.csv")
View(final)
```

#========== SVM ==========

Support vector machines (SVMs) offer a direct approach to binary classification. The popular kernel function used by SVMs are Linear `"svmLinear"`, Polynomial Kernel `"svmPoly"` and Radial basis kernel `"svmRadial"`

```
# Load Failure.binary data

final$Failure.binary=as.factor(final$Failure.binary)

# Create training (70%) and test (30%) sets
set.seed(123)  # for reproducibility
churn_split <- initial_split(final, prop = 0.8, strata = "Failure.binary")
split_train <- training(churn_split)
split_test  <- testing(churn_split)
```

```
# Linear (i.e., soft margin classifier)
caret::getModelInfo("svmLinear")$svmLinear$parameters
```

```
##   parameter   class label
```

```
## 1         C numeric  Cost
```

```
# Polynomial kernel
caret::getModelInfo("svmPoly")$svmPoly$parameters
```

```
##   parameter   class             label
## 1    degree numeric Polynomial Degree
## 2     scale numeric             Scale
## 3         C numeric              Cost
```

```
# Radial basis kernel
caret::getModelInfo("svmRadial")$svmRadial$parameters
```

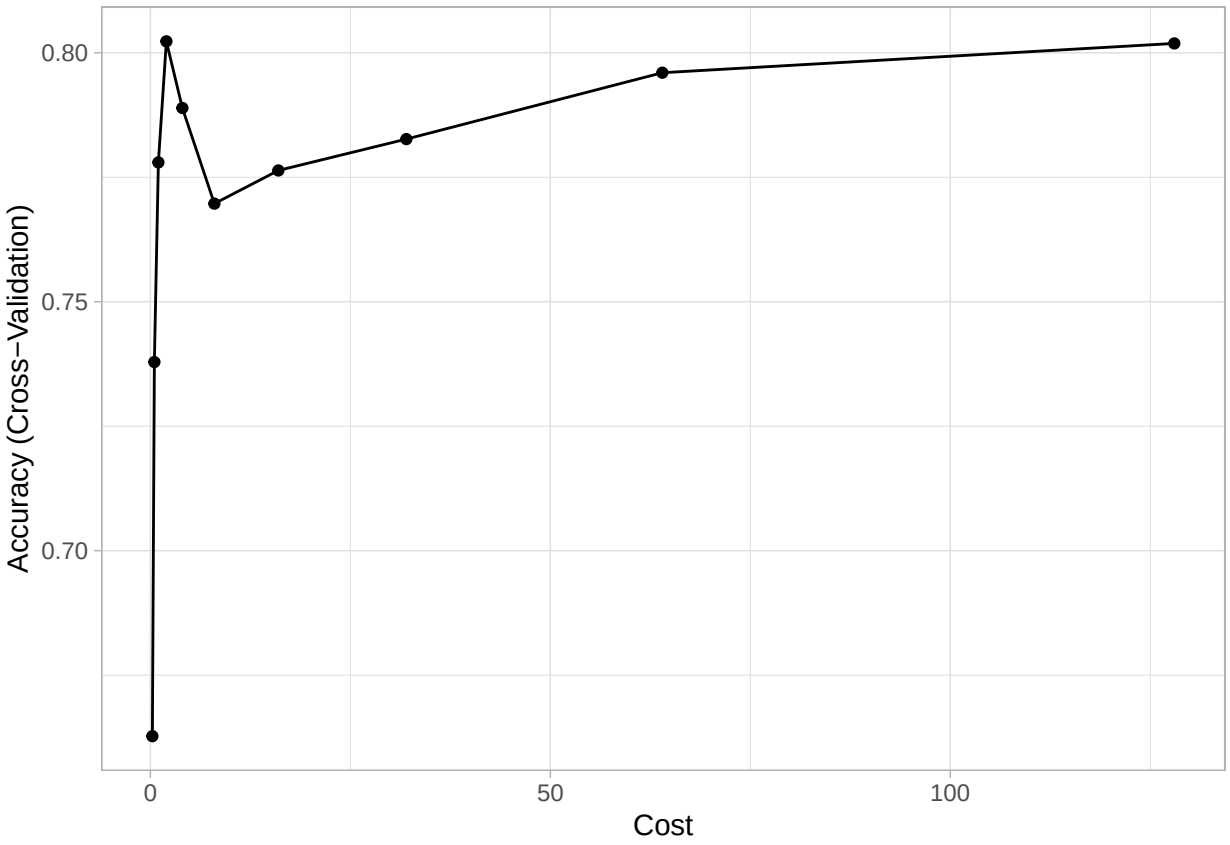
```
##   parameter   class label
## 1     sigma numeric Sigma
## 2         C numeric  Cost
```

Run SVM Model in Training phase

```
set.seed(1854)  # for reproducibility
split_svm <- train(
  Failure.binary ~ .,
  data = split_train,
  method = "svmRadial",
  preProcess = c("center", "scale"),
  trControl = trainControl(method = "cv", number = 10),
  tuneLength = 10
)
```

Plot and print SVM model with with radial basis kernel.

```
# Plot results
ggplot(split_svm) + theme_light()
```




```
# Print results
split_svm$results
```

```
##          sigma      C  Accuracy     Kappa AccuracySD   KappaSD
## 1  0.001998749   0.25 0.6627451 0.0000000 0.01891300 0.0000000
## 2  0.001998749   0.50 0.7378922 0.2715440 0.06418046 0.2198366
## 3  0.001998749   1.00 0.7779902 0.4565954 0.07142465 0.1608304
## 4  0.001998749   2.00 0.8023039 0.5196491 0.09057479 0.2186000
## 5  0.001998749   4.00 0.7889216 0.5030643 0.07639949 0.1942976
## 6  0.001998749   8.00 0.7697059 0.4653629 0.07092559 0.1830668
## 7  0.001998749  16.00 0.7763725 0.4861127 0.06283611 0.1498343
## 8  0.001998749  32.00 0.7826716 0.4985015 0.07602914 0.1806382
## 9  0.001998749  64.00 0.7960049 0.5248585 0.07147503 0.1670975
## 10 0.001998749 128.00 0.8018873 0.5429164 0.08701199 0.2010434
```

Control parameter

```
class.weights = c("No" = 1, "Yes" = 10)

# Control params for SVM
ctrl <- trainControl(
  method = "cv",
  number = 10,
  classProbs = TRUE,
  summaryFunction = twoClassSummary  # also needed for AUC/ROC
)
```

```
split_train$Failure.binary=fct_recode(split_train$Failure.binary,No="0",Yes="1")
```

Print the AUC values during Training

```
# Tune an SVM
set.seed(5628)  # for reproducibility
train_svm_auc <- train(
  Failure.binary ~ .,
  data = split_train,
  method = "svmRadial",
  preProcess = c("center", "scale"),
  metric = "ROC",  # area under ROC curve (AUC)
  trControl = ctrl,
  tuneLength = 10
)

# Print results
train_svm_auc$results
```

```
##          sigma      C       ROC      Sens      Spec      ROCSD     SensSD
## 1  0.001697891   0.25 0.8102727 0.8445455 0.5033333 0.09982583 0.12592723
## 2  0.001697891   0.50 0.8102727 0.8536364 0.5033333 0.09982583 0.12708861
## 3  0.001697891   1.00 0.8323939 0.8827273 0.5233333 0.09919217 0.11244425
## 4  0.001697891   2.00 0.8520606 0.9036364 0.6033333 0.09942461 0.09988055
## 5  0.001697891   4.00 0.8582121 0.9236364 0.6366667 0.09545946 0.09679909
## 6  0.001697891   8.00 0.8729697 0.9427273 0.5766667 0.11486557 0.06542227
## 7  0.001697891  16.00 0.8901818 0.9327273 0.6366667 0.13222606 0.07892762
## 8  0.001697891  32.00 0.8830000 0.9418182 0.5933333 0.13402578 0.06886193
## 9  0.001697891  64.00 0.8812121 0.9418182 0.6133333 0.15158268 0.05019704
## 10 0.001697891 128.00 0.8659697 0.9236364 0.6133333 0.15790577 0.08454491
##       SpecSD
## 1  0.2224721
## 2  0.2224721
## 3  0.2403958
## 4  0.2157101
## 5  0.2235792
## 6  0.1937607
## 7  0.2027283
## 8  0.2968144
## 9  0.2563755
## 10 0.3182514
```

```
confusionMatrix(train_svm_auc)
```

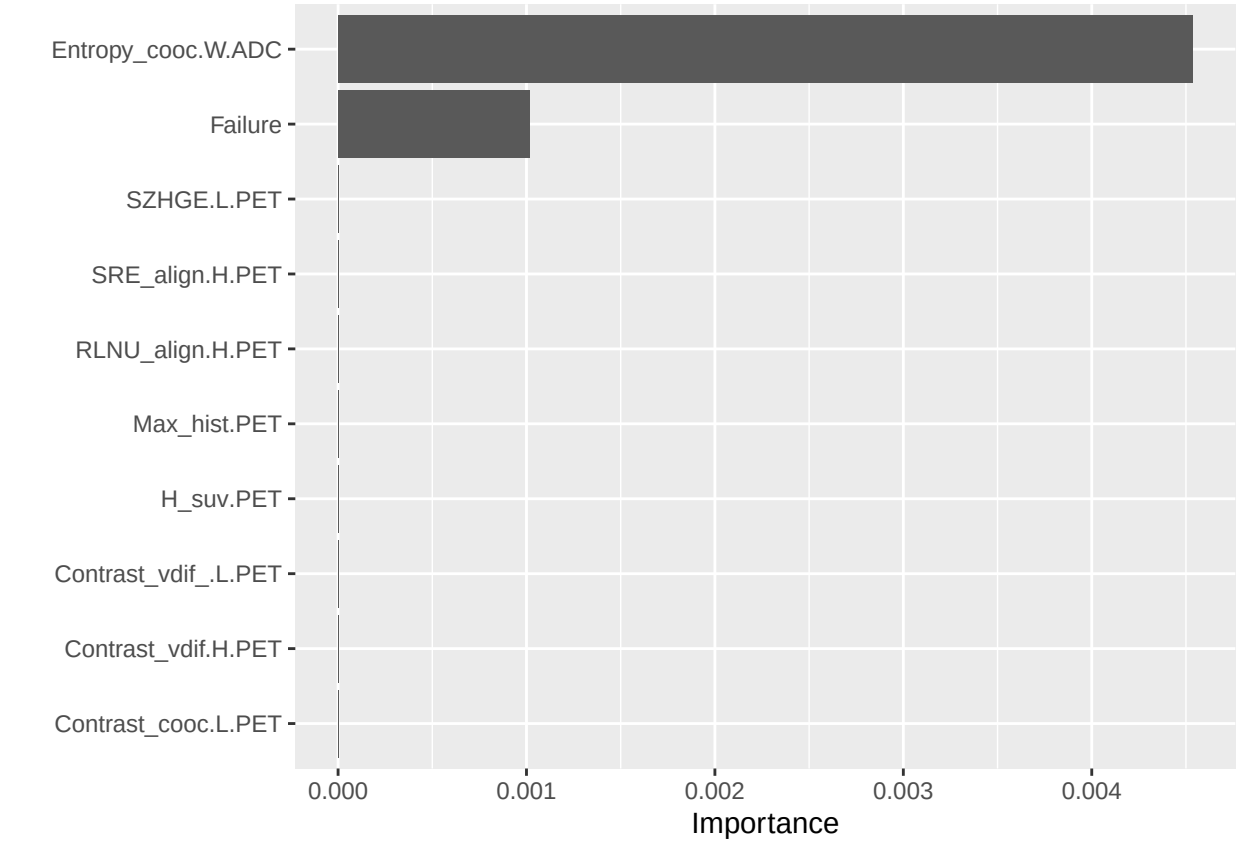
```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction   No  Yes
##        No  61.8 12.1
##        Yes  4.5 21.7
##
```

```
##  Accuracy (average) : 0.8344
```

Print the Top 20 important features during Training

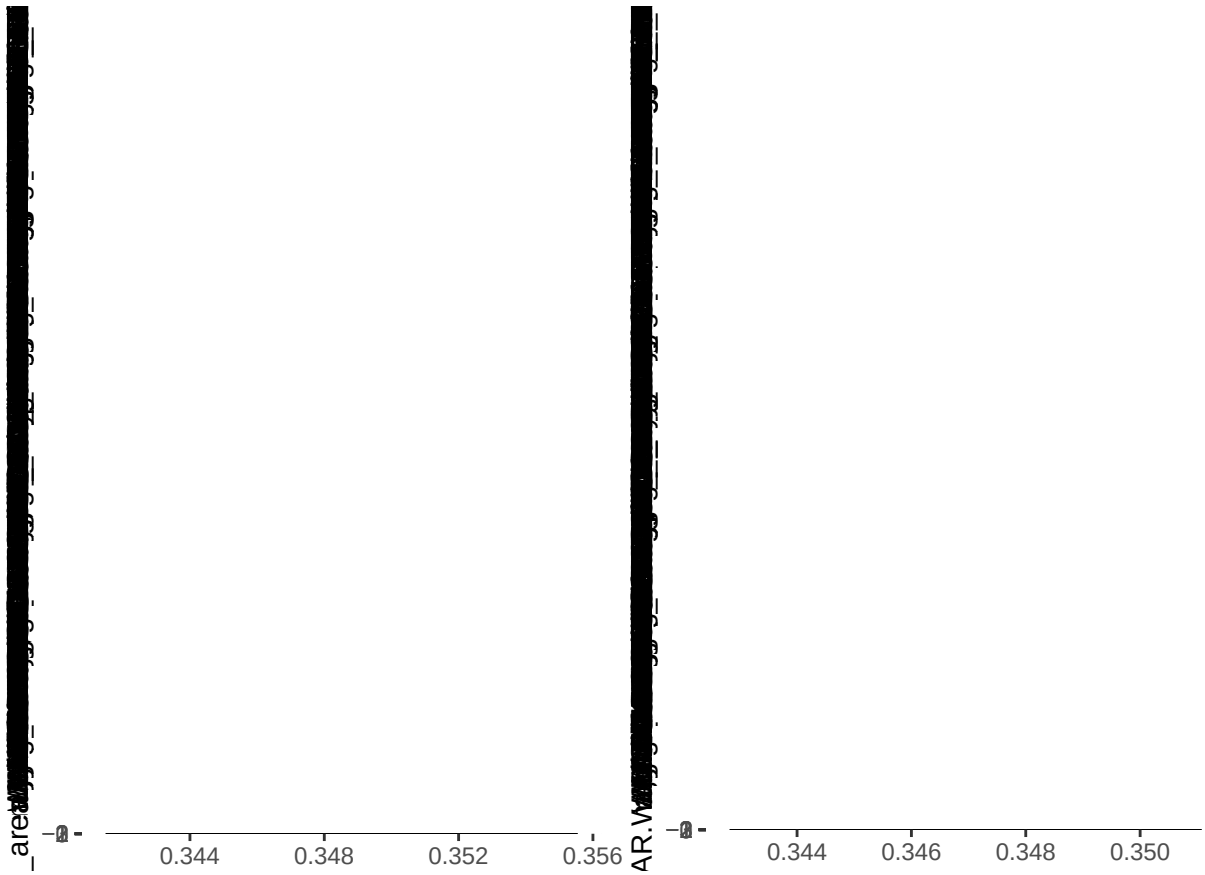
```
prob_yes <- function(object, newdata) {
  predict(object, newdata = newdata, type = "prob")[, "Yes"]
}

# Variable importance plot
set.seed(2827)  # for reproducibility
vip(train_svm_auc, method = "permute", nsim = 5, train = split_train,
    target = "Failure.binary", metric = "auc", reference_class = "Yes",
    pred_wrapper = prob_yes)
```




```
features <- setdiff(names(final), names(final)[c(1,2)])
pdps <- lapply(features, function(x) {
  partial(train_svm_auc, pred.var = x, which.class = 2,
          prob = TRUE, plot = TRUE, plot.engine = "ggplot2") +
    coord_flip()
})

grid.arrange(grobs = pdps,  ncol = 2)
```



Print the AUC values during Testing

```
split_test$Failure.binary=fct_recode(split_test$Failure.binary,No="0",Yes="1")

# Tune an SVM with radial
set.seed(5628)  # for reproducibility
test_svm_auc <- train(
  Failure.binary ~ .,
  data = split_test,
  method = "svmRadial",
  preProcess = c("center", "scale"),
  metric = "ROC",  # area under ROC curve (AUC)
  trControl = ctrl,
  tuneLength = 10
)

# Print results
test_svm_auc$results
```

```
##          sigma      C       ROC      Sens Spec     ROCSD    SensSD SpecSD
## 1  0.001959001   0.25 0.6750000 0.9666667    0 0.2872013 0.1054093      0
## 2  0.001959001   0.50 0.5750000 0.9333333    0 0.3320577 0.1405457      0
## 3  0.001959001   1.00 0.6250000 1.0000000    0 0.3148829 0.0000000      0
## 4  0.001959001   2.00 0.3083333 0.9000000    0 0.3168372 0.2249829      0
## 5  0.001959001   4.00 0.3500000 0.9000000    0 0.4021547 0.2249829      0
```

```
## 6  0.001959001   8.00 0.3916667 0.9000000    0 0.3889881 0.2249829      0
## 7  0.001959001  16.00 0.3083333 0.9000000    0 0.3514740 0.2249829      0
## 8  0.001959001  32.00 0.4250000 0.8333333    0 0.3976202 0.2832789      0
## 9  0.001959001  64.00 0.3750000 0.9333333    0 0.3833937 0.1405457      0
## 10 0.001959001 128.00 0.4083333 0.8666667    0 0.3937200 0.2810913      0
```

```
confusionMatrix(test_svm_auc)
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction   No  Yes
##        No  62.5 35.0
##        Yes  2.5  0.0
##
##  Accuracy (average) : 0.625
```