Kernel and Ensemble Methods: Classification

Isabelle Kirby, Bridgette Bryant

SVM Classification for Korea's top high elo teams in League of Legends

I utilized SVM linear, polynomial, and radial kernels to classify the League of Legends team won or lost based on the total team stats for kills, deaths, gold, and vision scores. Our dataset has all these values in two datasets lose_team_stats.csv and win_team_stats.csv. Each game also has a gameId which is unique to that specific game. This ID is identical in each dataset. The win_team_stats.csv file contains the winning teams kills, deaths, gold earned, damage dealt, crowd control, and vision scores for each player. The lose_team_stats is the same but for the losing teams. These datasets are great because they contain nearly 100k games and have no NA/Null values.

Cleaning Data

First we have to unzip the archive (they are large data sets nearly 100k games). Then we will save the

```
win_stats_df <- read.csv((unz("League_Data/league_korea_high_elo_team_stats.zip", "win_team_stats.csv"))
lose_stats_df <- read.csv((unz("League_Data/league_korea_high_elo_team_stats.zip", "lose_team_stats.csv

str(win_stats_df)
```

```
## 'data.frame':    90500 obs. of  31 variables:
##  $ win_kills1                      : num  10 3 7 11 0 7 9 8 18 4 ...
##  $ win_kills2                      : num  4 7 3 4 4 1 10 10 5 2 ...
##  $ win_kills3                      : num  4 0 5 7 2 11 9 5 2 2 ...
##  $ win_kills4                      : num  6 7 9 2 11 10 9 10 2 6 ...
##  $ win_kills5                      : num  7 2 4 3 8 8 2 0 11 4 ...
##  $ win_deaths1                     : num  4 5 5 2 1 3 2 5 2 3 ...
##  $ win_deaths2                     : num  1 0 1 2 4 0 2 5 6 0 ...
##  $ win_deaths3                     : num  4 0 2 2 4 3 3 4 8 1 ...
##  $ win_deaths4                     : num  4 0 1 3 3 6 1 7 4 0 ...
##  $ win_deaths5                     : num  2 3 2 4 4 5 5 6 8 1 ...
##  $ win_totalDamageDealtToChampions1: num  17898 16662 16241 12111 10900 ...
##  $ win_totalDamageDealtToChampions2: num  15800 11674 7572 5510 11362 ...
##  $ win_totalDamageDealtToChampions3: num  10786 7498 10024 8440 14494 ...
##  $ win_totalDamageDealtToChampions4: num  16964 13016 15115 5373 27391 ...
##  $ win_totalDamageDealtToChampions5: num  11568 11393 12395 11038 22399 ...
##  $ win_goldEarned1                 : num  9802 8452 9029 10175 7217 ...
##  $ win_goldEarned2                 : num  9203 9069 6921 5552 10497 ...
##  $ win_goldEarned3                 : num  11127 6023 8331 7439 10323 ...
##  $ win_goldEarned4                 : num  9286 9868 11860 5873 13499 ...
##  $ win_goldEarned5                 : num  10414 7660 8589 7033 12720 ...
##  $ win_visionScore1                : num  28 14 11 17 42 41 19 23 72 9 ...
##  $ win_visionScore2                : num  16 27 35 25 38 41 46 55 50 21 ...
##  $ win_visionScore3                : num  23 46 25 17 30 21 25 47 29 13 ...
```

```
##  $ win_visionScore4                : num  17 16 15 9 18 39 31 25 80 19 ...
##  $ win_visionScore5                : num  36 22 21 19 26 19 86 70 22 10 ...
##  $ win_totalTimeCrowdControlDealt1 : num  183 33 178 134 69 310 168 279 365 15 ...
##  $ win_totalTimeCrowdControlDealt2 : num  92 291 82 61 503 45 291 493 119 62 ...
##  $ win_totalTimeCrowdControlDealt3 : num  231 31 371 332 562 133 102 287 456 126 ...
##  $ win_totalTimeCrowdControlDealt4 : num  54 235 140 274 79 78 444 501 215 209 ...
##  $ win_totalTimeCrowdControlDealt5 : num  281 407 122 163 69 73 92 193 300 168 ...
##  $ gameId                          : num  4.25e+09 4.25e+09 4.26e+09 4.26e+09 4.26e+09 ...
```

```
str(lose_stats_df)
```

```
## 'data.frame':    90500 obs. of  31 variables:
##  $ lose_kills1                     : num  3 0 3 1 4 7 0 11 3 0 ...
##  $ lose_kills2                     : num  0 2 5 6 2 1 3 5 10 1 ...
##  $ lose_kills3                     : num  4 3 1 2 4 4 4 3 7 2 ...
##  $ lose_kills4                     : num  4 3 1 1 1 5 2 7 2 2 ...
##  $ lose_kills5                     : num  4 0 1 3 5 0 4 1 6 0 ...
##  $ lose_deaths1                    : num  6 4 7 6 7 7 10 10 7 5 ...
##  $ lose_deaths2                    : num  6 6 4 3 6 9 5 4 6 2 ...
##  $ lose_deaths3                    : num  5 3 7 7 1 11 8 6 10 2 ...
##  $ lose_deaths4                    : num  7 4 5 4 7 4 9 7 10 3 ...
##  $ lose_deaths5                    : num  7 2 5 7 4 6 7 6 5 6 ...
##  $ lose_totalDamageDealtToChampions1: num  10844 4618 7096 9492 9686 ...
##  $ lose_totalDamageDealtToChampions2: num  7095 14837 17030 8557 14045 ...
##  $ lose_totalDamageDealtToChampions3: num  13458 9197 8735 6679 24086 ...
##  $ lose_totalDamageDealtToChampions4: num  9670 10035 7849 4058 2959 ...
##  $ lose_totalDamageDealtToChampions5: num  14972 5531 5815 6912 16719 ...
##  $ lose_goldEarned1                : num  6844 4524 6551 5442 7928 ...
##  $ lose_goldEarned2                : num  5205 8823 7562 7846 8042 ...
##  $ lose_goldEarned3                : num  8226 7788 5346 5092 12502 ...
##  $ lose_goldEarned4                : num  7911 9008 5004 3931 6185 ...
##  $ lose_goldEarned5                : num  8815 6993 5931 5071 10729 ...
##  $ lose_visionScore1               : num  14 30 14 1 25 11 17 46 30 13 ...
##  $ lose_visionScore2               : num  34 16 13 27 10 48 30 34 25 7 ...
##  $ lose_visionScore3               : num  8 27 40 9 29 14 38 19 39 13 ...
##  $ lose_visionScore4               : num  21 28 15 26 65 15 81 44 84 24 ...
##  $ lose_visionScore5               : num  14 10 9 5 28 13 13 68 45 8 ...
##  $ lose_totalTimeCrowdControlDealt1 : num  19 80 133 71 422 188 97 71 246 20 ...
##  $ lose_totalTimeCrowdControlDealt2 : num  38 67 249 476 151 77 36 327 98 102 ...
##  $ lose_totalTimeCrowdControlDealt3 : num  173 491 135 13 161 109 347 433 36 169 ...
##  $ lose_totalTimeCrowdControlDealt4 : num  305 50 125 52 63 204 208 44 95 47 ...
##  $ lose_totalTimeCrowdControlDealt5 : num  237 0 226 88 97 101 81 242 447 182 ...
##  $ gameId                          : num  4.25e+09 4.25e+09 4.26e+09 4.26e+09 4.26e+09 ...
```

Now we will merge the two data sets together based on their matching gameId column. This way our model can catagorize our data by team 0 or team 1 winning based on both teams contrasting stats.

```
# Replacing column names for rbind
colnames(win_stats_df) <- c('kill1', 'kill2', 'kill3', 'kill4', 'kill5', 'death1', 'death2', 'death3',
colnames(lose_stats_df) <- c('kill1', 'kill2', 'kill3', 'kill4', 'kill5', 'death1', 'death2', 'death3',

# Adding column based on dataset it is in
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
win_stats_df <- win_stats_df %>%
  mutate(won="True")

lose_stats_df <- lose_stats_df %>%
  mutate(won ="False")

#full_stats_df <- merge(win_stats_df, lose_stats_df, by = "gameId")
full_stats_df <- rbind(win_stats_df, lose_stats_df)
drop <- c("gameId")
full_stats_df <- full_stats_df[,!(names(full_stats_df) %in% drop)]

# Make our won column a factor for classification
full_stats_df$won <- as.factor(full_stats_df$won)

str(full_stats_df)
```

```
## 'data.frame':    181000 obs. of  31 variables:
##  $ kill1                       : num  10 3 7 11 0 7 9 8 18 4 ...
##  $ kill2                       : num  4 7 3 4 4 1 10 10 5 2 ...
##  $ kill3                       : num  4 0 5 7 2 11 9 5 2 2 ...
##  $ kill4                       : num  6 7 9 2 11 10 9 10 2 6 ...
##  $ kill5                       : num  7 2 4 3 8 8 2 0 11 4 ...
##  $ death1                      : num  4 5 5 2 1 3 2 5 2 3 ...
##  $ death2                      : num  1 0 1 2 4 0 2 5 6 0 ...
##  $ death3                      : num  4 0 2 2 4 3 3 4 8 1 ...
##  $ death4                      : num  4 0 1 3 3 6 1 7 4 0 ...
##  $ death5                      : num  2 3 2 4 4 5 5 6 8 1 ...
##  $ totalDamageDealtToChampions1: num  17898 16662 16241 12111 10900 ...
##  $ totalDamageDealtToChampions2: num  15800 11674 7572 5510 11362 ...
##  $ totalDamageDealtToChampions3: num  10786 7498 10024 8440 14494 ...
##  $ totalDamageDealtToChampions4: num  16964 13016 15115 5373 27391 ...
##  $ totalDamageDealtToChampions5: num  11568 11393 12395 11038 22399 ...
##  $ goldEarned1                 : num  9802 8452 9029 10175 7217 ...
##  $ goldEarned2                 : num  9203 9069 6921 5552 10497 ...
##  $ goldEarned3                 : num  11127 6023 8331 7439 10323 ...
##  $ goldEarned4                 : num  9286 9868 11860 5873 13499 ...
##  $ goldEarned5                 : num  10414 7660 8589 7033 12720 ...
##  $ visionScore1                : num  28 14 11 17 42 41 19 23 72 9 ...
##  $ visionScore2                : num  16 27 35 25 38 41 46 55 50 21 ...
##  $ visionScore3                : num  23 46 25 17 30 21 25 47 29 13 ...
##  $ visionScore4                : num  17 16 15 9 18 39 31 25 80 19 ...
##  $ visionScore5                : num  36 22 21 19 26 19 86 70 22 10 ...
##  $ totalTimeCrowdControlDealt1 : num  183 33 178 134 69 310 168 279 365 15 ...
##  $ totalTimeCrowdControlDealt2 : num  92 291 82 61 503 45 291 493 119 62 ...
##  $ totalTimeCrowdControlDealt3 : num  231 31 371 332 562 133 102 287 456 126 ...
##  $ totalTimeCrowdControlDealt4 : num  54 235 140 274 79 78 444 501 215 209 ...
##  $ totalTimeCrowdControlDealt5 : num  281 407 122 163 69 73 92 193 300 168 ...
```

```
##  $ won                          : Factor w/ 2 levels "False","True": 2 2 2 2 2 2 2 2 2 2 ...
```

```
i <- sample(1:nrow(full_stats_df), .1*nrow(full_stats_df), replace=FALSE)
full_stats_smol <- full_stats_df[i,]

lolDataless <- full_stats_smol %>% rowwise() %>% mutate(TotalKill = sum(c_across(kill1:kill5)))

lolDataless <- lolDataless %>% rowwise() %>% mutate(TotalDeath = sum(c_across(death1:death5)))

lolDataless <- lolDataless %>% rowwise() %>% mutate(TotalDamage = sum(c_across(totalDamageDealtToChampi

lolDataless <- lolDataless %>% rowwise() %>% mutate(TotalGold = sum(c_across(goldEarned1:goldEarned5)))

lolDataless <- lolDataless %>% rowwise() %>% mutate(TotalVision = sum(c_across(visionScore1:visionScore

lolDataless <- lolDataless %>% rowwise() %>% mutate(TotalCrowdControl = sum(c_across(totalTimeCrowdCont

drop <- c('kill1', 'kill2', 'kill3', 'kill4', 'kill5', 'death1', 'death2', 'death3', 'death4', 'death5'

lolDataless = lolDataless[, !(names(lolDataless) %in% drop)]
summary(lolDataless)
```

```
##     won          TotalKill       TotalDeath     TotalDamage       TotalGold
##  False:9069   Min.   : 0.00   Min.   : 0.00   Min.   :   109   Min.   :  3355
##  True :9031   1st Qu.:14.00   1st Qu.:14.00   1st Qu.: 39329   1st Qu.: 35199
##               Median :22.00   Median :22.00   Median : 60925   Median : 46462
##               Mean   :22.47   Mean   :22.51   Mean   : 64779   Mean   : 46273
##               3rd Qu.:30.00   3rd Qu.:30.00   3rd Qu.: 85922   3rd Qu.: 57396
##               Max.   :80.00   Max.   :84.00   Max.   :257629   Max.   :104458
##   TotalVision    TotalCrowdControl
##  Min.   :  0.0   Min.   :   0.0
##  1st Qu.: 82.0   1st Qu.: 568.8
##  Median :126.0   Median : 799.0
##  Mean   :129.7   Mean   : 849.5
##  3rd Qu.:171.0   3rd Qu.:1069.2
##  Max.   :456.0   Max.   :3652.0
```

Next let's randomly divide the data into train, test, and validate:

```
set.seed(1010)
spec <-c(train=.6, test=.2, validate=.2)
i <- sample(cut(1:nrow(lolDataless), nrow(lolDataless)*cumsum(c(0,spec)), labels=names(spec)))
full_stats_train <- lolDataless[i=="train",]
full_stats_test <- lolDataless[i=="test",]
full_stats_validate <- lolDataless[i=="validate",]

summary(full_stats_train)
```

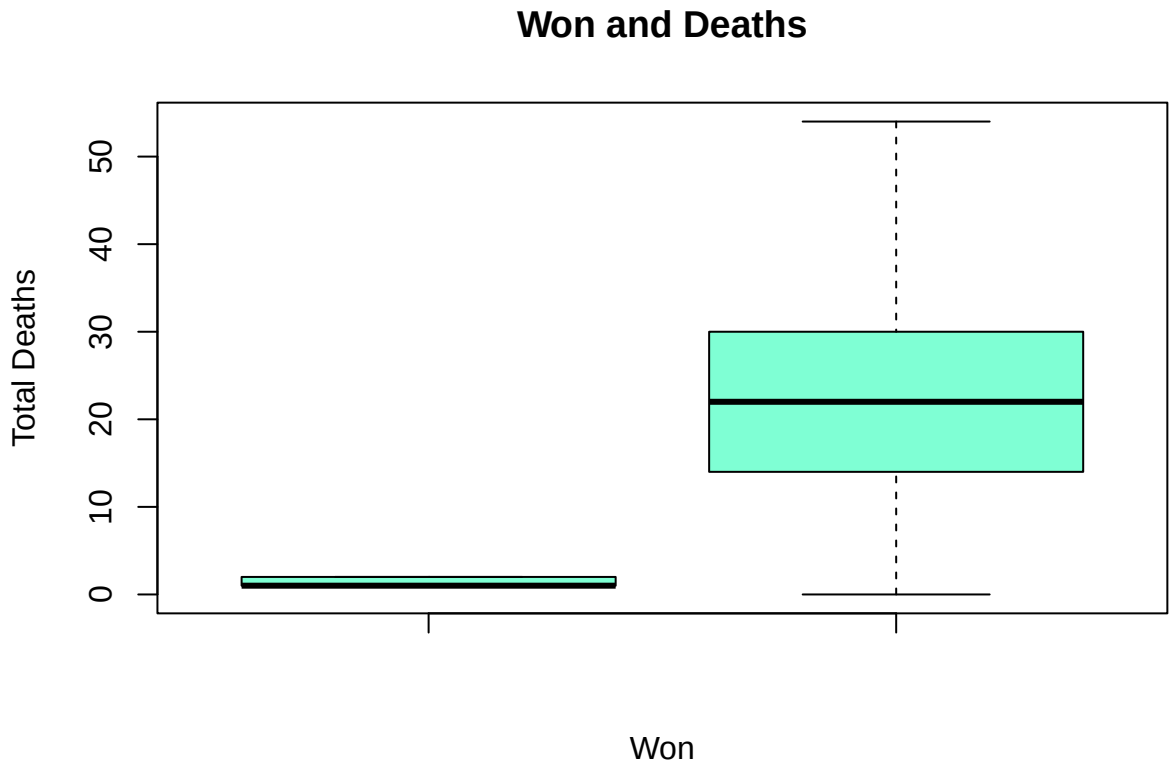
```
##     won          TotalKill       TotalDeath     TotalDamage       TotalGold
##  False:5488   Min.   : 0.00   Min.   : 0.00   Min.   :   199   Min.   : 3468
##  True :5372   1st Qu.:14.00   1st Qu.:14.00   1st Qu.: 38773   1st Qu.:34848
##               Median :22.00   Median :22.00   Median : 60322   Median :46228
##               Mean   :22.34   Mean   :22.51   Mean   : 64362   Mean   :46060
##               3rd Qu.:30.00   3rd Qu.:30.00   3rd Qu.: 85796   3rd Qu.:57293
##               Max.   :80.00   Max.   :74.00   Max.   :257629   Max.   :99827
##   TotalVision    TotalCrowdControl
```

```
##  Min.   :  0.0   Min.   :   1.0
##  1st Qu.: 81.0   1st Qu.: 564.0
##  Median :125.0   Median : 794.0
##  Mean   :128.8   Mean   : 845.9
##  3rd Qu.:171.0   3rd Qu.:1077.0
##  Max.   :456.0   Max.   :3397.0
```

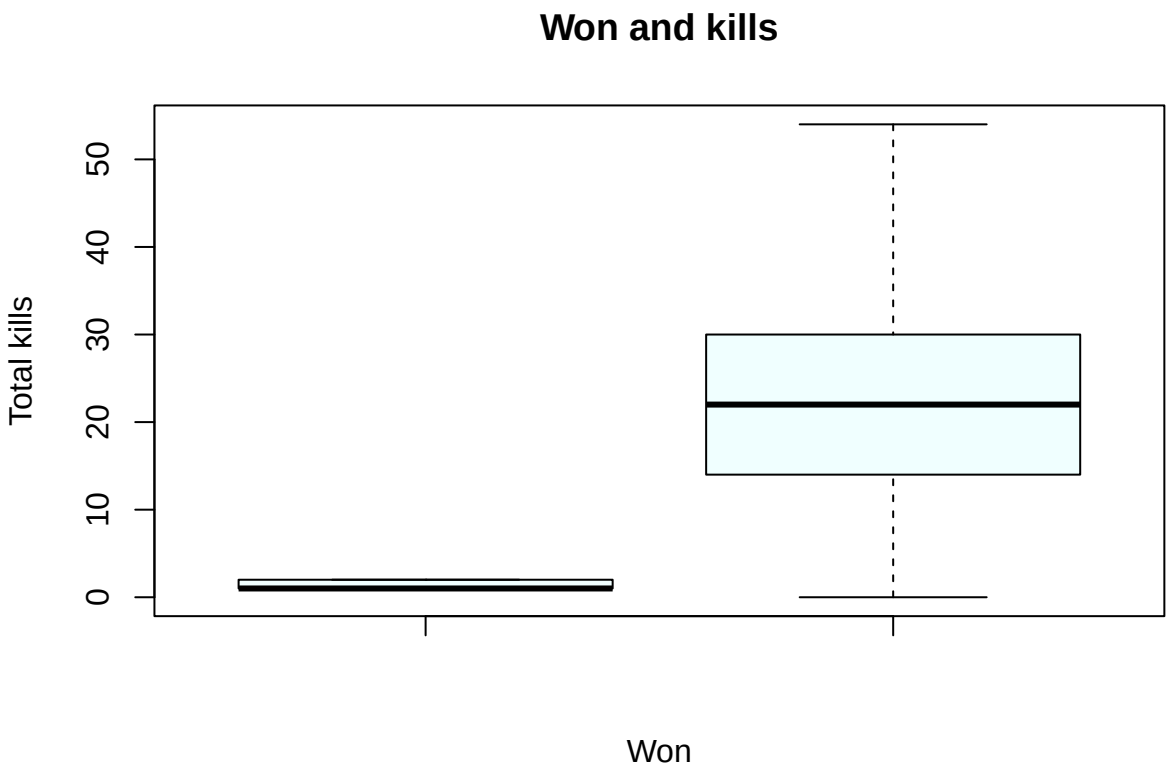
Data Exploration

Next we will plot some of our data to see possible differences/correlations. Time to do some data exploration.

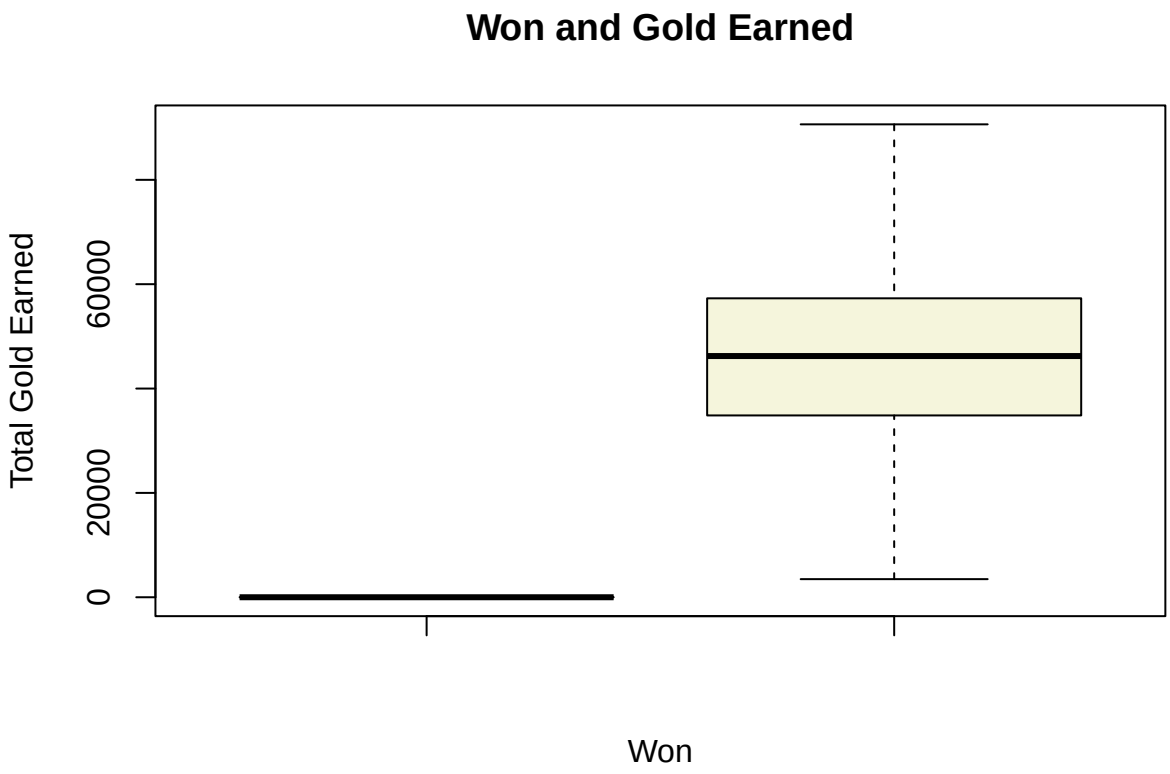
```
boxplot(full_stats_train$won, full_stats_train$TotalDeath, main="Won and Deaths", xlab="Won", ylab="Tota
```



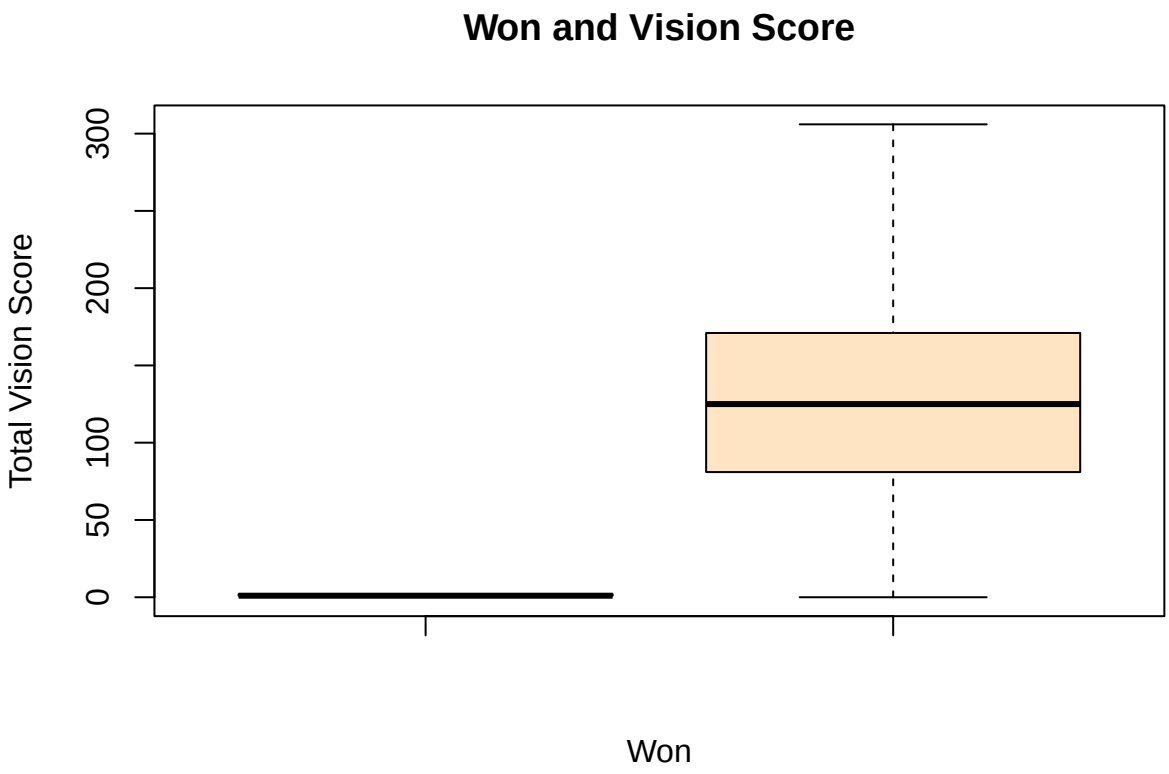

```
boxplot(full_stats_train$won, full_stats_train$TotalKill, main="Won and kills", xlab="Won", ylab="Total
```



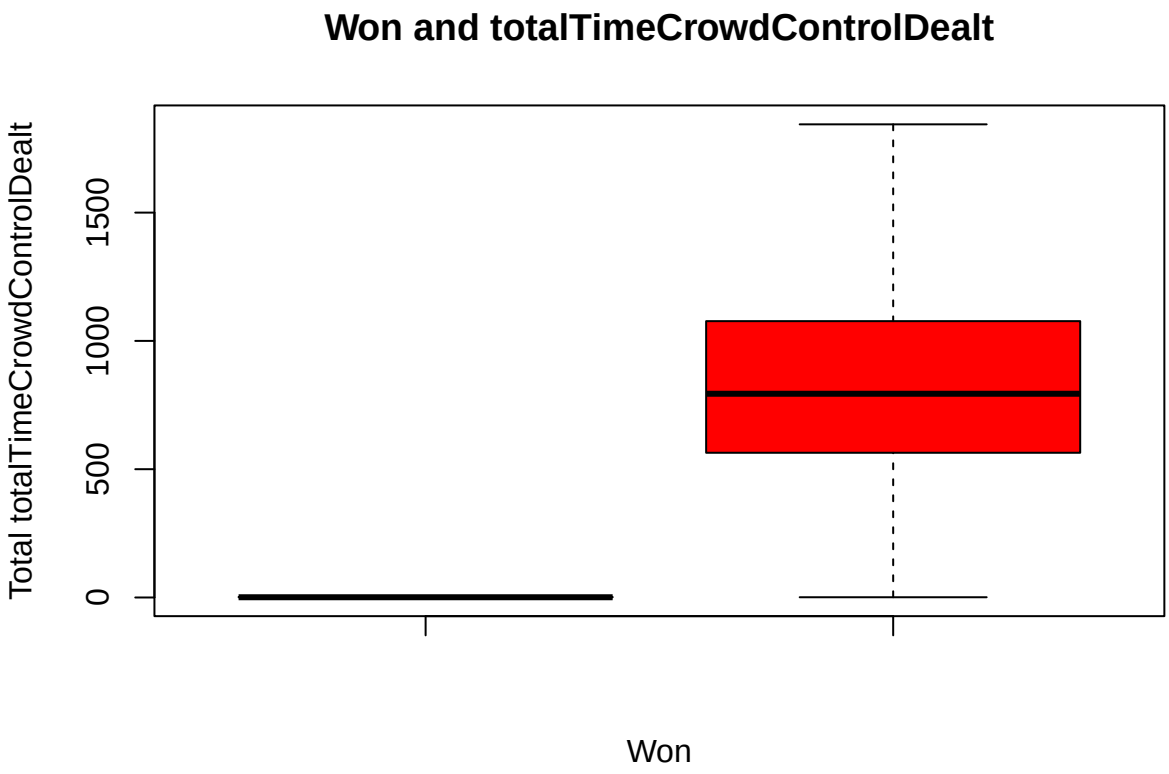

```
boxplot(full_stats_train$won, full_stats_train$TotalGold, main="Won and Gold Earned", xlab="Won", ylab=
```



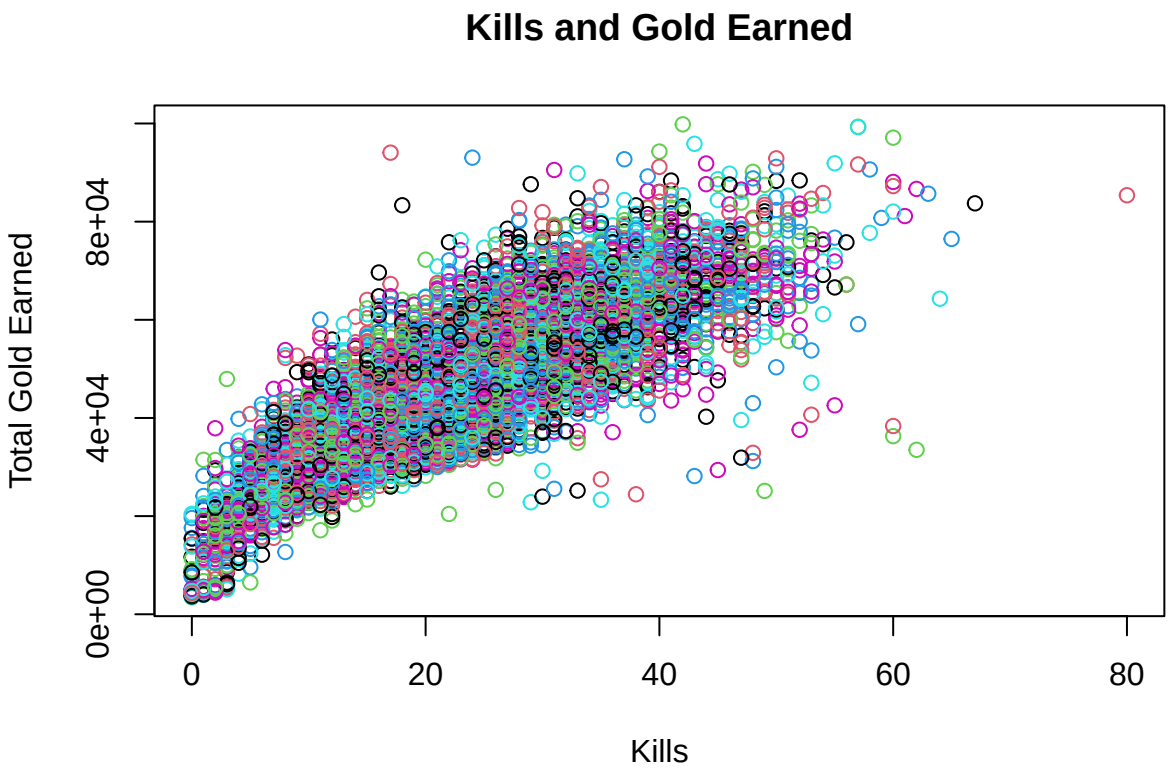

```
boxplot(full_stats_train$won, full_stats_train$TotalVision, main="Won and Vision Score", xlab="Won", yla
```



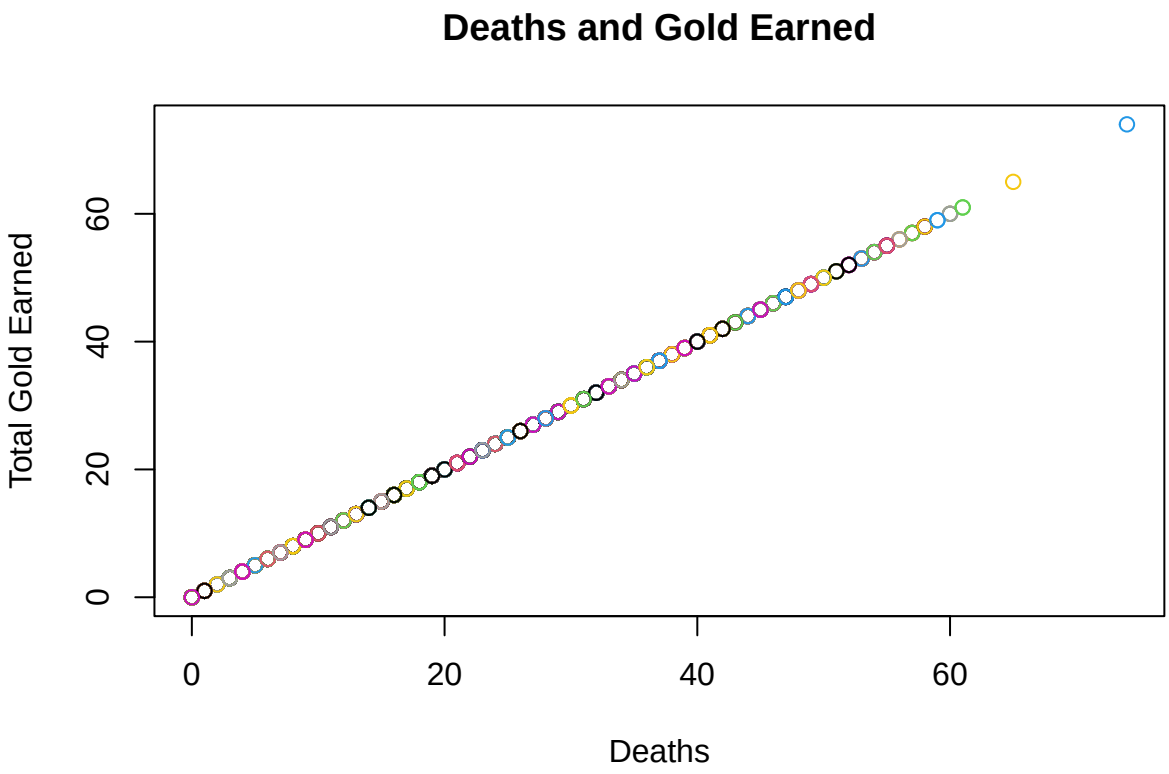

```
boxplot(full_stats_train$won, full_stats_train$TotalCrowdControl, main="Won and totalTimeCrowdControlDea
```



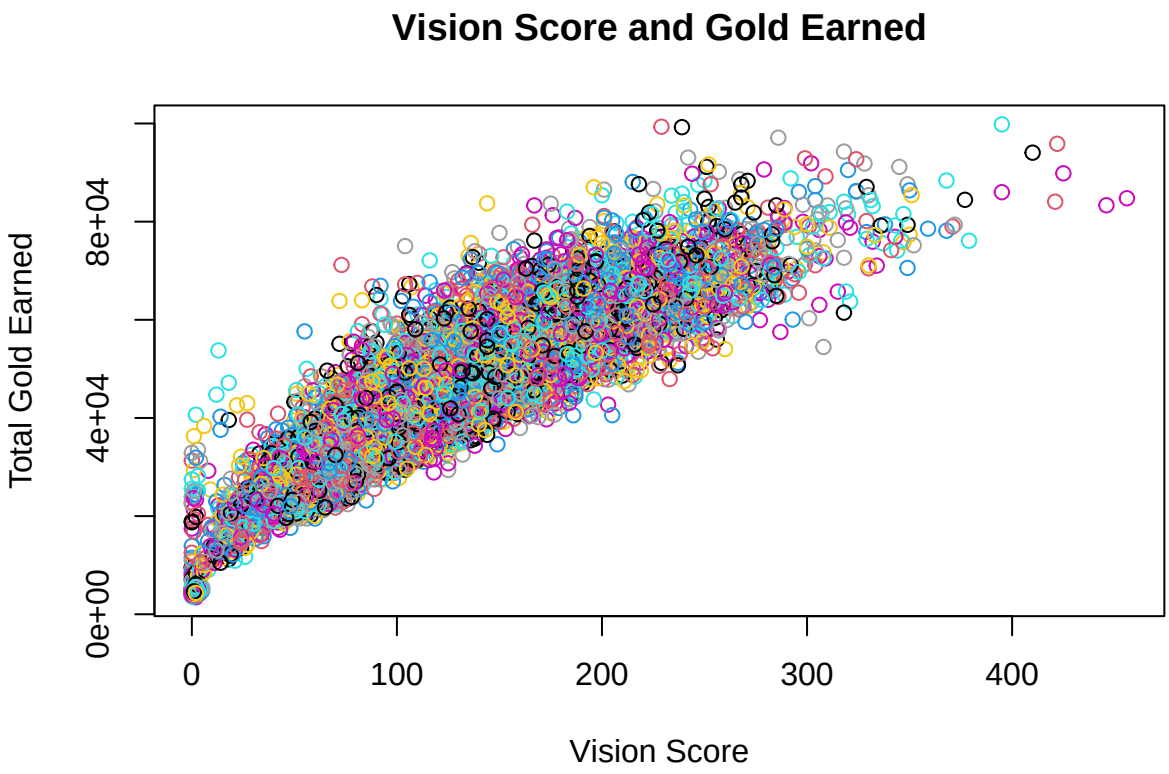

```
plot(full_stats_train$TotalKill, full_stats_train$TotalGold, main="Kills and Gold Earned", xlab="Kills"
```



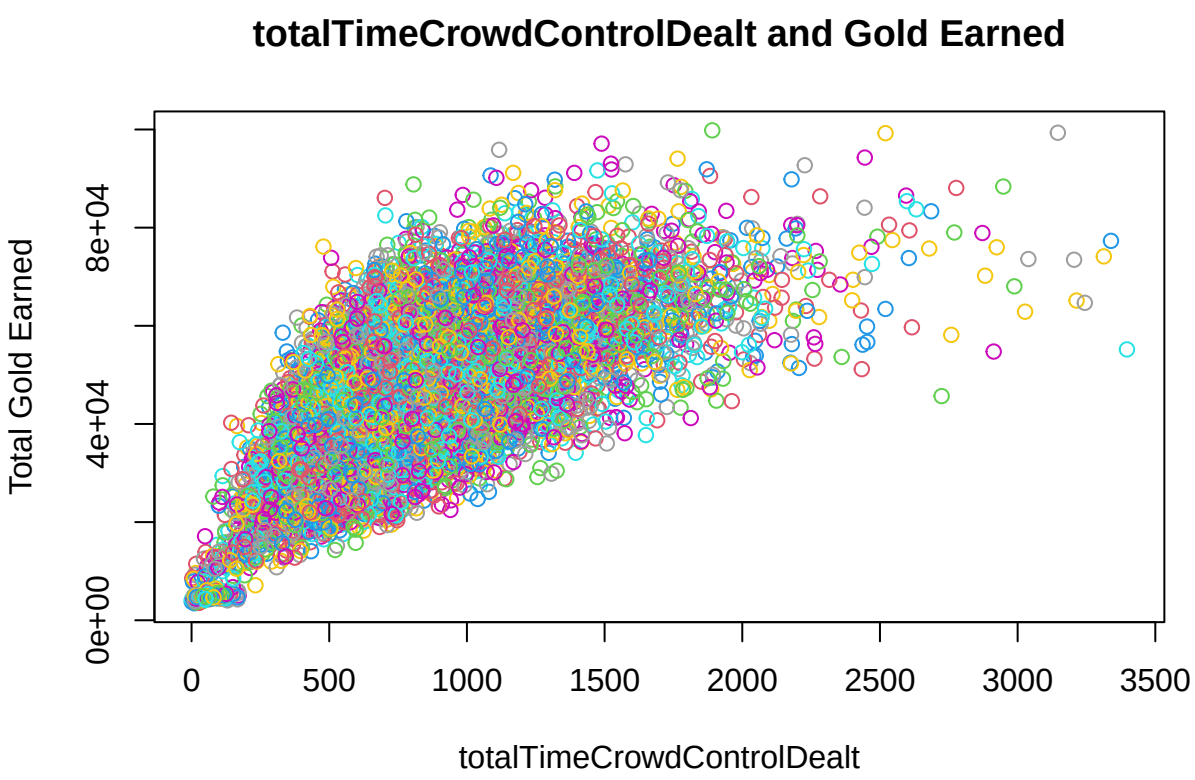

```
plot(full_stats_train$TotalDeath, full_stats_train$TotalDeath, main="Deaths and Gold Earned", xlab="Dea
```




```
plot(full_stats_train$TotalVision, full_stats_train$TotalGold, main="Vision Score and Gold Earned", xlab
```




```
plot(full_stats_train$TotalCrowdControl, full_stats_train$TotalGold, main="totalTimeCrowdControlDealt ar
```



Performing Logistic Regression

```
library(caret)
```

```
## Loading required package: ggplot2
```

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

```
glm_won <- glm(won~., data=full_stats_train, family = "binomial")
summary(glm_won)
```

```
##
## Call:
## glm(formula = won ~ ., family = "binomial", data = full_stats_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.0805  -0.1020  -0.0017   0.0934   4.0701
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -1.506e+00  1.676e-01  -8.985  < 2e-16 ***
## TotalKill          3.301e-01  1.349e-02  24.469  < 2e-16 ***
## TotalDeath        -4.939e-01  1.289e-02 -38.304  < 2e-16 ***
## TotalDamage        1.218e-05  4.187e-06   2.909  0.00362 **
## TotalGold          1.825e-04  1.320e-05  13.824  < 2e-16 ***
## TotalVision       -2.542e-02  2.062e-03 -12.329  < 2e-16 ***
## TotalCrowdControl -8.625e-04  1.710e-04  -5.044 4.56e-07 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 15053.9  on 10859  degrees of freedom
## Residual deviance:  2504.6  on 10853  degrees of freedom
## AIC: 2518.6
##
## Number of Fisher Scoring iterations: 8
```

```
glm_probs <- predict(glm_won, newdata = full_stats_test)
str(glm_probs)
```

```
##  Named num [1:3620] -1.81 2.89 -1.53 3.71 1.3 ...
##  - attr(*, "names")= chr [1:3620] "1" "2" "3" "4" ...
```

```
glm_pred <- ifelse(glm_probs > 0.5, "True", "False")
glm_acc <- mean(glm_pred == full_stats_test$won)

confusionMatrix(as.factor(glm_pred), reference = full_stats_test$won)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  1738  117
##      True     59 1706
##
##                Accuracy : 0.9514
##                  95% CI : (0.9439, 0.9582)
##     No Information Rate : 0.5036
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9028
##
##  Mcnemar's Test P-Value : 1.735e-05
##
##             Sensitivity : 0.9672
##             Specificity : 0.9358
##          Pos Pred Value : 0.9369
##          Neg Pred Value : 0.9666
##              Prevalence : 0.4964
##          Detection Rate : 0.4801
##    Detection Prevalence : 0.5124
##       Balanced Accuracy : 0.9515
##
##        'Positive' Class : False
##
```

SVM Classification

Linear

Training Now we will build our SVM Linear model

```
library(e1071)
svm_lin <- svm(won~., data=full_stats_train, kernel="linear", cost=10, scale=TRUE)
summary(svm_lin)
```

```
##
## Call:
## svm(formula = won ~ ., data = full_stats_train, kernel = "linear",
##     cost = 10, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  1184
##
##  ( 592 592 )
##
##
## Number of Classes:  2
##
## Levels:
##  False True
```

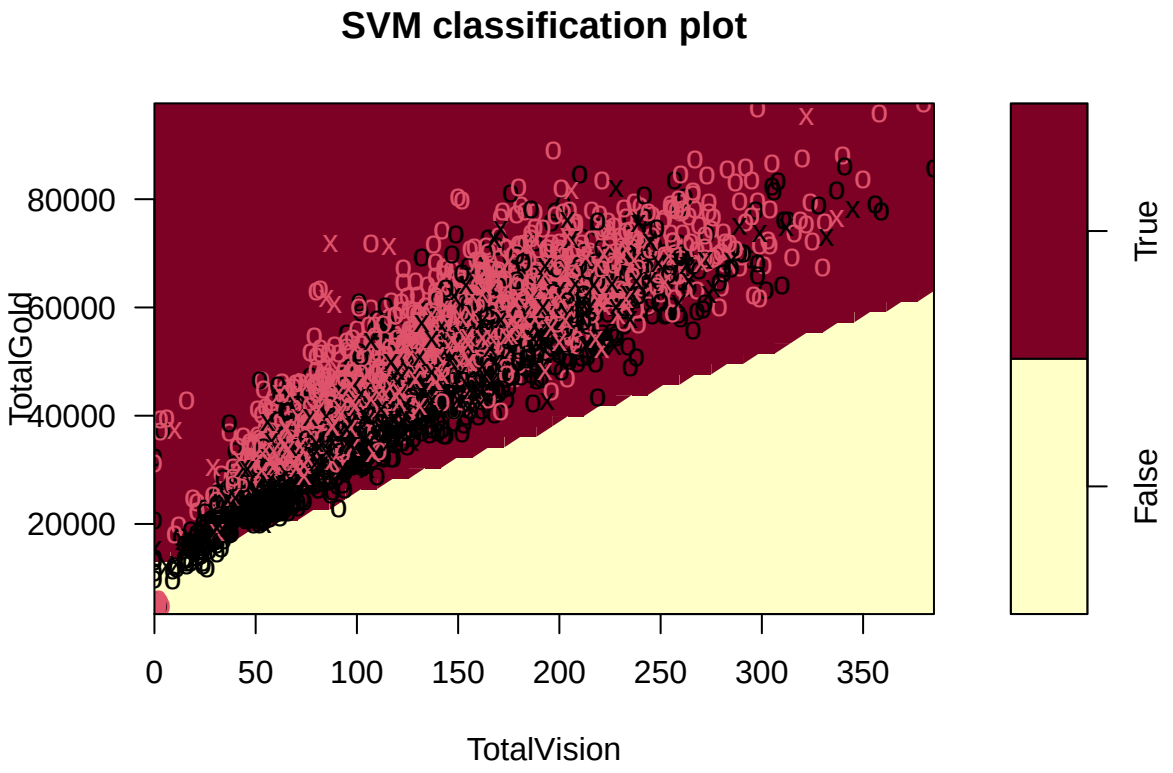
Testing & Evaluation Now we can evaluate on the test set:

```
library(caret)
svm_probs_lin <- predict(svm_lin, newdata = full_stats_test)

confusionMatrix(svm_probs_lin, reference = full_stats_test$won)
```

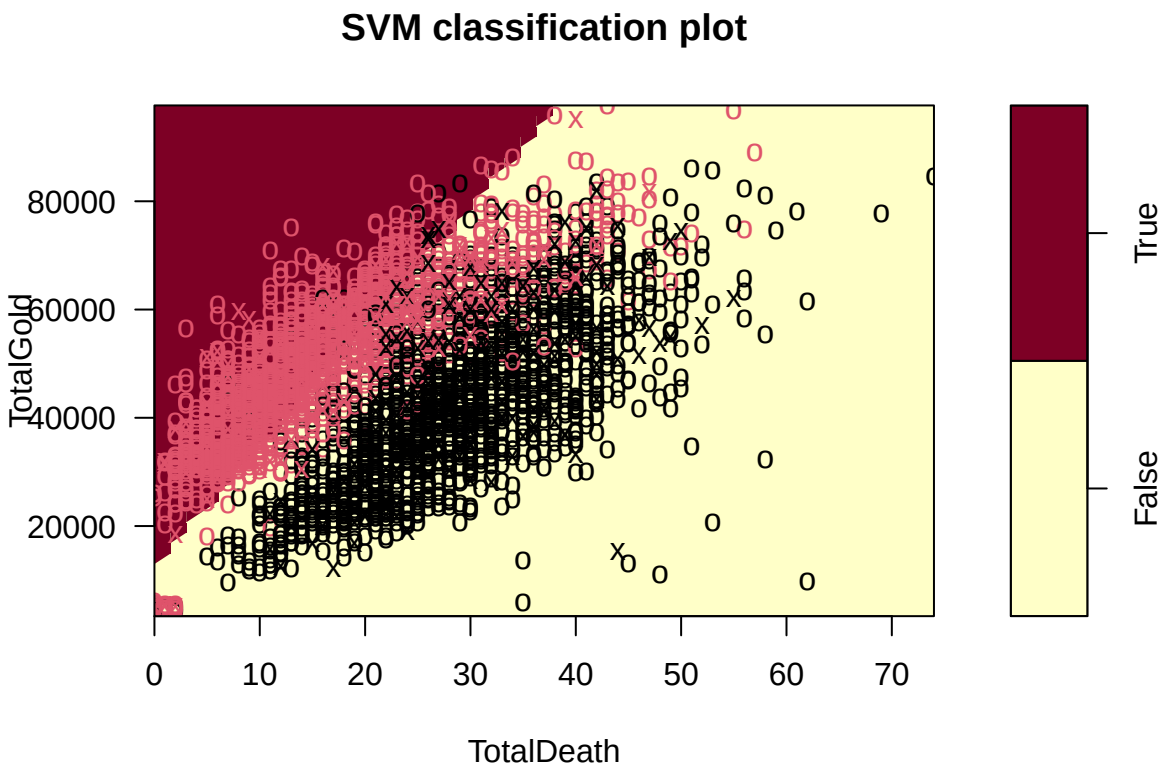
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  1719  101
##      True     78 1722
##
##                Accuracy : 0.9506
##                  95% CI : (0.943, 0.9574)
##     No Information Rate : 0.5036
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9011
##
##  Mcnemar's Test P-Value : 0.1001
##
##             Sensitivity : 0.9566
##             Specificity : 0.9446
##          Pos Pred Value : 0.9445
##          Neg Pred Value : 0.9567
##              Prevalence : 0.4964
##          Detection Rate : 0.4749
```

```
##    Detection Prevalence : 0.5028
##       Balanced Accuracy : 0.9506
##
##        'Positive' Class : False
##
```

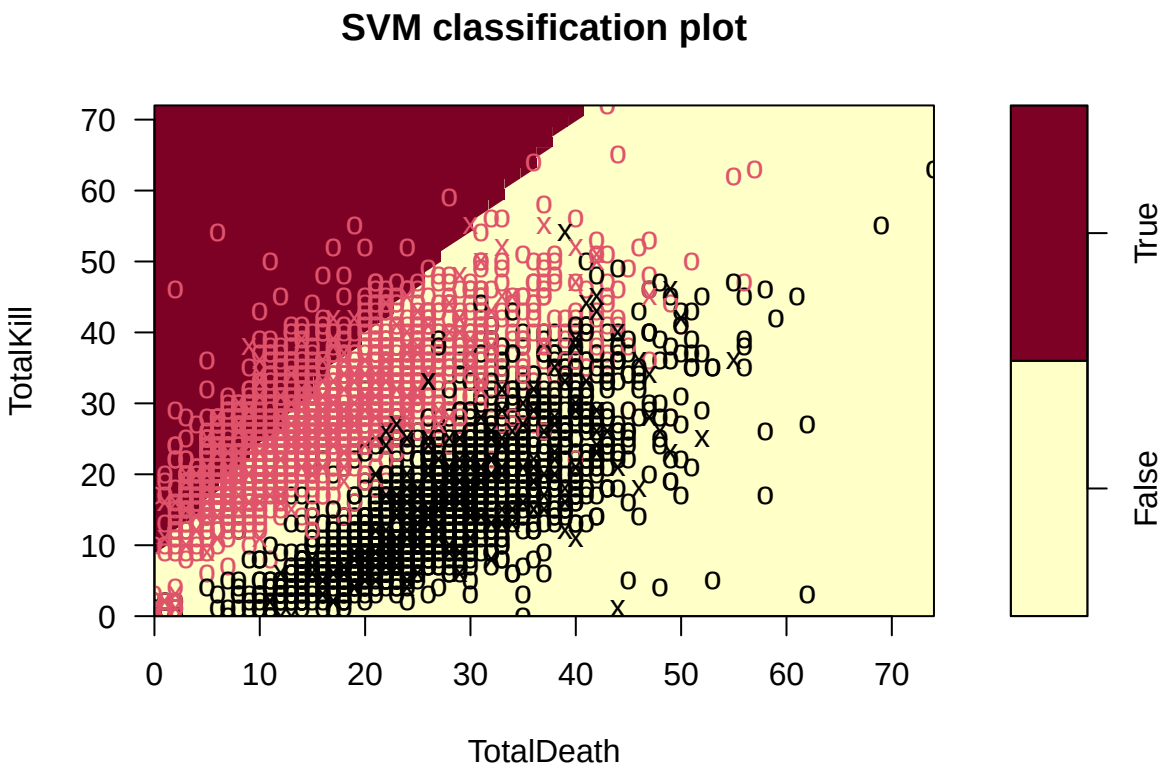
```
plot(svm_lin, full_stats_test, TotalGold ~ TotalVision)
```



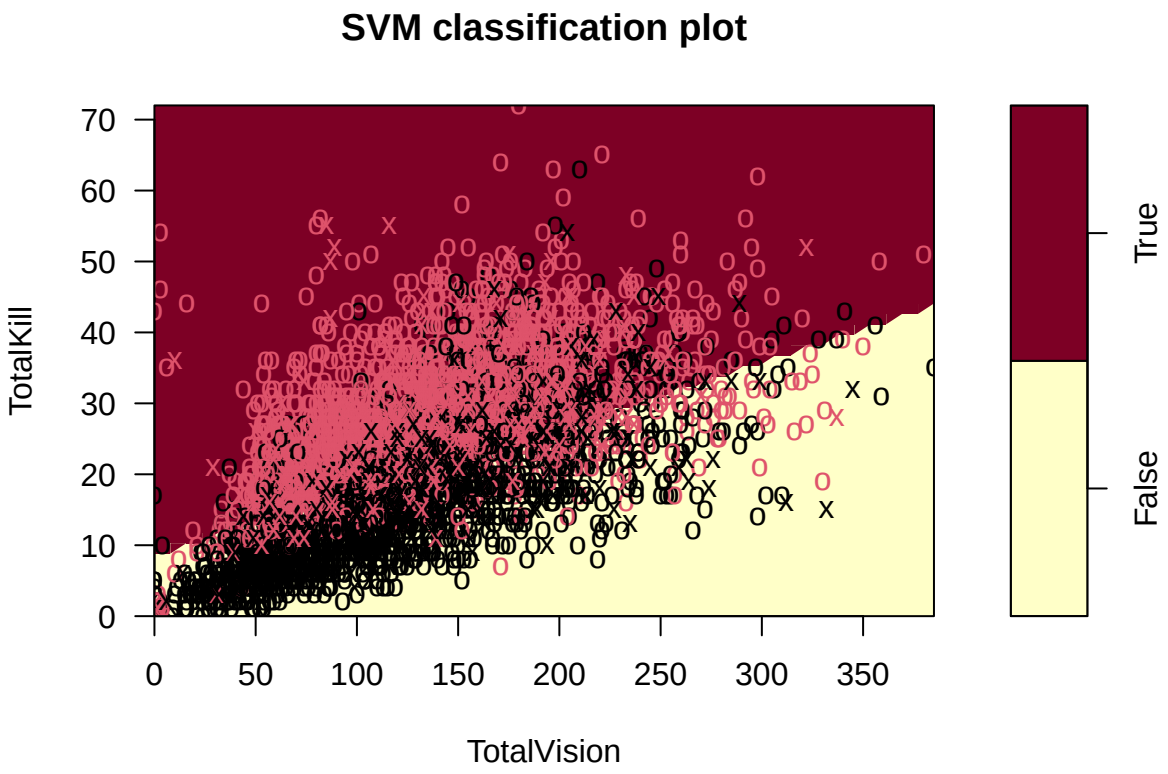

```
plot(svm_lin, full_stats_test, TotalGold ~ TotalDeath)
```



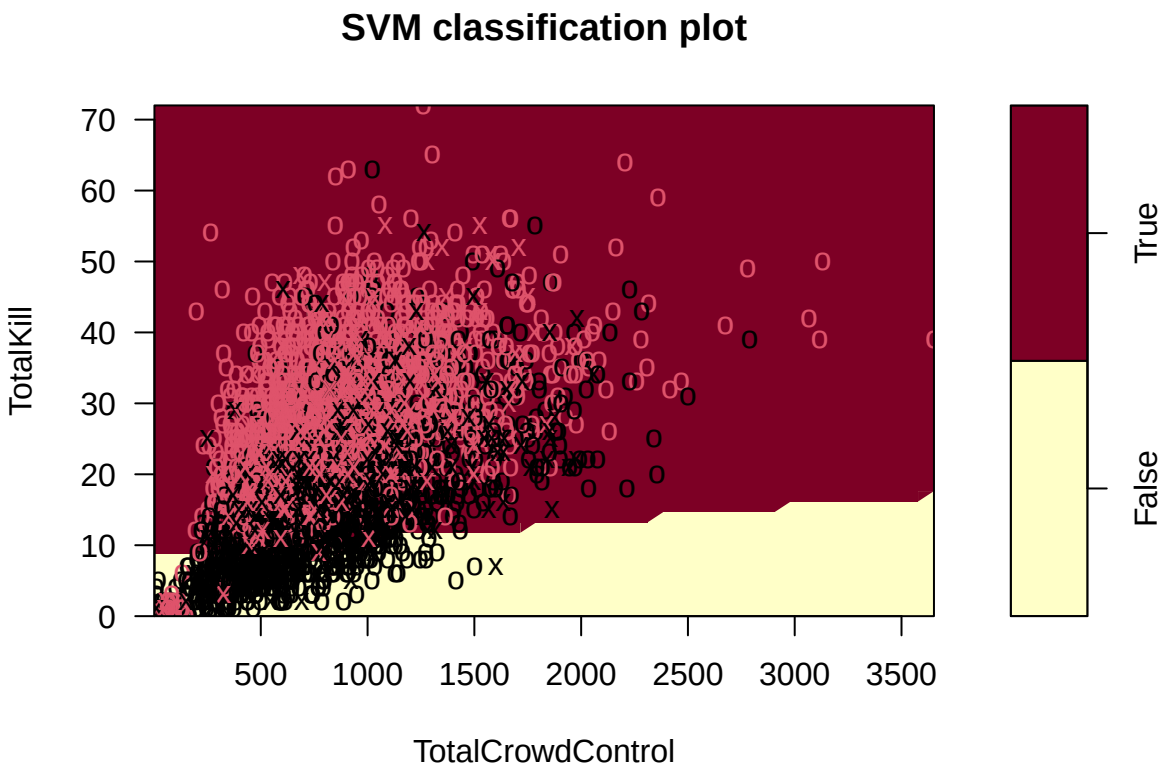

```
plot(svm_lin, full_stats_test, TotalKill ~ TotalDeath)
```



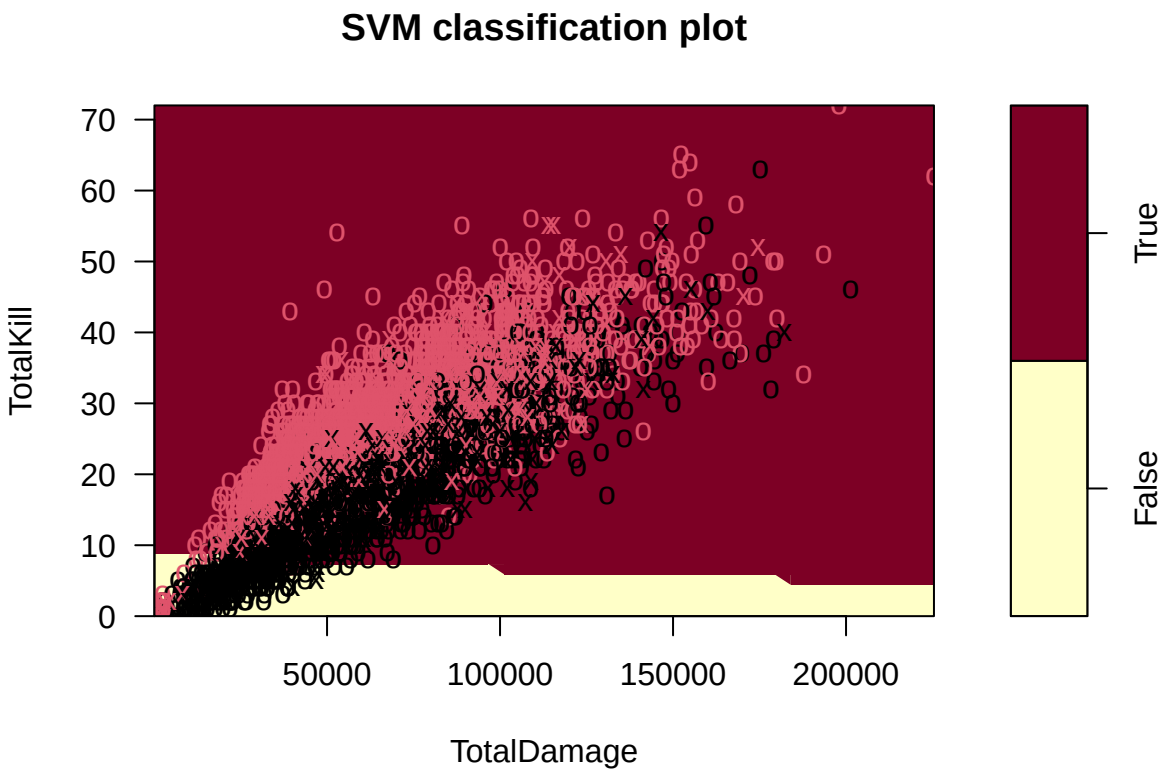

```
plot(svm_lin, full_stats_test, TotalKill ~ TotalVision)
```




```
plot(svm_lin, full_stats_test, TotalKill ~ TotalCrowdControl)
```




```
plot(svm_lin, full_stats_test, TotalKill ~ TotalDamage)
```



Tuning

```
set.seed(1010)
tune_out <- tune(svm, won~., data=full_stats_validate, kernel="linear",
                 ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(tune_out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.04198895
##
## - Detailed performance results:
##    cost      error  dispersion
## 1 1e-03 0.04834254 0.008759806
## 2 1e-02 0.04806630 0.009498193
## 3 1e-01 0.04364641 0.008215362
## 4 1e+00 0.04198895 0.008716140
## 5 5e+00 0.04281768 0.008261675
## 6 1e+01 0.04309392 0.008358605
## 7 1e+02 0.04309392 0.008358605
```

```
best_lin_model <- tune_out$best.model
summary(best_lin_model)
```

Getting Best Model after Tune

```
##
## Call:
## best.tune(method = svm, train.x = won ~ ., data = full_stats_validate,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  408
##
##  ( 204 204 )
##
##
## Number of Classes:  2
##
## Levels:
##  False True
```

```
best_lin_pred <- predict(best_lin_model, newdata=full_stats_test)
acc_best_lin <- mean(best_lin_pred==full_stats_test$won)
```

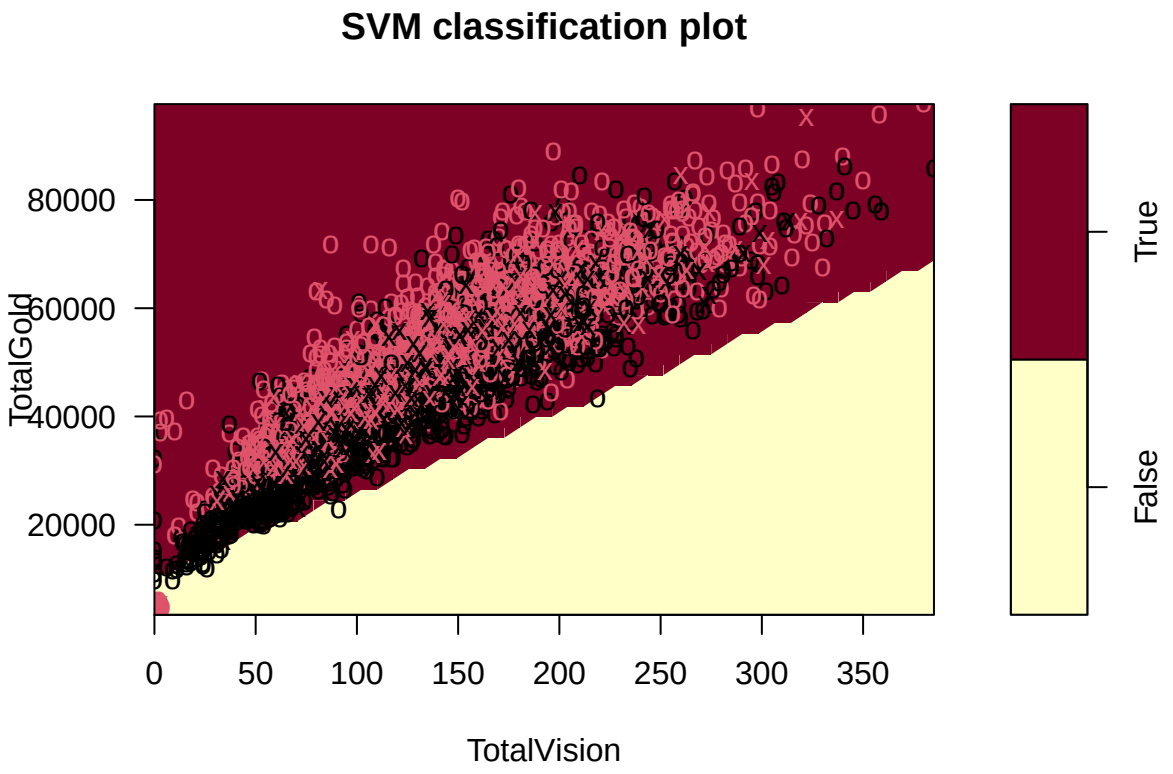
```
confusionMatrix(best_lin_pred, reference = full_stats_test$won)
```

Evaluating Best Model

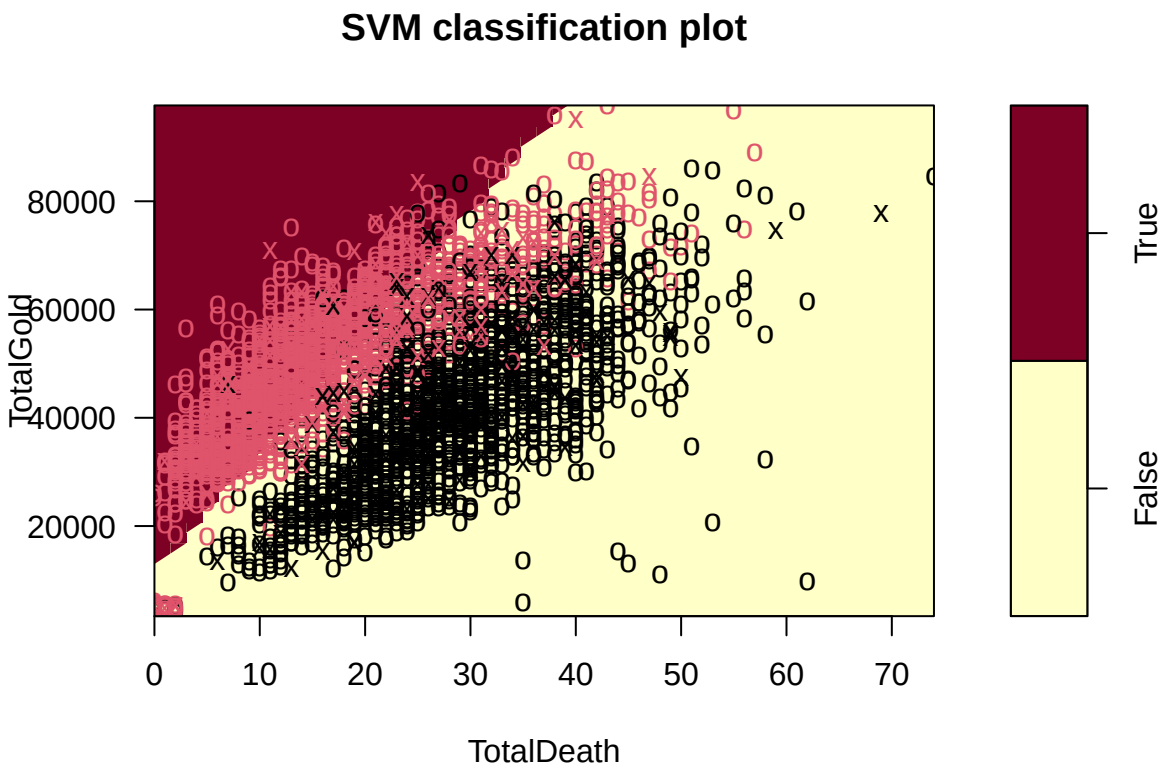
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  1723  100
##      True     74 1723
##
##                Accuracy : 0.9519
##                  95% CI : (0.9445, 0.9587)
##     No Information Rate : 0.5036
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9039
##
##  Mcnemar's Test P-Value : 0.05806
##
##             Sensitivity : 0.9588
##             Specificity : 0.9451
##          Pos Pred Value : 0.9451
##          Neg Pred Value : 0.9588
```

```
##              Prevalence : 0.4964
##          Detection Rate : 0.4760
##    Detection Prevalence : 0.5036
##       Balanced Accuracy : 0.9520
##
##        'Positive' Class : False
##
```

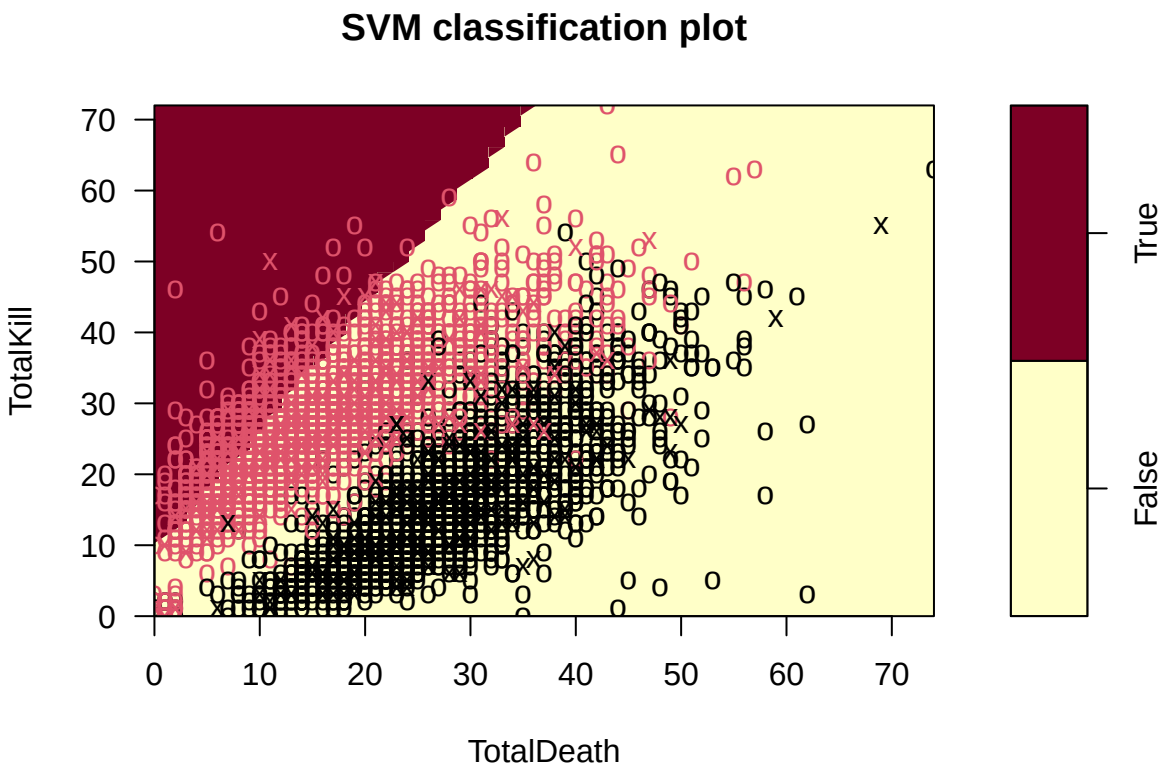
```
plot(best_lin_model, full_stats_test, TotalGold ~ TotalVision)
```



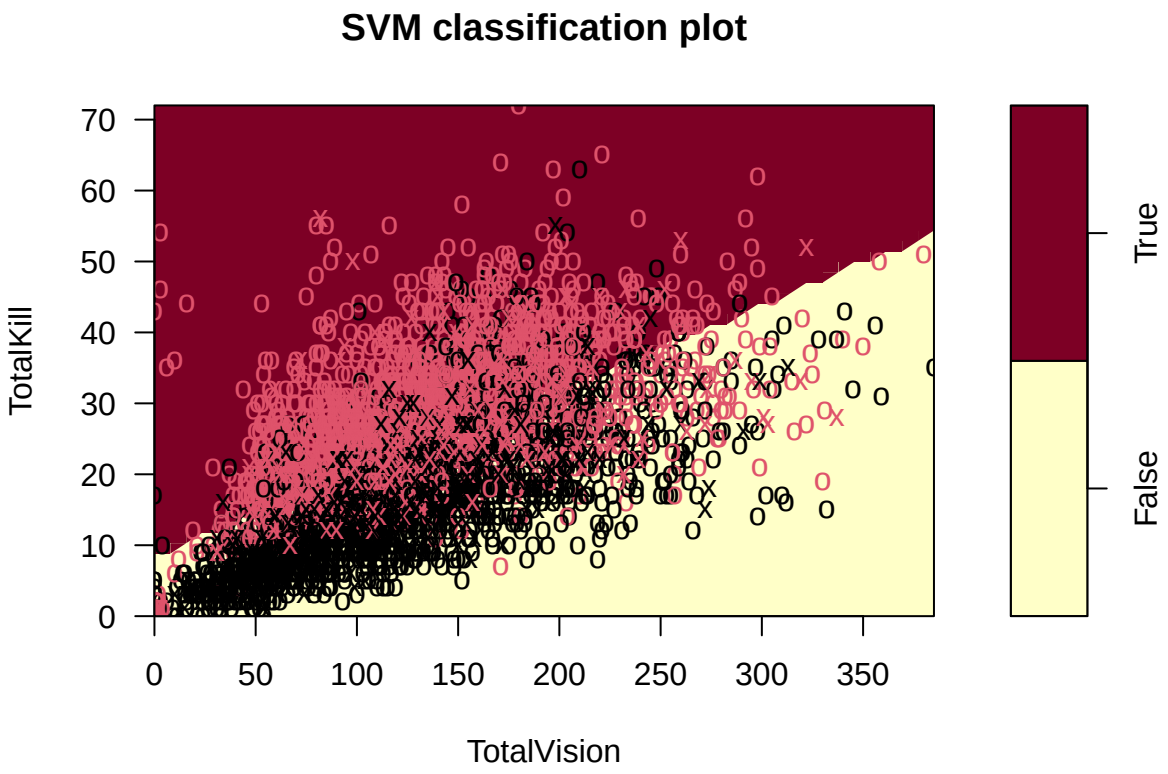

```
plot(best_lin_model, full_stats_test, TotalGold ~ TotalDeath)
```



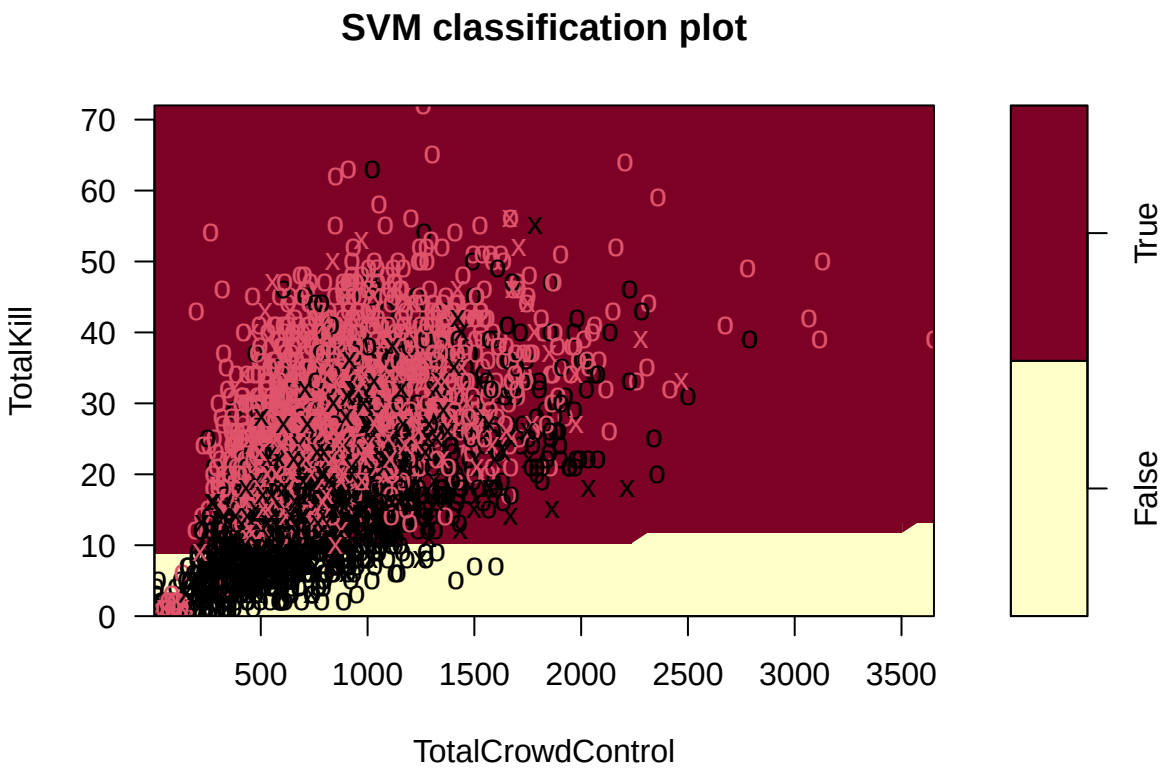

```
plot(best_lin_model, full_stats_test, TotalKill ~ TotalDeath)
```



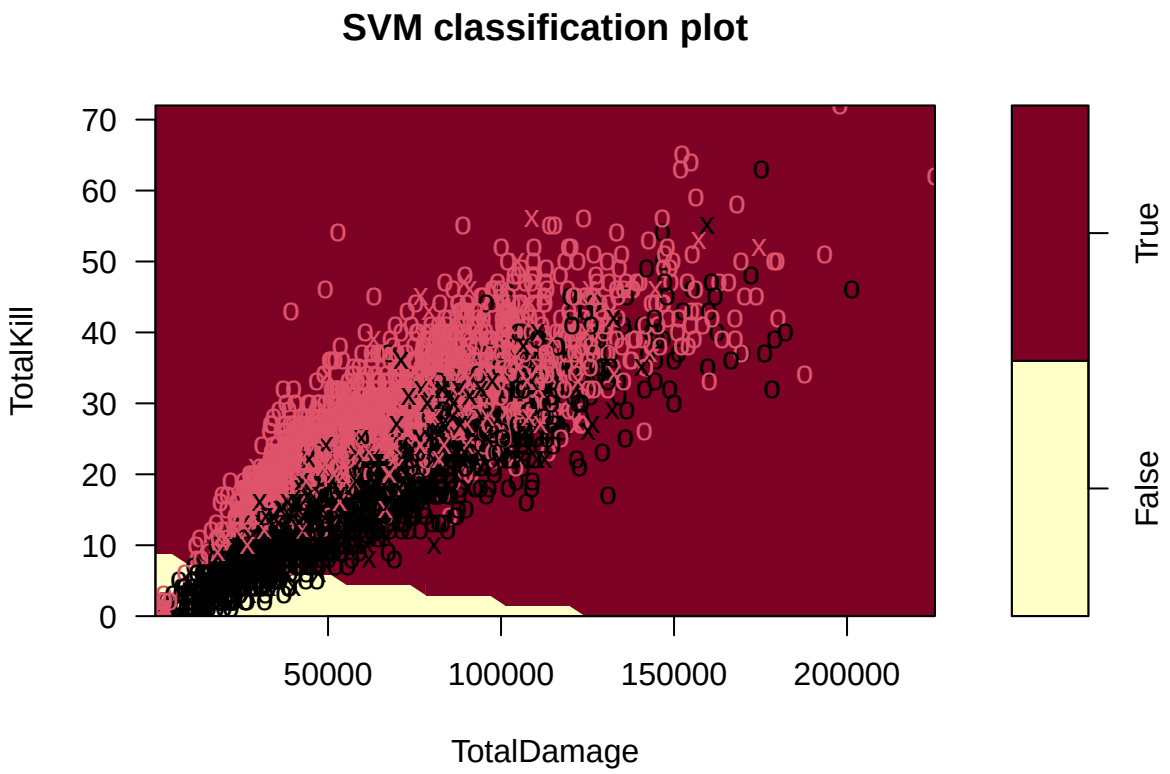

```
plot(best_lin_model, full_stats_test, TotalKill ~ TotalVision)
```




```
plot(best_lin_model, full_stats_test, TotalKill ~ TotalCrowdControl)
```




```
plot(best_lin_model, full_stats_test, TotalKill ~ TotalDamage)
```



Polynomial

Training Now we will build our SVM Polynomial model

```
library(e1071)
svm_poly <- svm(won~., data=full_stats_train, kernel="polynomial", cost=10, scale=TRUE)
summary(svm_poly)
```

```
##
## Call:
## svm(formula = won ~ ., data = full_stats_train, kernel = "polynomial",
##     cost = 10, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  2126
##
##  ( 1060 1066 )
##
##
## Number of Classes:  2
```

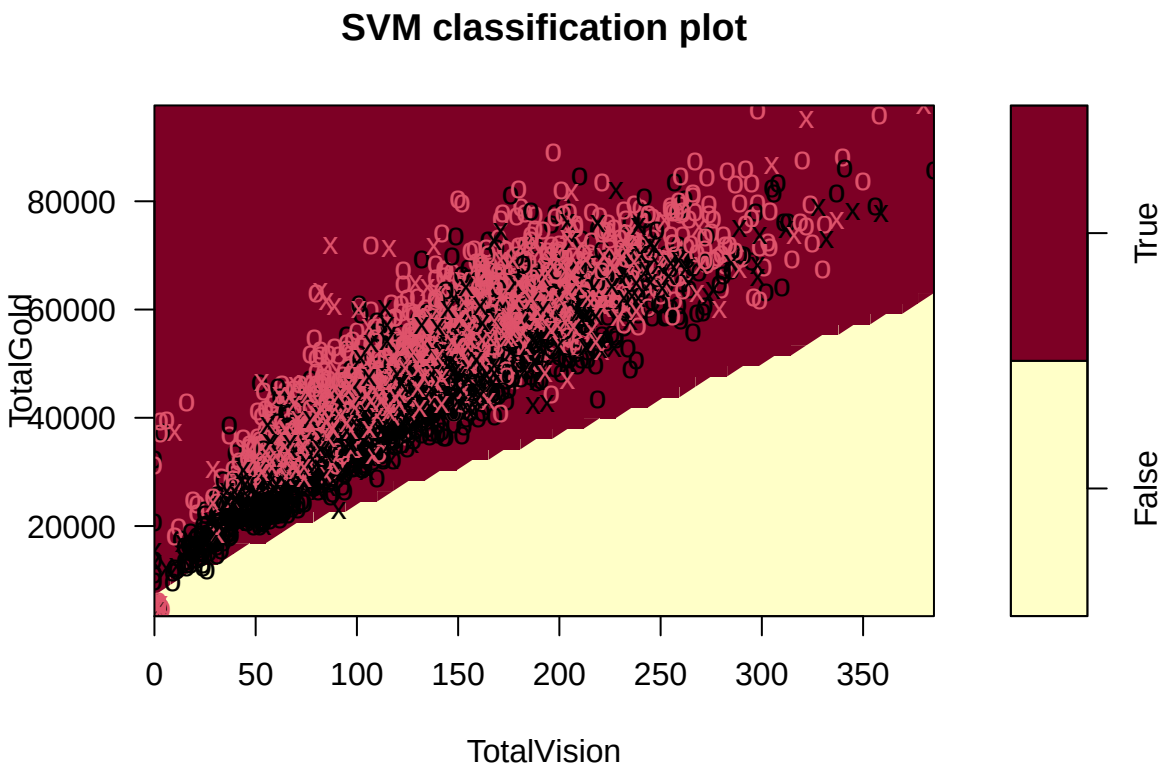
```
##
## Levels:
##  False True
```

Testing & Evaluation Now we can evaluate on the test set:

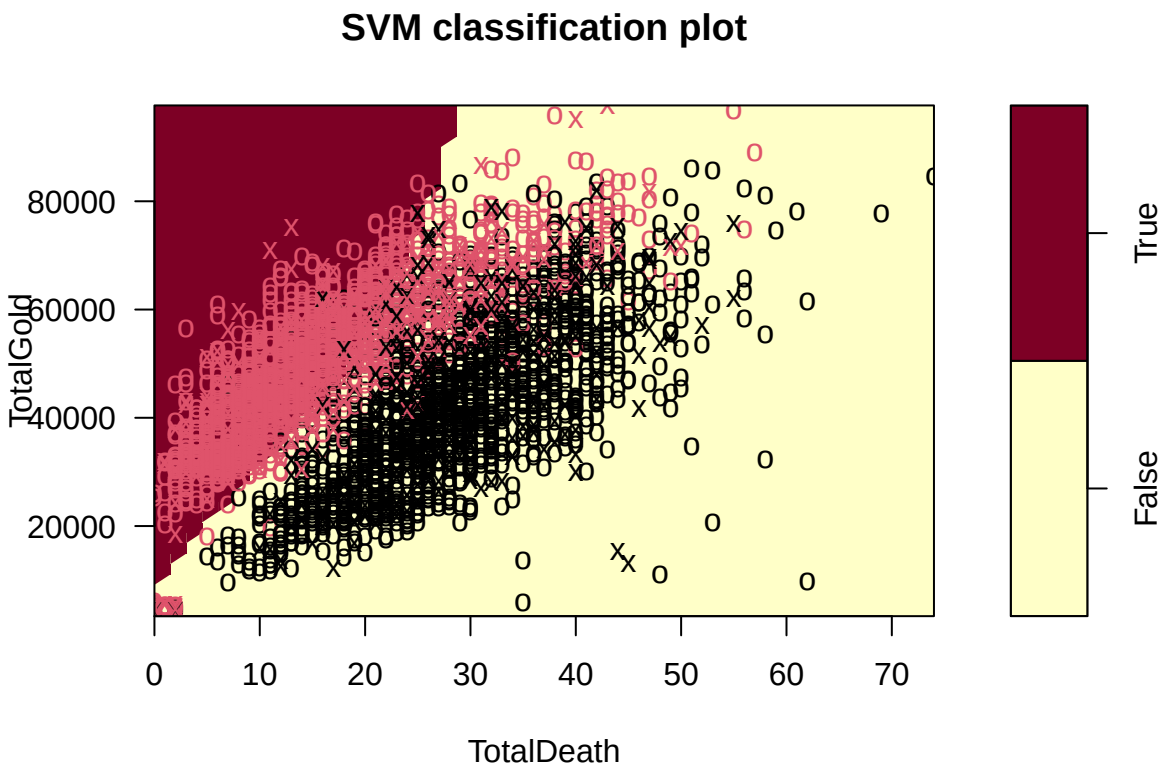
```
library(caret)
svm_probs_poly <- predict(svm_poly, newdata = full_stats_test)

confusionMatrix(svm_probs_poly, reference = full_stats_test$won)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  1728  103
##      True     69 1720
##
##                Accuracy : 0.9525
##                  95% CI : (0.945, 0.9592)
##     No Information Rate : 0.5036
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.905
##
##  Mcnemar's Test P-Value : 0.01186
##
##             Sensitivity : 0.9616
##             Specificity : 0.9435
##          Pos Pred Value : 0.9437
##          Neg Pred Value : 0.9614
##              Prevalence : 0.4964
##          Detection Rate : 0.4773
##    Detection Prevalence : 0.5058
##       Balanced Accuracy : 0.9526
##
##        'Positive' Class : False
##
```

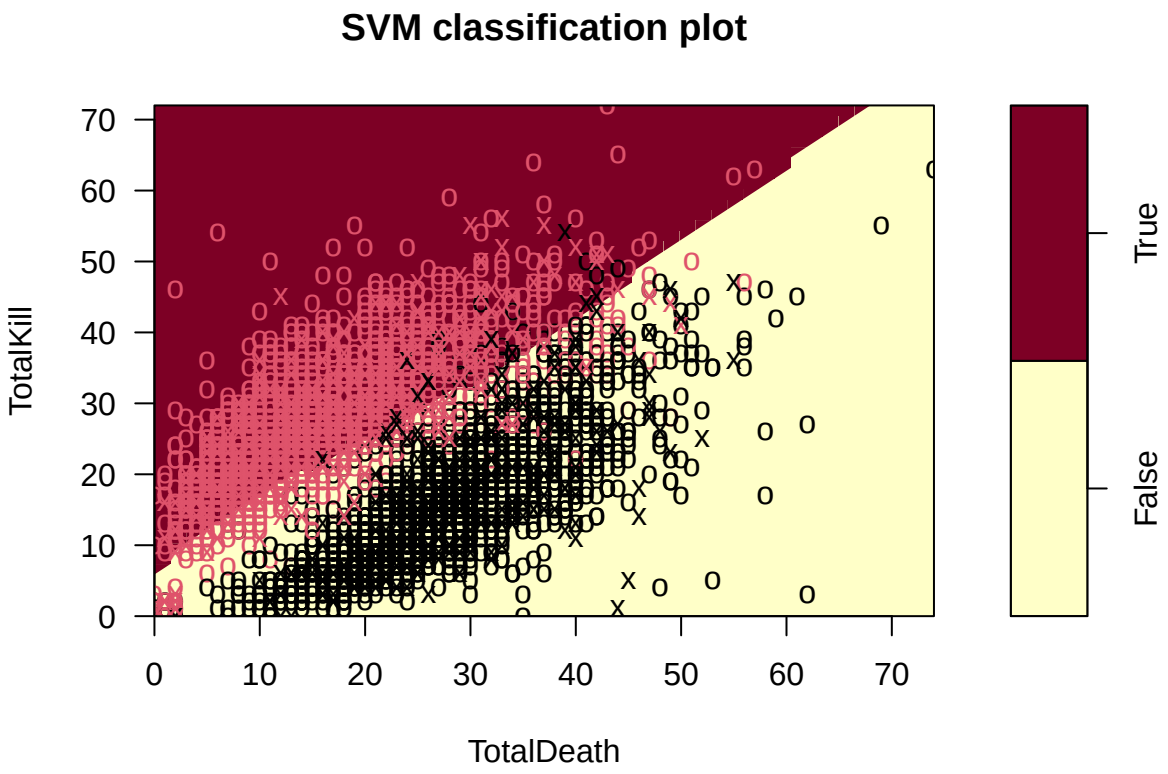
```
plot(svm_poly, full_stats_test, TotalGold ~ TotalVision)
```



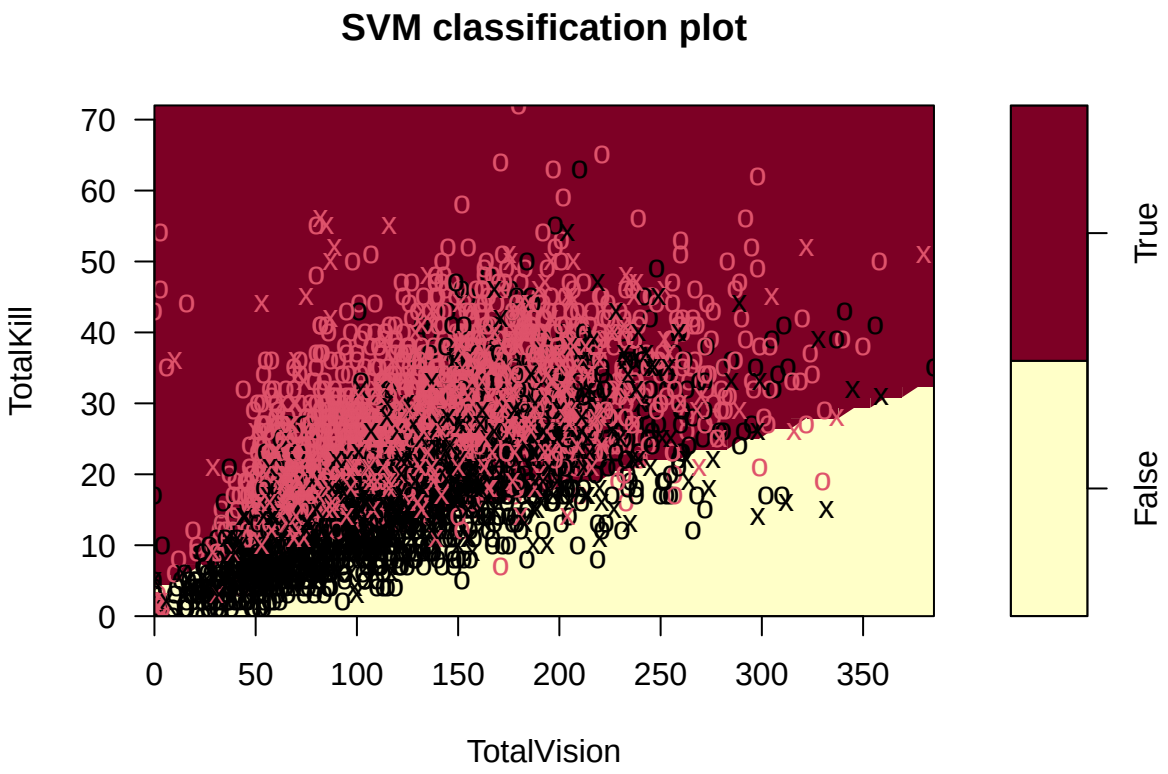

```
plot(svm_poly, full_stats_test, TotalGold ~ TotalDeath)
```



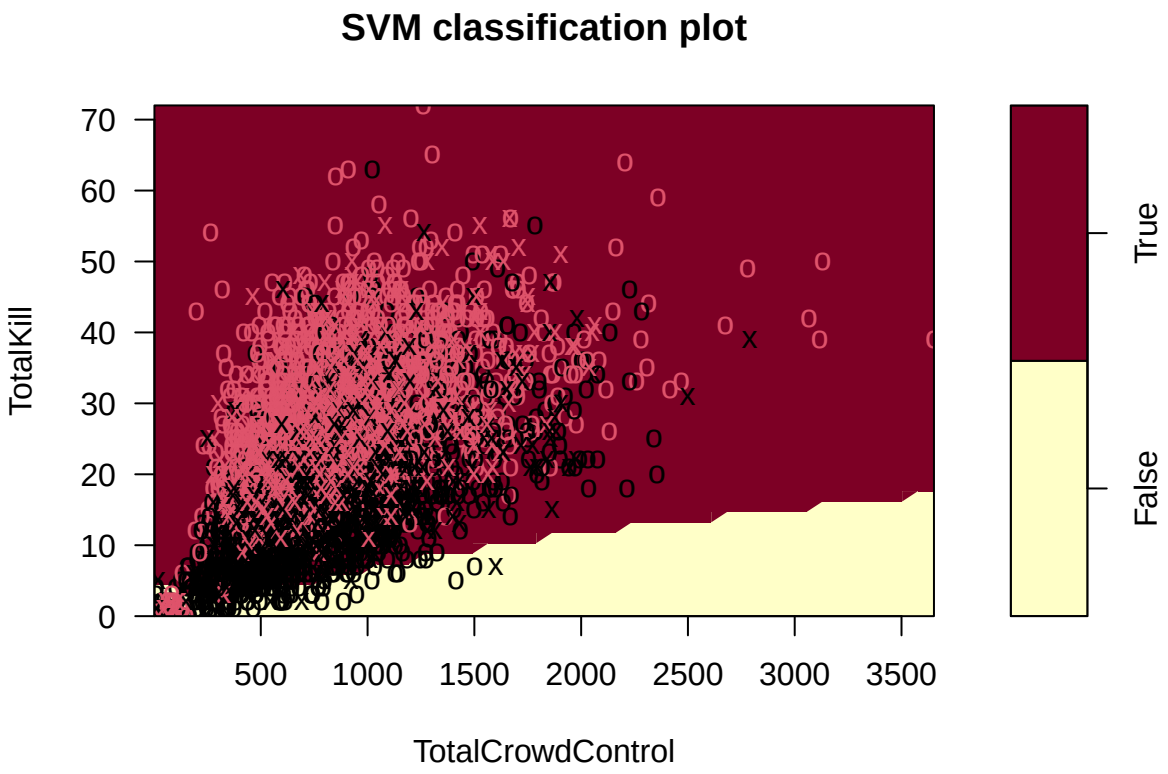

```
plot(svm_poly, full_stats_test, TotalKill ~ TotalDeath)
```



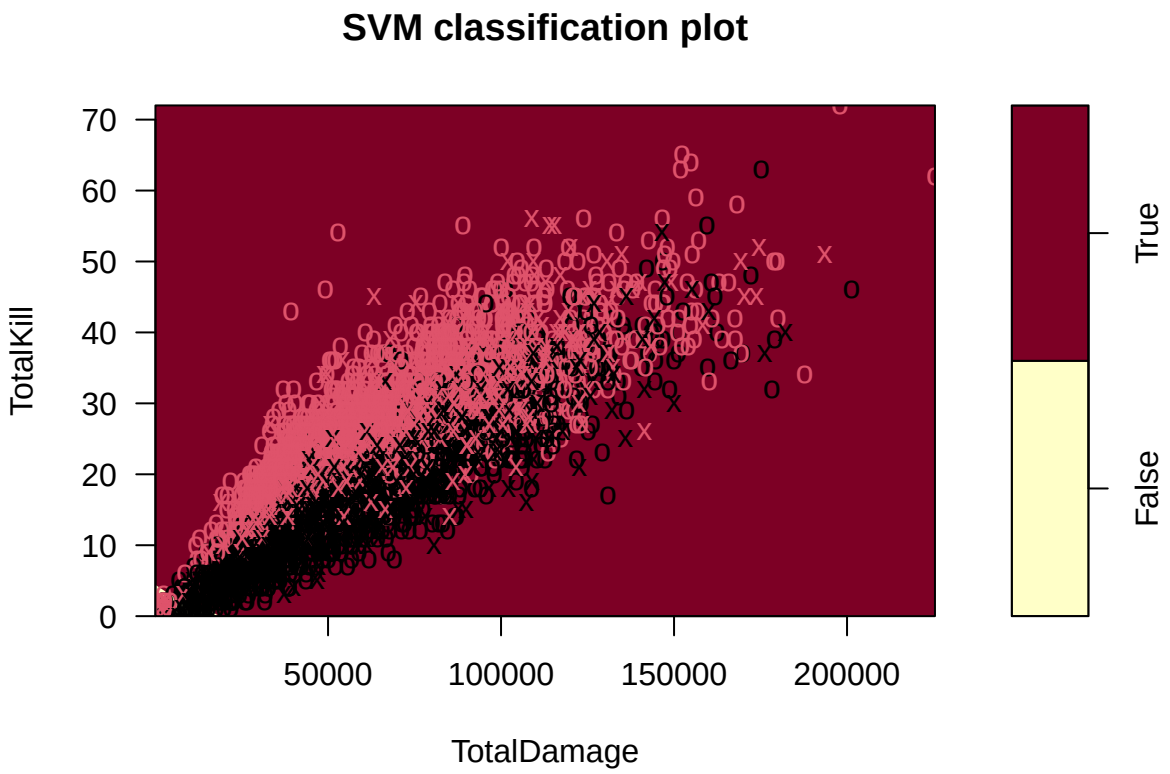

```
plot(svm_poly, full_stats_test, TotalKill ~ TotalVision)
```




```
plot(svm_poly, full_stats_test, TotalKill ~ TotalCrowdControl)
```




```
plot(svm_poly, full_stats_test, TotalKill ~ TotalDamage)
```



Tuning

```
set.seed(1010)
tune_out <- tune(svm, won~., data=full_stats_validate, kernel="poly",
                 ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(tune_out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 0.04337017
##
## - Detailed performance results:
##    cost      error  dispersion
## 1 1e-03 0.29033149 0.053903987
## 2 1e-02 0.12320442 0.016482247
## 3 1e-01 0.05165746 0.011504603
## 4 1e+00 0.04392265 0.008282176
## 5 5e+00 0.04364641 0.009096935
## 6 1e+01 0.04392265 0.011752512
## 7 1e+02 0.04337017 0.009394973
```

```
best_poly_model <- tune_out$best.model
summary(best_poly_model)
```

Getting Best Model after Tune

```
##
## Call:
## best.tune(method = svm, train.x = won ~ ., data = full_stats_validate,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)), kernel = "poly")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  100
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  582
##
##  ( 289 293 )
##
##
## Number of Classes:  2
##
## Levels:
##  False True
```

```
best_poly_pred <- predict(best_poly_model, newdata=full_stats_test)
acc_best_poly <- mean(best_poly_pred==full_stats_test$won)
```

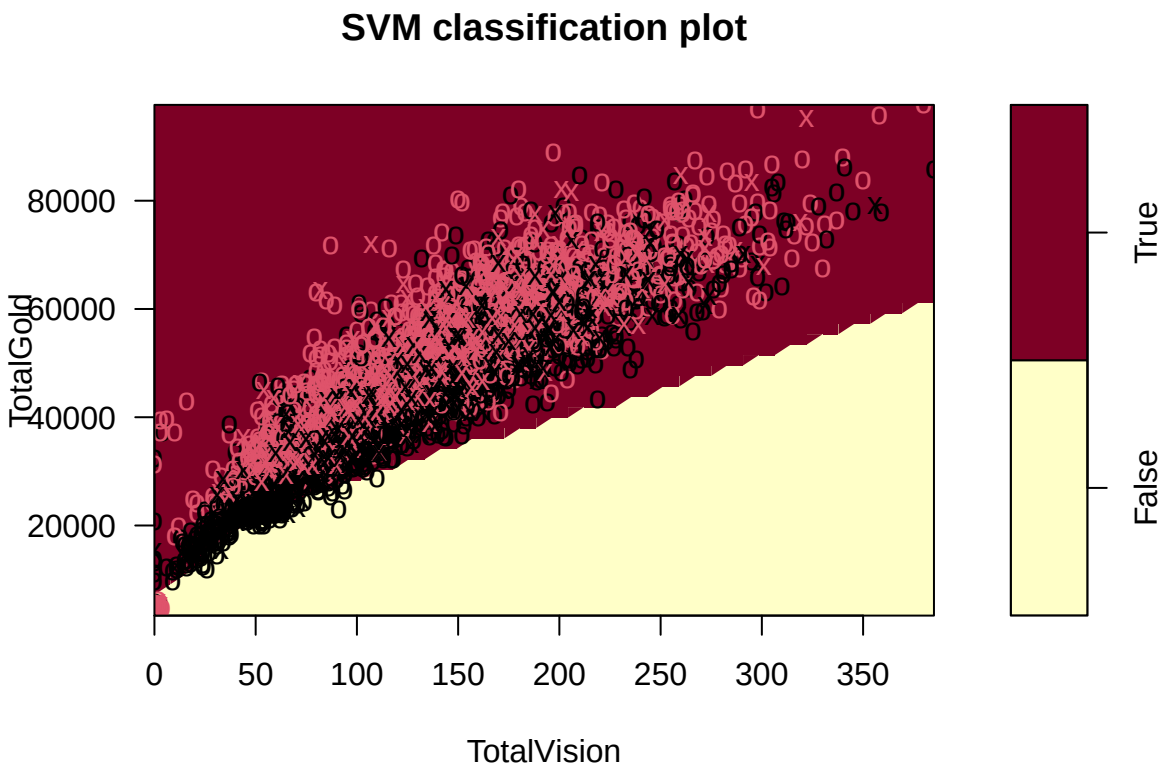
```
confusionMatrix(best_poly_pred, reference = full_stats_test$won)
```

Evaluating Best Model

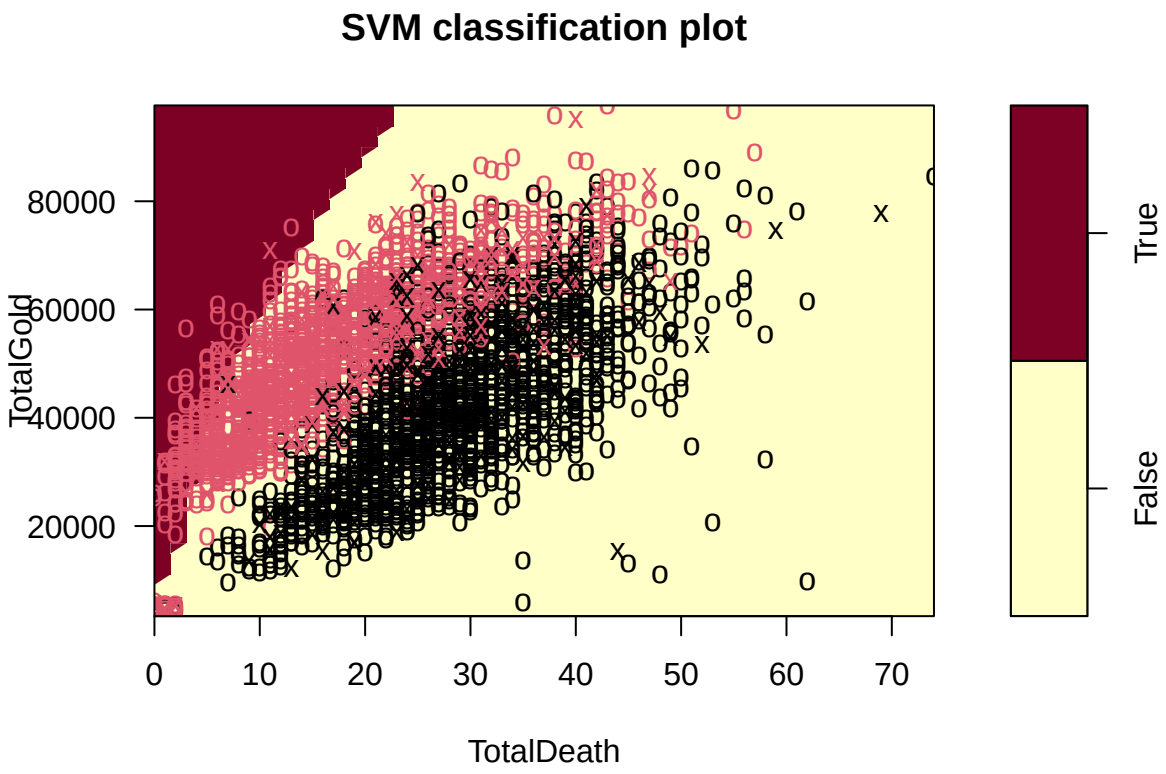
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  1703   93
##      True     94 1730
##
##                Accuracy : 0.9483
##                  95% CI : (0.9406, 0.9553)
##     No Information Rate : 0.5036
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8967
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9477
##             Specificity : 0.9490
```

```
##          Pos Pred Value : 0.9482
##          Neg Pred Value : 0.9485
##              Prevalence : 0.4964
##          Detection Rate : 0.4704
##    Detection Prevalence : 0.4961
##       Balanced Accuracy : 0.9483
##
##        'Positive' Class : False
##
```

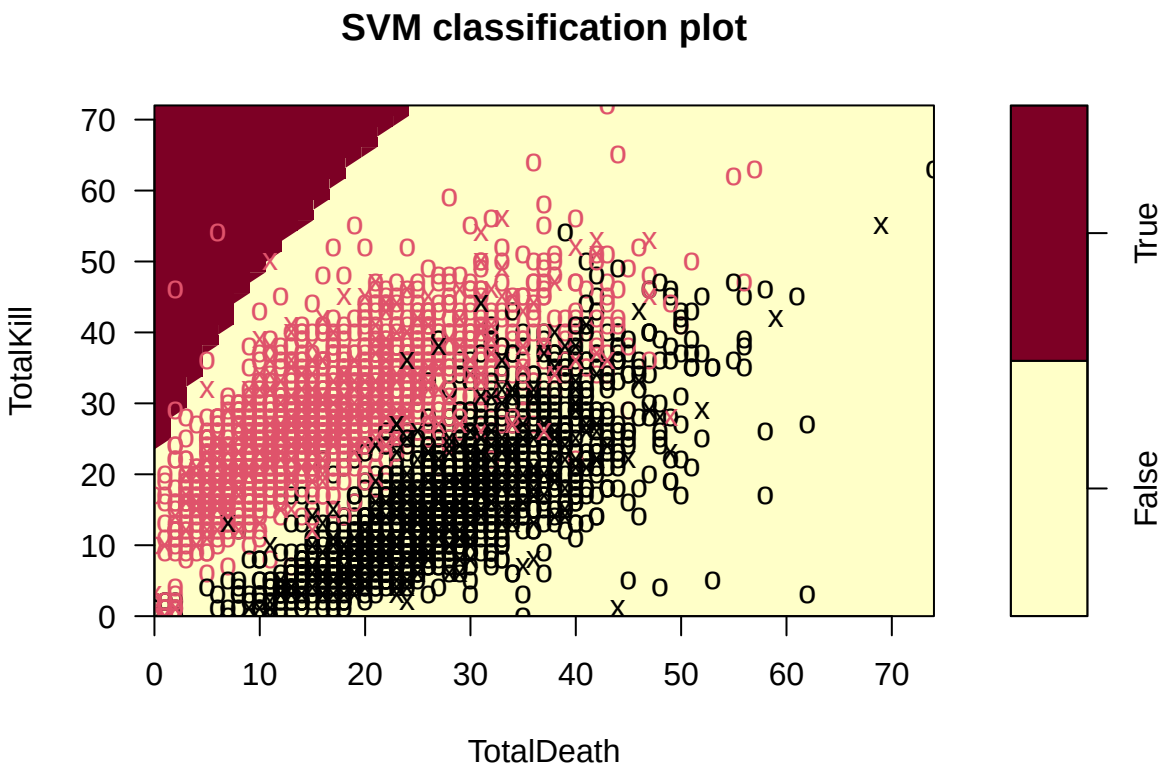
```
plot(best_poly_model, full_stats_test, TotalGold ~ TotalVision)
```



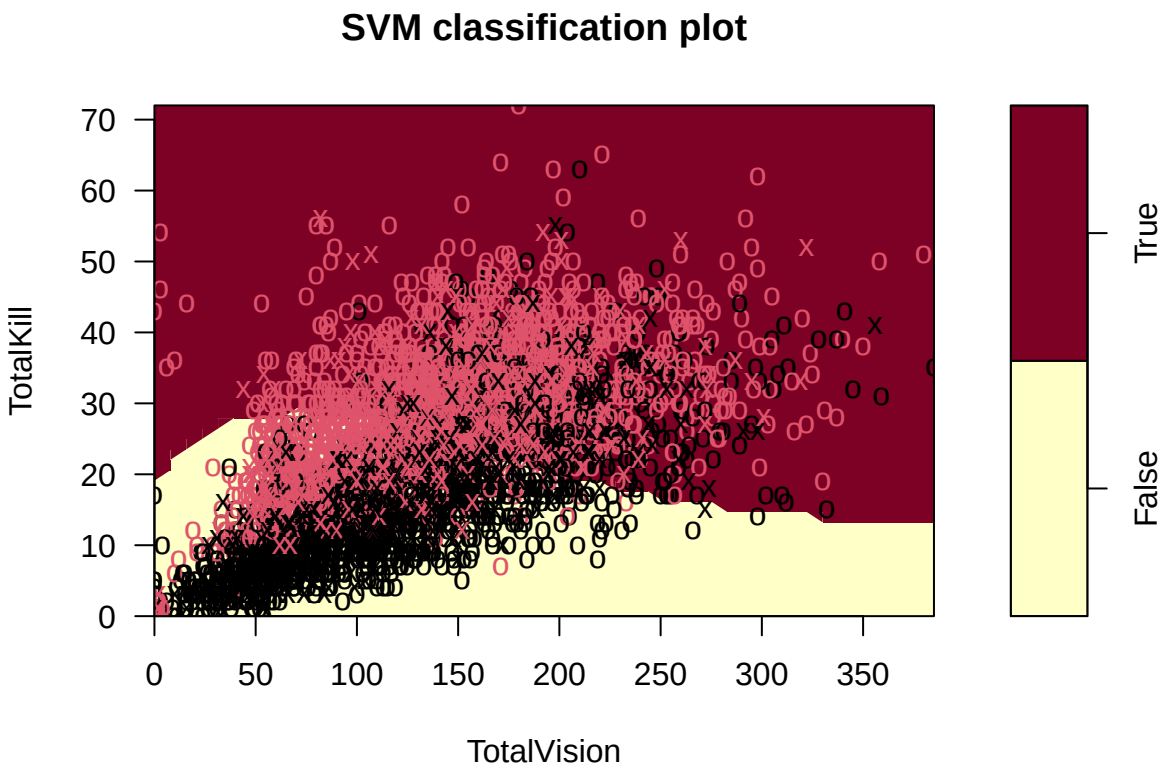

```
plot(best_poly_model, full_stats_test, TotalGold ~ TotalDeath)
```



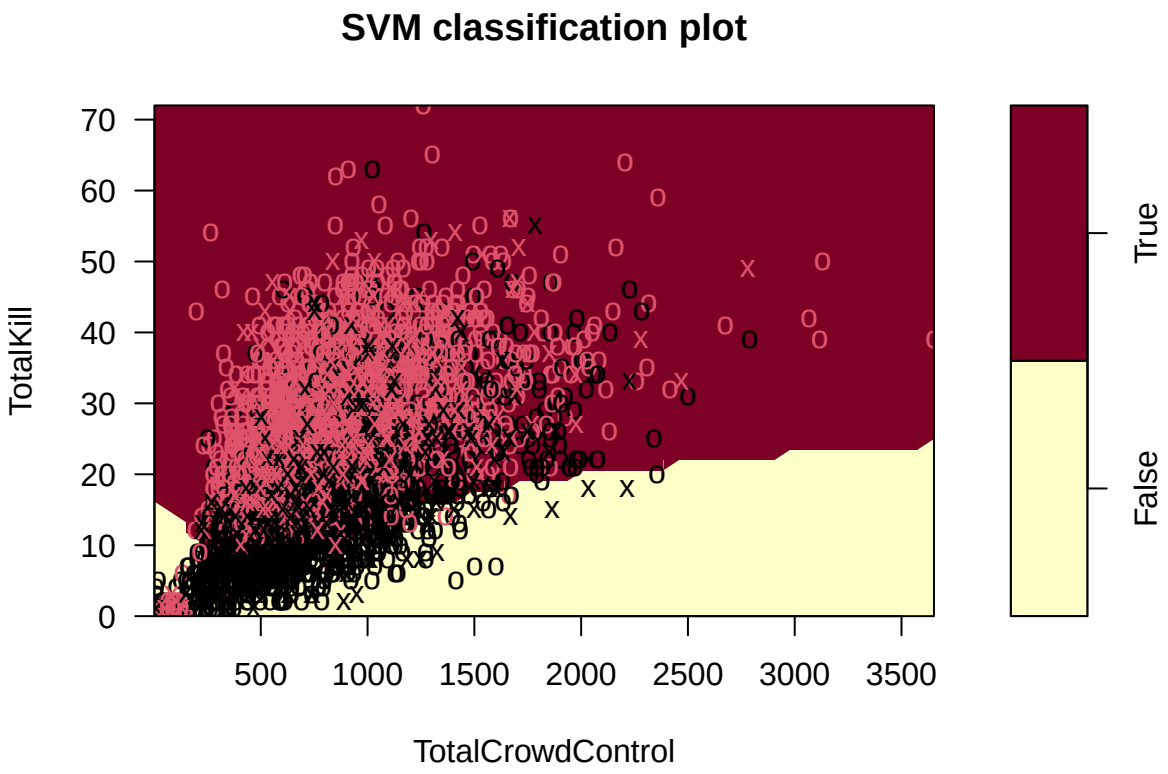

```
plot(best_poly_model, full_stats_test, TotalKill ~ TotalDeath)
```



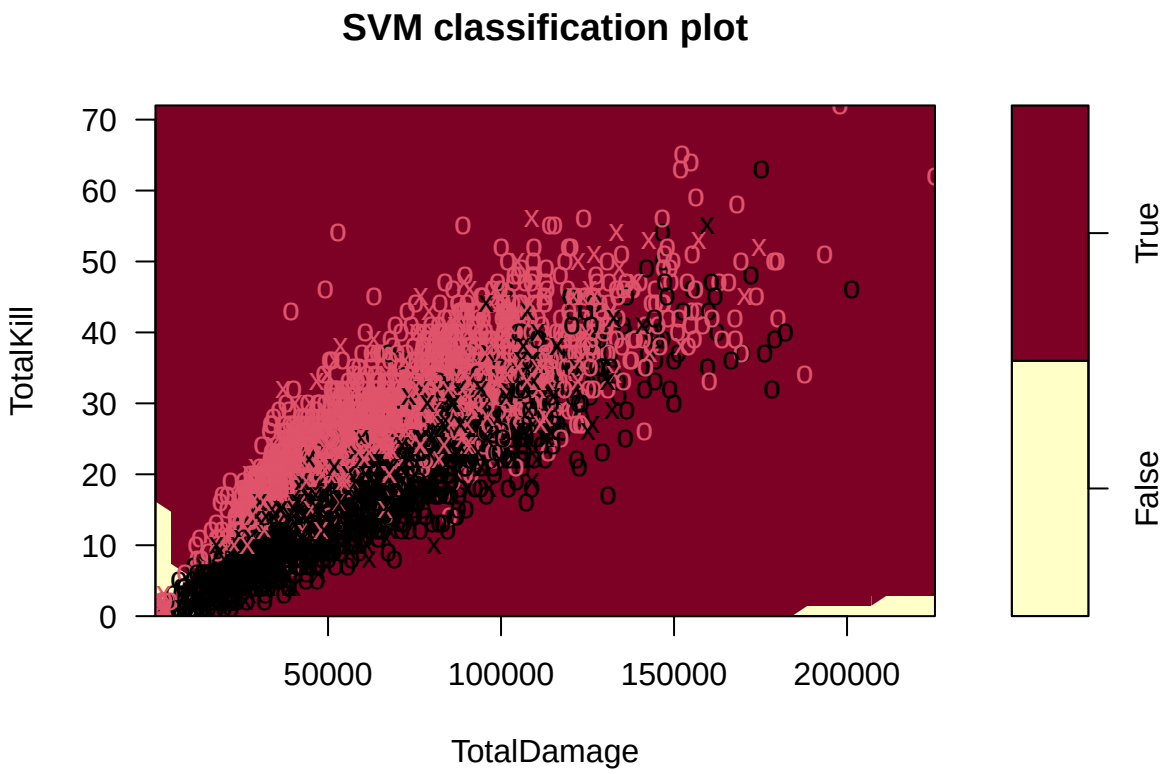

```
plot(best_poly_model, full_stats_test, TotalKill ~ TotalVision)
```




```
plot(best_poly_model, full_stats_test, TotalKill ~ TotalCrowdControl)
```




```
plot(best_poly_model, full_stats_test, TotalKill ~ TotalDamage)
```



Radial

Training Now we will build our SVM Radial model

```
library(e1071)
svm_rad <- svm(won~., data=full_stats_train, kernel="radial", cost=10, scale=TRUE)
summary(svm_rad)
```

```
##
## Call:
## svm(formula = won ~ ., data = full_stats_train, kernel = "radial",
##     cost = 10, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  1163
##
##  ( 574 589 )
##
##
## Number of Classes:  2
##
## Levels:
```

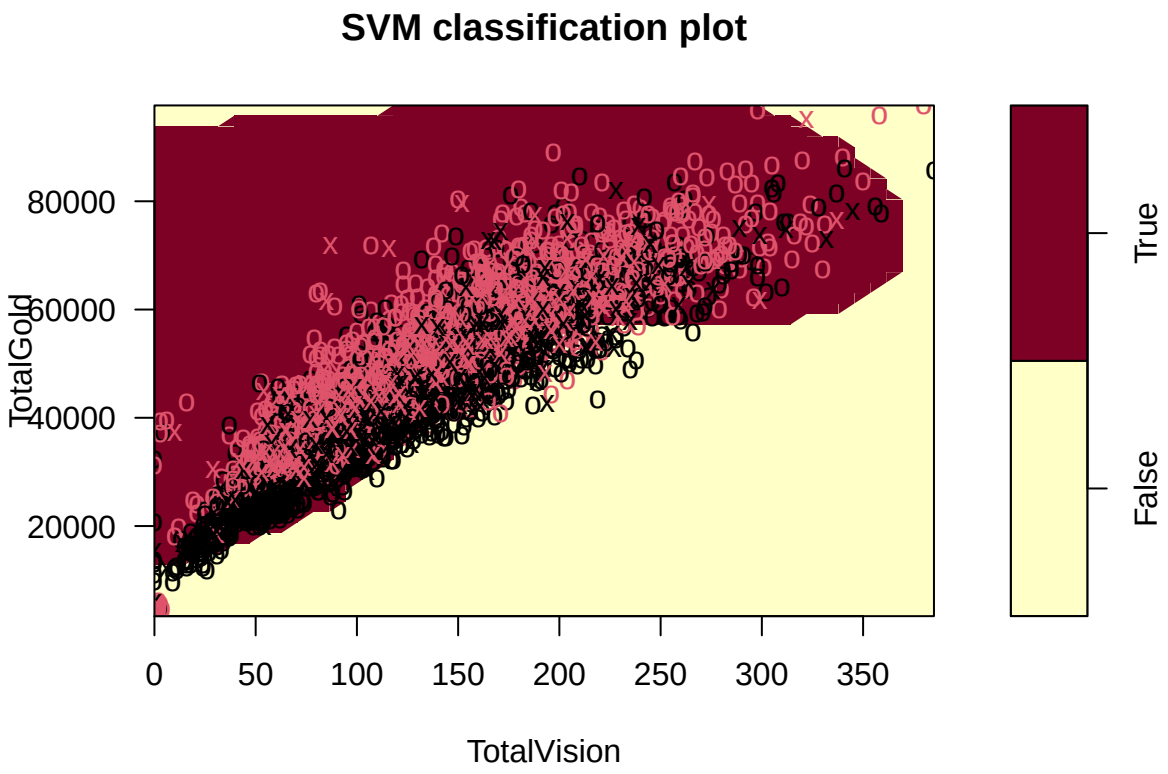
```
##  False True
```

Testing & Evaluation Now we can evaluate on the test set:

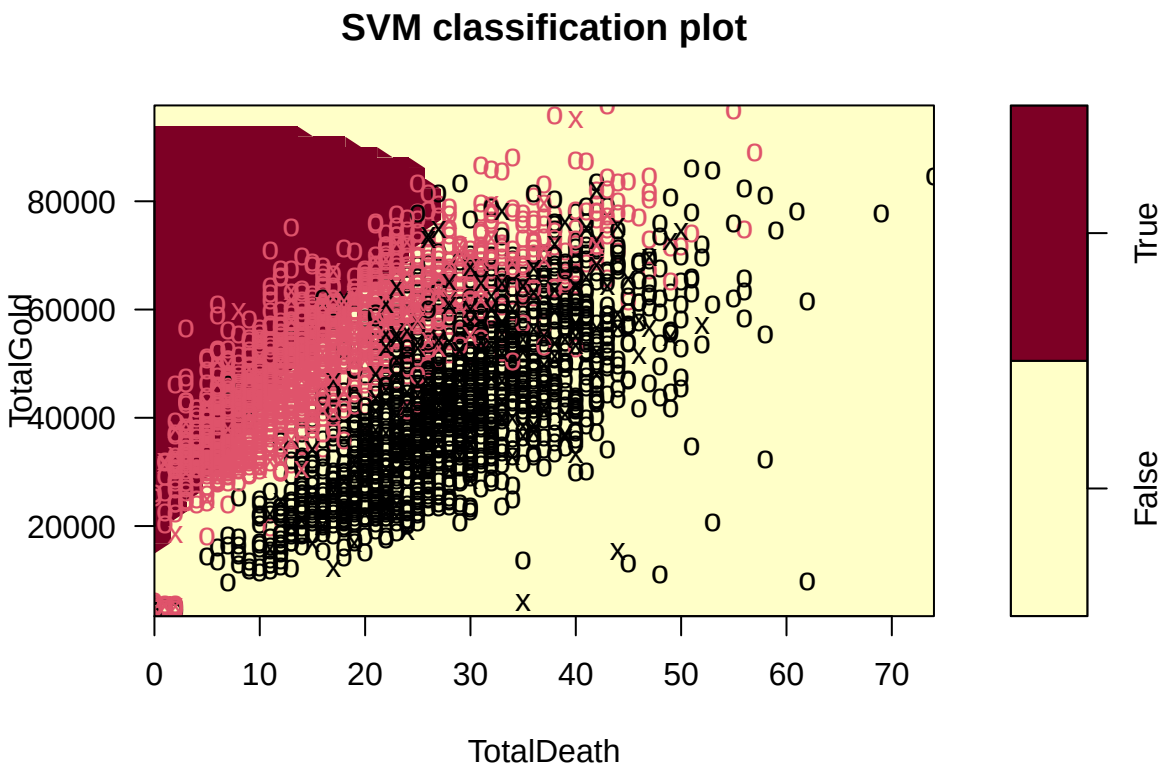
```
library(caret)
svm_probs_rad <- predict(svm_rad, newdata = full_stats_test)

confusionMatrix(svm_probs_rad, reference = full_stats_test$won)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  1706   94
##      True     91 1729
##
##                Accuracy : 0.9489
##                  95% CI : (0.9412, 0.9558)
##     No Information Rate : 0.5036
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8978
##
##  Mcnemar's Test P-Value : 0.8831
##
##             Sensitivity : 0.9494
##             Specificity : 0.9484
##          Pos Pred Value : 0.9478
##          Neg Pred Value : 0.9500
##              Prevalence : 0.4964
##          Detection Rate : 0.4713
##    Detection Prevalence : 0.4972
##       Balanced Accuracy : 0.9489
##
##        'Positive' Class : False
##
```

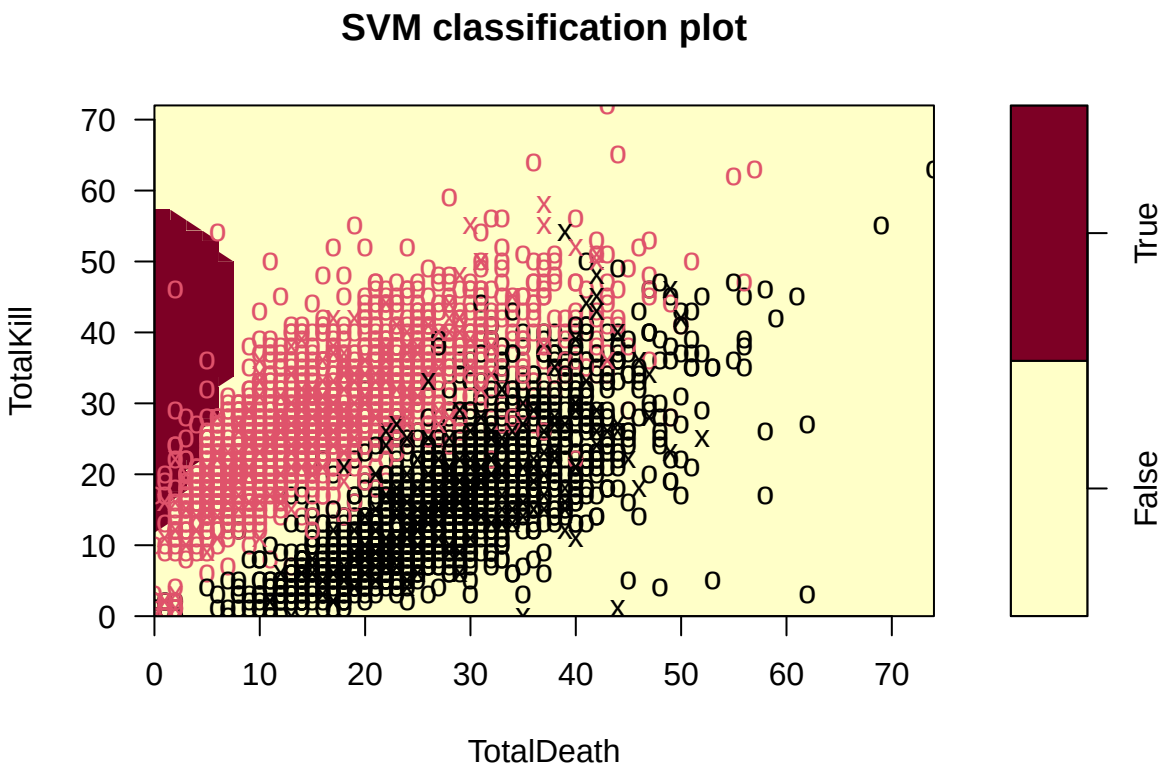
```
plot(svm_rad, full_stats_test, TotalGold ~ TotalVision)
```



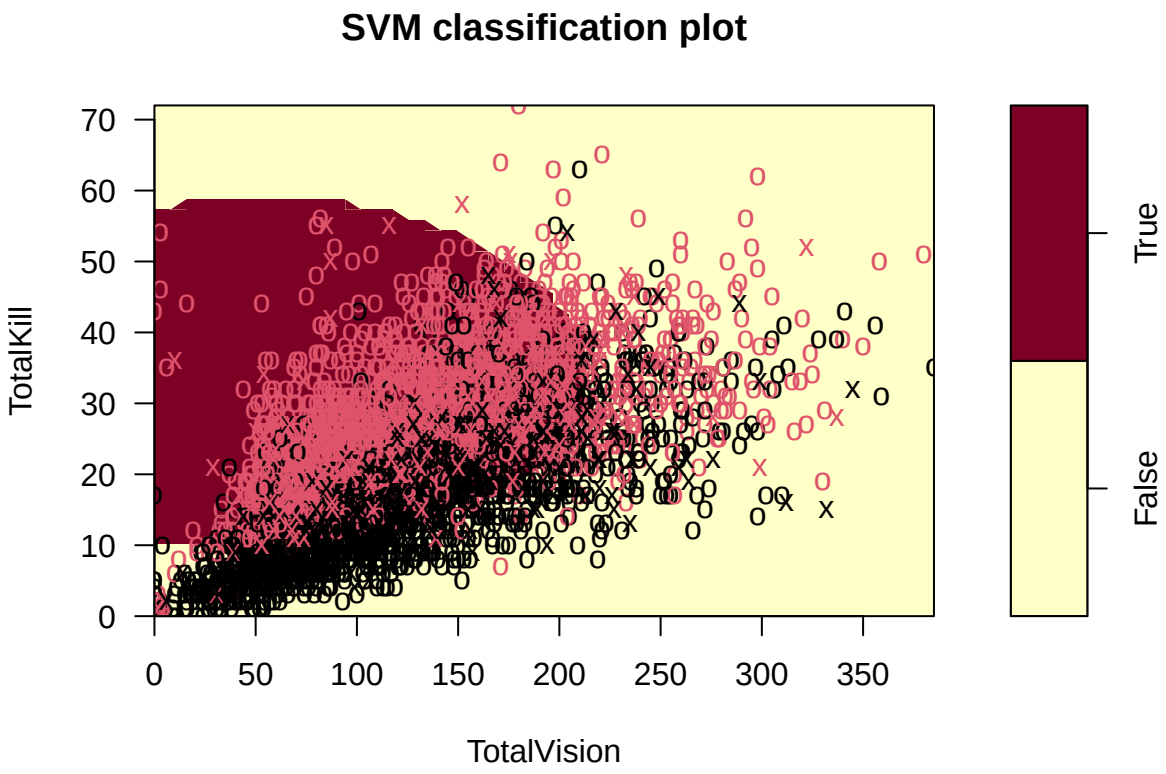

```
plot(svm_rad, full_stats_test, TotalGold ~ TotalDeath)
```



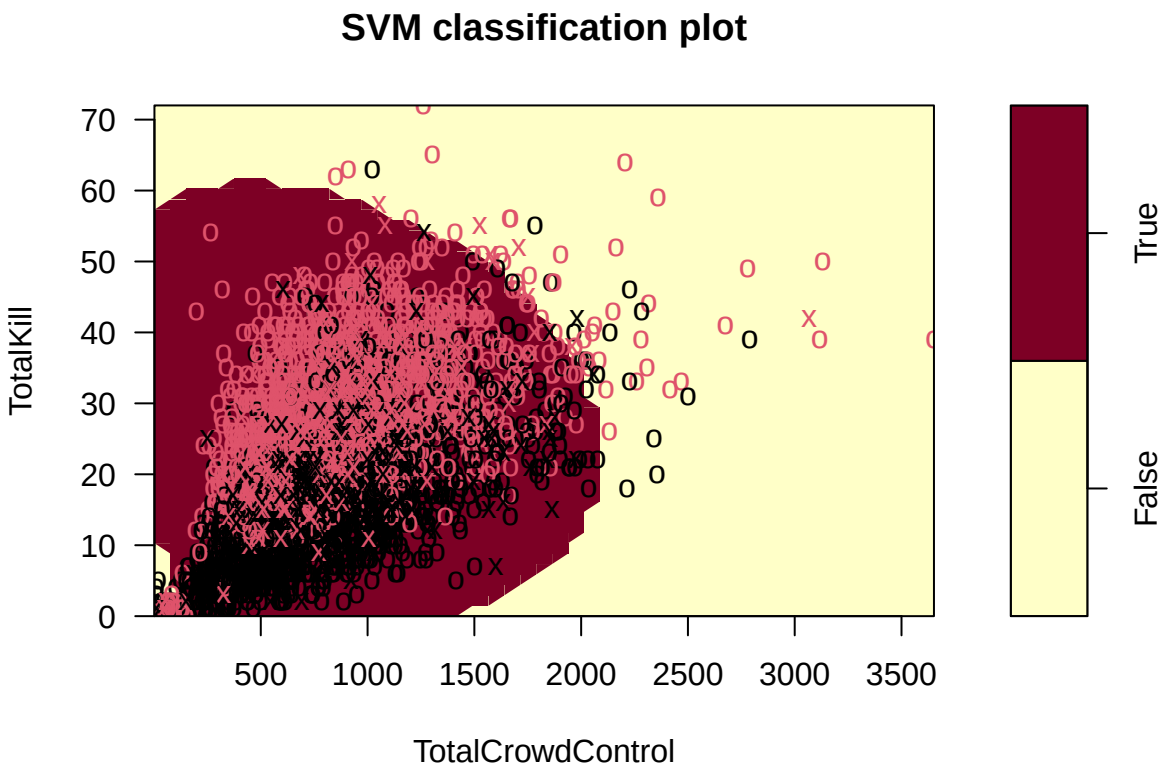

```
plot(svm_rad, full_stats_test, TotalKill ~ TotalDeath)
```



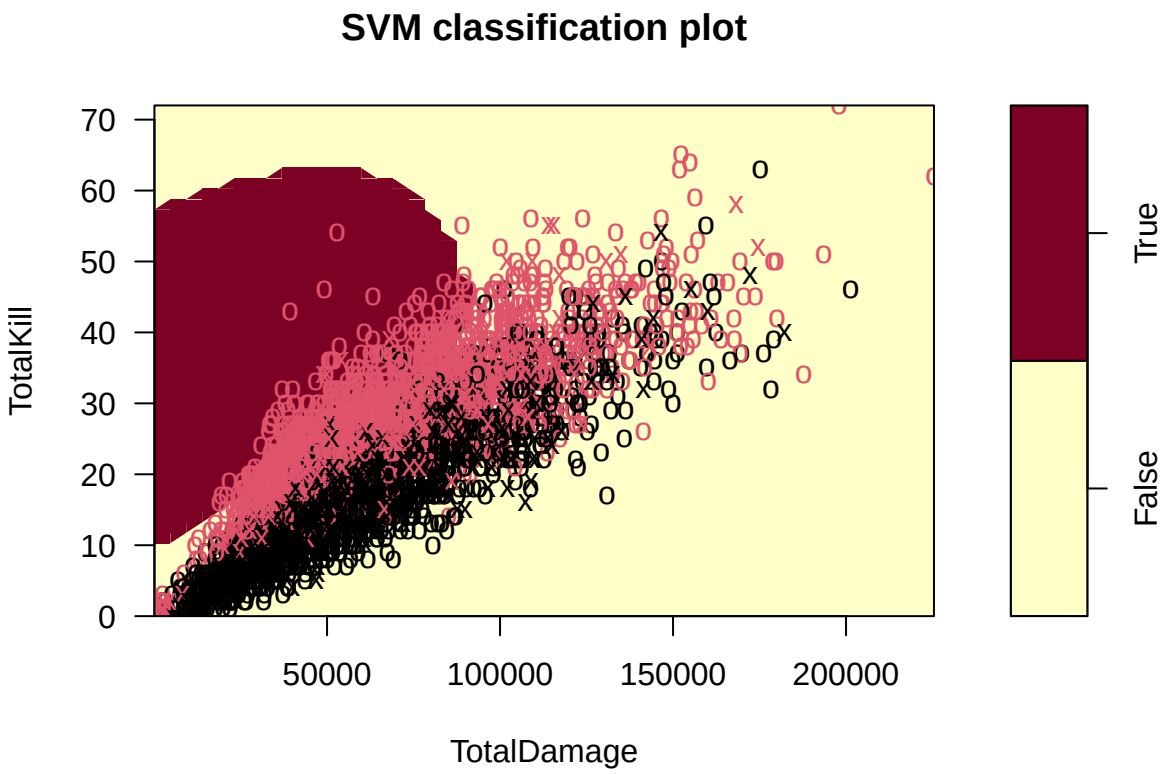

```
plot(svm_rad, full_stats_test, TotalKill ~ TotalVision)
```




```
plot(svm_rad, full_stats_test, TotalKill ~ TotalCrowdControl)
```




```
plot(svm_rad, full_stats_test, TotalKill ~ TotalDamage)
```



Tuning

```
set.seed(1010)
tune_out <- tune(svm, won~., data=full_stats_validate, kernel="radial",
                 ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(tune_out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.04558011
##
## - Detailed performance results:
##    cost      error  dispersion
## 1 1e-03 0.49281768 0.024018856
## 2 1e-02 0.05911602 0.011441786
## 3 1e-01 0.04972376 0.008439366
## 4 1e+00 0.04558011 0.010191503
## 5 5e+00 0.04696133 0.009299728
## 6 1e+01 0.04613260 0.007372235
## 7 1e+02 0.04917127 0.005794524
```

```
best_rad_model <- tune_out$best.model
summary(best_rad_model)
```

Getting Best Model after Tune

```
##
## Call:
## best.tune(method = svm, train.x = won ~ ., data = full_stats_validate,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  539
##
##  ( 267 272 )
##
##
## Number of Classes:  2
##
## Levels:
##  False True
```

```
best_rad_pred <- predict(best_rad_model, newdata=full_stats_test)
acc_best_rad <- mean(best_rad_pred==full_stats_test$won)
```

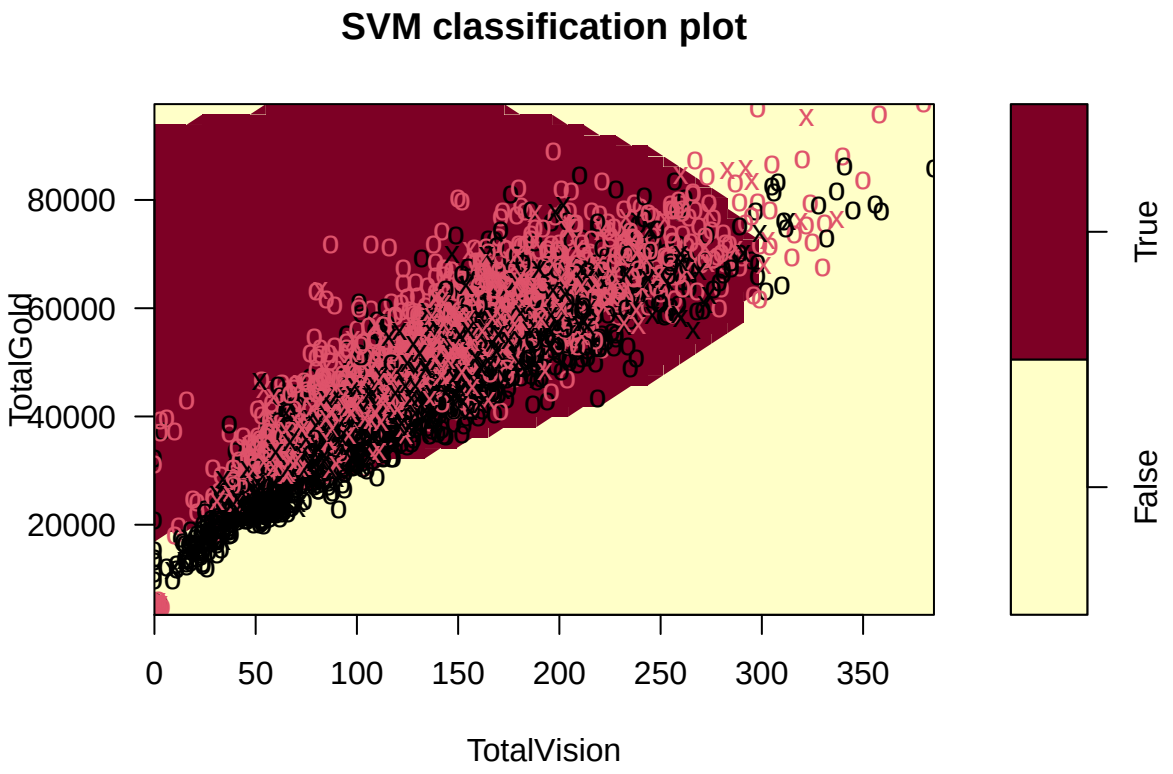
```
confusionMatrix(best_rad_pred, reference = full_stats_test$won)
```

Evaluating Best Model

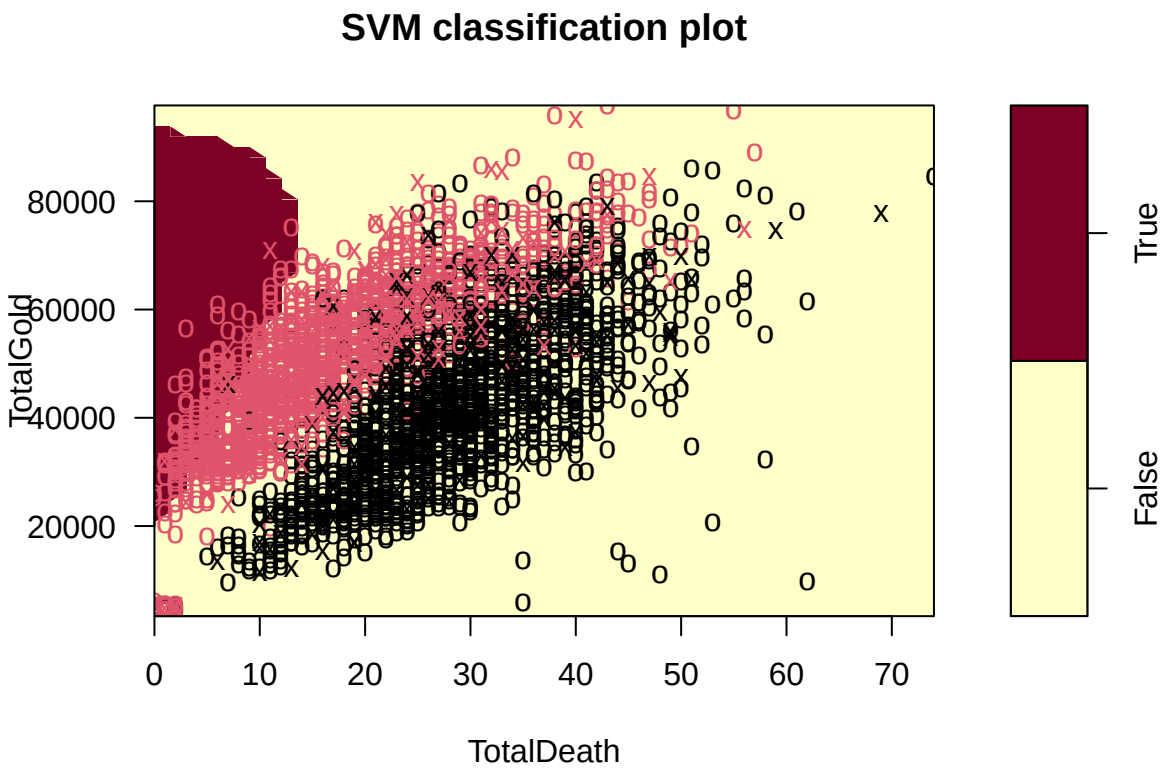
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  1717   93
##      True     80 1730
##
##                Accuracy : 0.9522
##                  95% CI : (0.9447, 0.9589)
##     No Information Rate : 0.5036
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9044
##
##  Mcnemar's Test P-Value : 0.3616
##
##             Sensitivity : 0.9555
##             Specificity : 0.9490
##          Pos Pred Value : 0.9486
##          Neg Pred Value : 0.9558
```

```
##              Prevalence : 0.4964
##          Detection Rate : 0.4743
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.9522
##
##        'Positive' Class : False
##
```

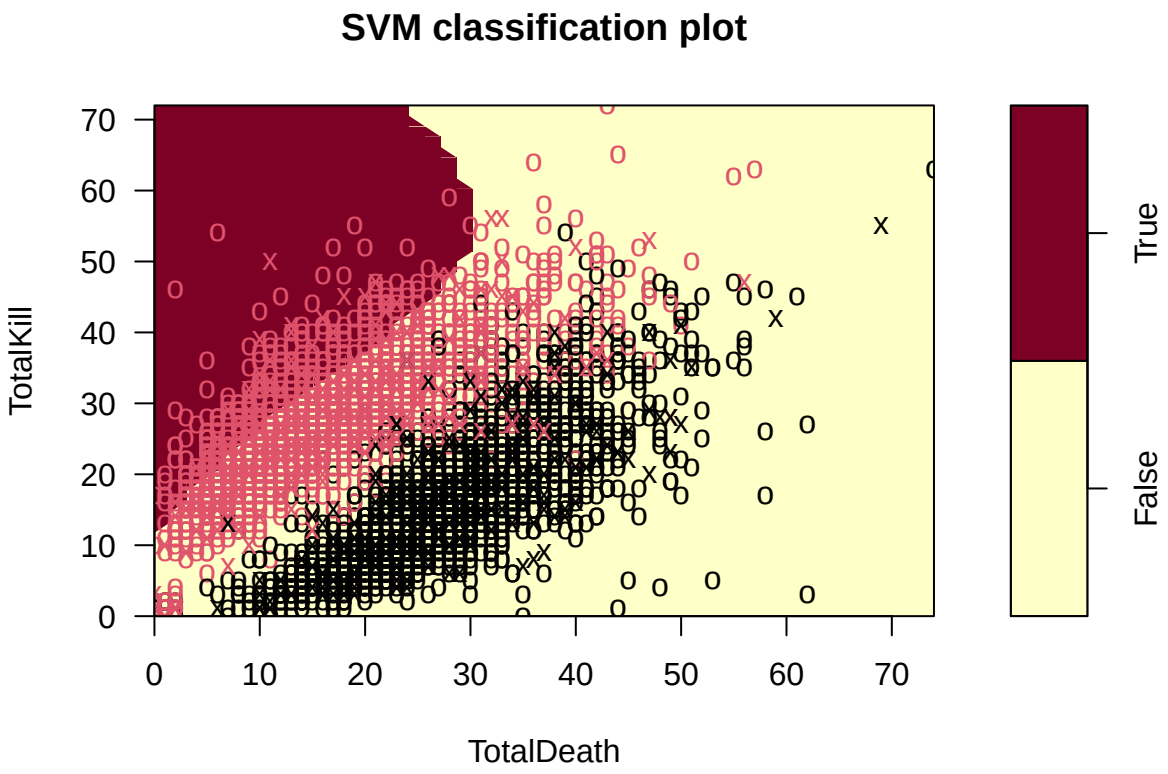
```
plot(best_rad_model, full_stats_test, TotalGold ~ TotalVision)
```



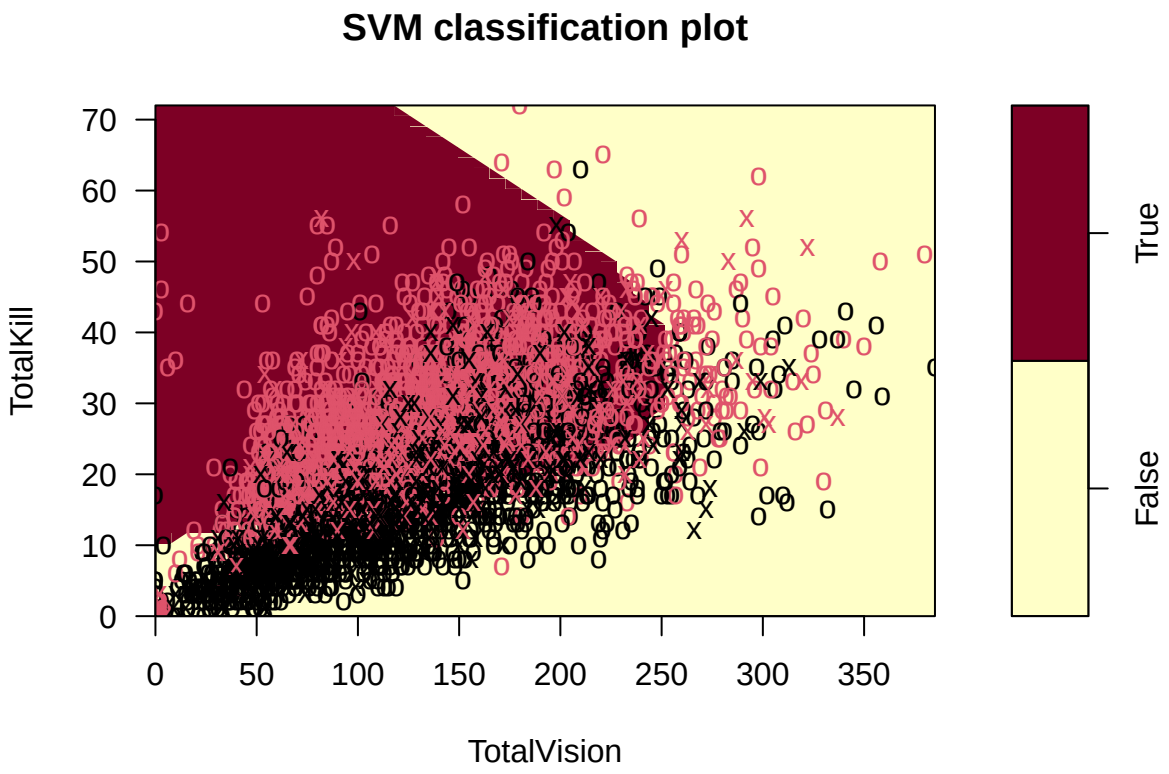

```
plot(best_rad_model, full_stats_test, TotalGold ~ TotalDeath)
```



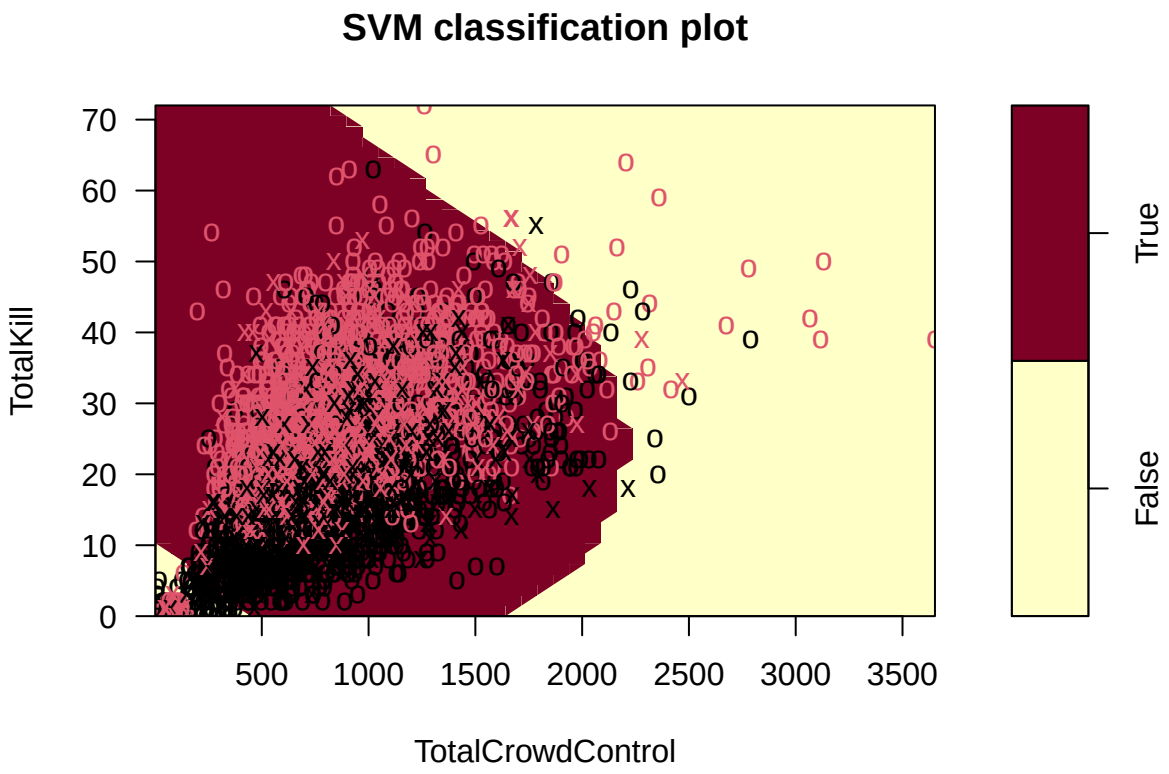

```
plot(best_rad_model, full_stats_test, TotalKill ~ TotalDeath)
```



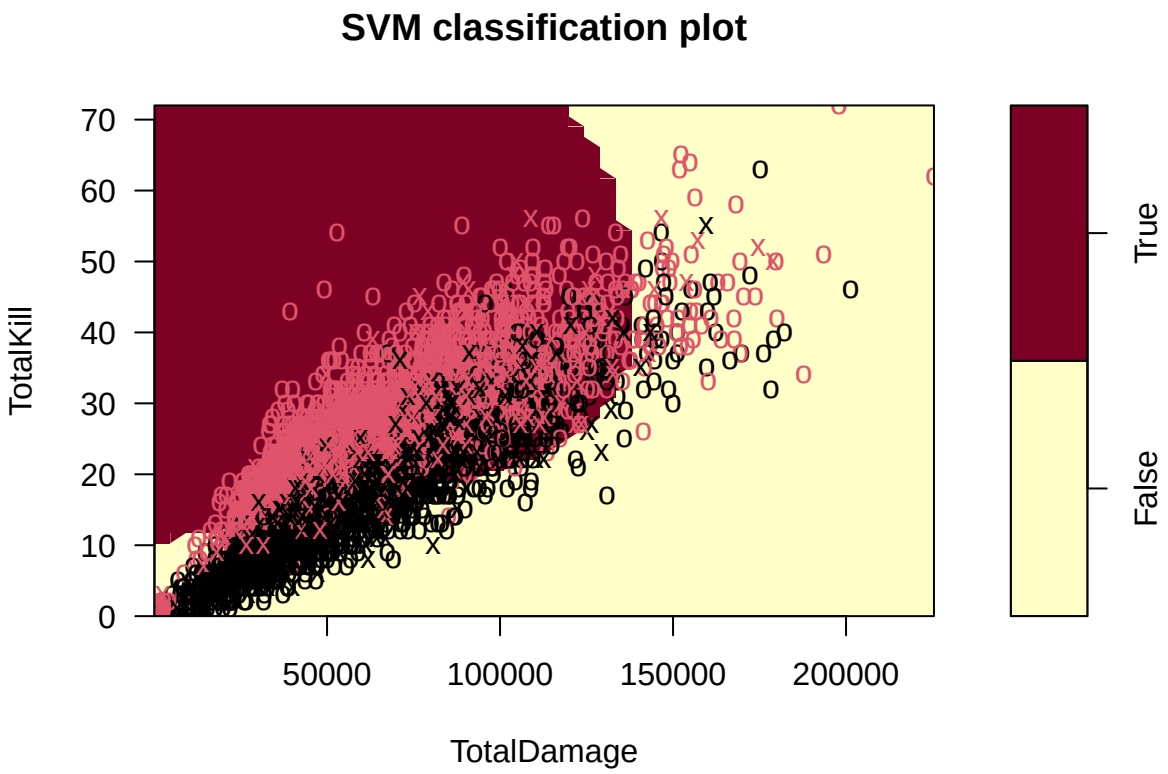

```
plot(best_rad_model, full_stats_test, TotalKill ~ TotalVision)
```




```
plot(best_rad_model, full_stats_test, TotalKill ~ TotalCrowdControl)
```




```
plot(best_rad_model, full_stats_test, TotalKill ~ TotalDamage)
```



SVM Linear vs Polynomial vs Radial

Analyzing the results of each model based on the algorithms.

SVM Linear

Our SVM Linear model utilized 371 support vectors along the margin. It had an accuracy of 95.47%, the balanced accuracy is 95.48%. This is a fairly high accuracy, which isn't surprising to me because our data as seen in the graphs is very linear and could be split easily using a linear function. The sensitivity is about .96 and the specificity is about .95 which are both close to 1 and very good. The p-value is also very low which shows this a good model. The Kappas is about .91, which shows a excellent positive agreement. Overall, this model has very good metrics and is a very good model for our data. The linear model utilizes a linear decision boundary, which for most of our data as seen by the graphs is very effective.

SVM Polynomial

Our SVM Linear model utilized 824 support vectors along the margin. It had an accuracy of 95.22%, the balanced accuracy is 95.24%. This is a fairly high accuracy, which isn't surprising to me because our data as seen in the graphs is very linear and could be split easily using a polynomial function. The sensitivity is about .96 and the specificity is about .94 which are both close to 1 and very good. The p-value is also very low which shows this a good model. The Kappas is about .91, which shows a excellent positive agreement. Overall, this model has very good metrics and is a very good model for our data. The linear model utilizes a polynomial decision boundary, which for most of our data as seen by the graphs is very effective.

SVM Radial

Our SVM Linear model utilized 522 support vectors along the margin. It had an accuracy of 95.58%, the balanced accuracy is 95.60%. This is a fairly high accuracy, which isn't surprising to me because our data as seen in the graphs is very linear and could be split easily using a polynomial function. The sensitivity is about .97 and the specificity is about .95 which are both close to 1 and very good. The p-value is also very low which shows this a good model. The Kappas is about .91, which shows a excellent positive agreement. Overall, this model has very good metrics and is a very good model for our data. The linear model utilizes a radial decision boundary, with an additional hyperparameter gamma to control the shape. After tuning, it was very effective for most of our data as seen by the graphs.

Summary

The SVM polynomial model had the most support vectors but the lowest accuracy of all the models. The radial model had the highest accuracy, with linear not far behind, and the polynomial model being the least. However, the difference between them is less than .5%. The Kappa is pretty much the same for all of them, as well as the specificity and sensitivity. Overall it was surprising to me that the radial model was the most accurate. However, by looking at the graphs you can see how well of it fit it really is. Although the differences in accuracy are very small. Overall, I think all of these models represent and predict the data well. However, it is unfortunate that I couldn't use more of the data set because of the time complexity of SVM. Overall the SVM algorithms outperformed the initial logistic regression model created during data exploration, but it wasn't by a landslide as our model is very ideal for logistic regression.