Bookbinders Case Study

Austin Vanderlyn, Christine Kelly, Richard Tarbell

2/16/2022

I. Executive Summary - Richard

LM

Logit

SVM

II. The Problem - Austin

III. Review of Related Literature - Christine

Linear Regression is a parametric model used to find if any explanatory variables influence the response variable. There are four assumptions for linear regression. The first is the linear relationship between X and Y. The second is normal distribution. The third is homoscedasticity. Finally, all observations must be independent of each other. The model finds the best line to predict the behavior of Y, when X is increased by 1 unit. Y must be continuous for the model to work.

Logistic Regression is used to predict the probability of an outcome based on given variables, or to see how variables are related to the outcome. For the logistic model Y is binary and not continuous. Interpretation of the logistic model is not as straightforward and requires an understanding of odds and odds ratio.

Support Vector Machine (SVM) is a popular model that was developed in the 1990s by Vladimir Vapnik and Corinna Cortes. This model finds a hyperplane that separates the data as well as possible and allows some misclassification. To accommodate a non-linear boundary between classes, SVM enlarges the feature space using polynomial terms. The SVM enlarges this feature space, using kernel tricks, in a way that is efficient with these computations.

All three of these models are beneficial in providing insights and predictions when applied to marketing campaign selections.

IV. Methodologies

The first model attempted was a linear regression. This model is not useful for this dataset because the response variable in linear regression must be continuous. In this case the response variable Choice is categorical and must be converted to a factor. If the regression model is used leaving the Choice as numeric, the information it provides is not useful for the criteria we are trying to meet.

Along with the Linear and Logit model we performed tests using the SVM models. Some advantages of using SVM are its versatility when working with high dimensional spaces and it's ability to be memory efficient. The SVM model used for the BookBinders took two hyperparameters as input, gamma and cost. The gamma hyperparameter is used to give curvature weight of the decision boundary or how far the influence of a single training sample reaches. The cost hyperparameter is used to control regularization which tells the algorithm how much you wish to misclassify each training sample. Testing the SVM models we have various kernels we can use Radial, Linear, Polynomial, and Sigmoid. In order to reduce overfitting we start with the simplest model which is the linear SVM.

V. Data / Cleaning - Richard

The datasets required minimal cleaning. Both the Train (BBBC_Train) and Test (BBBC_Test) datasets originally had 12 variables and the Training set consisted of 1600 observations while the Testing set had 2300. Performing basic analysis of the data we saw there were no missing values and there happened to be a column in each titled Observation. We removed this column because it would add no significant value to our models.

Checking for correlations within the pairwise plot the strongest correlation came out to be between Last_Purchase and First_Purchase which resemble the months since the customers last purchase and the months since their first purchase. With this being the only positive correlation > 0.7 we may run into the issue of multicollinearity.

The variables Choice and Gender were both treated as numeric variables within the data however we converted these to factor variables given their binary values. Further exploring these variables we discovered that approximately 70% of customers who did not a purchase were Males and ~54% of the customers who did make a purchase were also Males. Comparing the customers who did not make a purchase to those who did we find a class imbalance with only 25% of the Choice class to being customers who did make a purchase. This may cause issues in model accuracy given that SVMs do not perform well on imbalanced datasets.

VI. Findings

Using a 3 different kinds of model in determining what parameters lead to customers purchasing a certain book. This was important because the Bookbinders Book Club doesn't want to waste money sending out mailings to customers who will not end up purchasing a book. With our three models we tested Linear Regression, Logistic Regression, and Support Vector Machines.

Before we performed logistic regression we removed Last_Purchase and First_Purchase from the model due to their high collinearity. After this we performed a step wise selection which allowed us to remove the P_Youth predictor as it was insignificant. This left us with a final model of $\log \frac{1}{(1-p)})$ = -0.289 + 1.244*P_Art - 0.088*Frequency - 0.812*Gender1 - 0.294*P_Cook - 0.282*P_DIY + 0.002*Amount_Purchased - 0.196*P_Child.

Using the equation for logistic regression and taking the exponential for each coefficient value we can state the following,

The odds of a customer buying The Art History of Florence change by a factor of 3.46 with each additional art book purchased, assuming other variables remain constant.

The odds of a customer buying The Art History of Florence change by a factor of 0.915 with each additional book purchased, assuming other variables remain constant.

The odds of a male customer are .443 times that of a female customer, assuming other variables remain constant.

The odds of a customer buying The Art History of Florence change by a factor of 0.745 with each additional cook book purchased, assuming other variables remain constant.

The odds of a customer buying The Art History of Florence change by a factor of 1.002 with each additional dollar spent, assuming other variables remain constant.

The odds of a customer buying The Art History of Florence change by a factor of 1.21 with each additional children's book purchased, assuming other variables remain constant.

Logistic Regression with 0.5 cutoff

	Reference	
Prediction	0	1
0	1986	110
1	131	73

Logistic Regression with optimal cutoff (0.23)

	Reference	
Prediction	0	1
0	1490	606
1	55	149

Following the logistic regression was the test of the Support Vector Machine method using Linear, Radial, and Polynomial Kernels. A wide range of hyperparameters were used for these kernels. For the Linear and Radial kernels we used gamma values with a range defined by the sequence `seq(.01, .1, by = .01)`, with all three we tested cost with a range defined by the sequence `cost = seq(.1, 1, by = .1)`. Finally for the polynomial kernel the degree sequence was defined by `seq(2, 4, by = 1)` and the tested coef0 values came from an array of numbers `(0.01, 0.1, 0.25, 0.5, 0.75, 1, 2, 3)`. The following confusion matrices resulted.

SVM with Linear Kernel

	Reference	
Prediction	0	1
0	2088	192
1	8	12

SVM with Radial Kernel

	Reference	
Prediction	0	1
0	2047	160
1	49	44

SVM with Polynomial Kernel

	Reference	
Prediction	0	1
0	2018	140
1	78	64

Comparing the Logistic Regression Model and the SVMs we can view the accuracy's and how well the models predicted false positives or false negatives.

	Logit	SVM-Linear	SVM-Radial	SVM-Polynomial
Accuracy	0.9043	0.913	0.9091	0.9052
Sensitivity	0.9123	0.9962	0.9766	0.9628
Specificity	0.6913	0.0588	0.2157	0.3137
AUC	0.787			
Cost		0.3	0.8	1
Degree				2
Coef0				0.5

VII. Conclusion and Recommendations

Appendix

Data importing and cleaning Exploration

Any missing values?

```
anyNA(BBBC_Train)
```

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

```r
anyNA(BBBC_Test)
```

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

Check the size of the training and test datasets

```r
dim(BBBC_Train)
```

```
## [1] 1600   12
```

```r
dim(BBBC_Test)
```

```
## [1] 2300   12
```

Class imbalance

```r
table(BBBC_Train$Choice)
```

```
##
##    0    1
## 1200  400
```

View details of the data

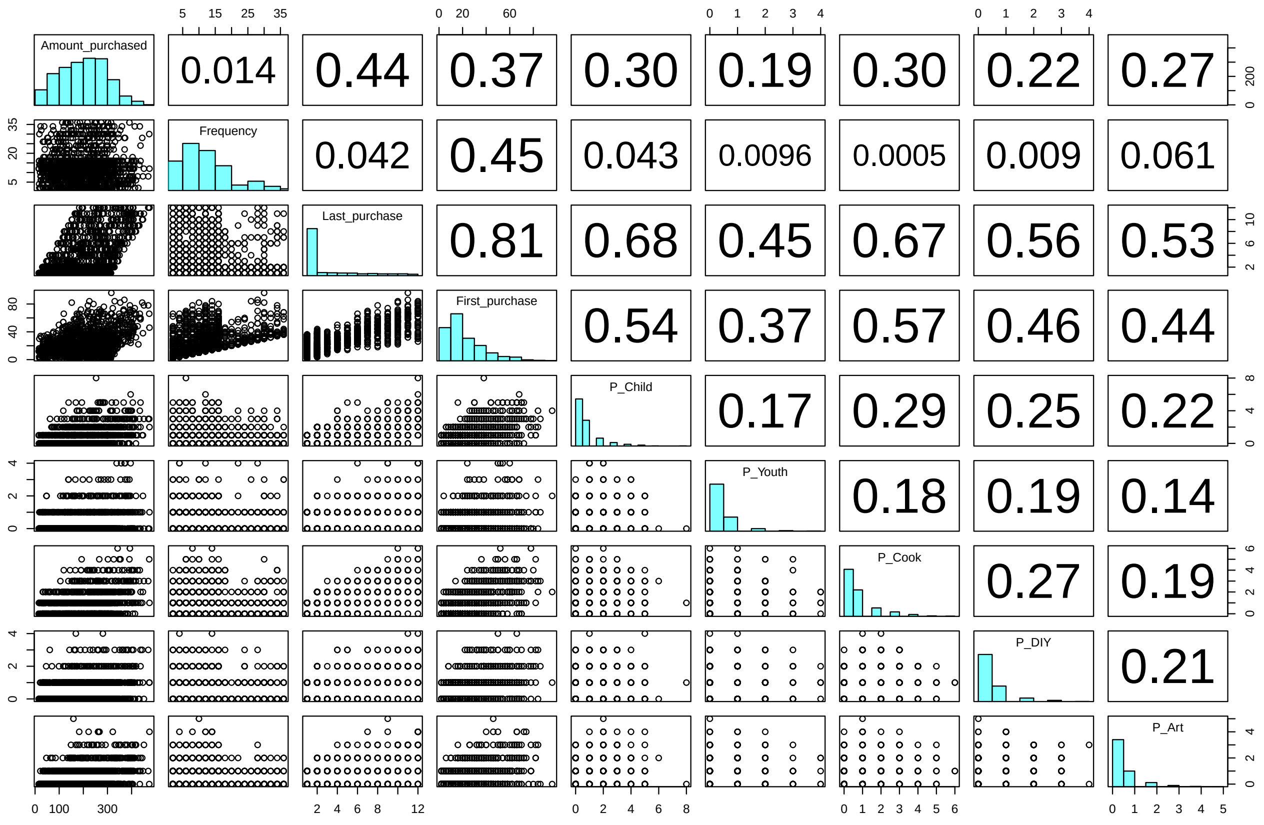
```r
str(BBBC_Train)
```

```
## tibble [1,600 x 12] (S3: tbl_df/tbl/data.frame)
##  $ Observation     : num [1:1600] 1 2 3 4 5 6 7 8 9 10 ...
##  $ Choice          : num [1:1600] 1 1 1 1 1 1 1 1 1 1 ...
##  $ Gender          : num [1:1600] 1 1 1 1 0 1 1 0 1 1 ...
##  $ Amount_purchased: num [1:1600] 113 418 336 180 320 268 198 280 393 138 ...
##  $ Frequency       : num [1:1600] 8 6 18 16 2 4 2 6 12 10 ...
##  $ Last_purchase   : num [1:1600] 1 11 6 5 3 1 12 2 11 7 ...
##  $ First_purchase  : num [1:1600] 8 66 32 42 18 4 62 12 50 38 ...
##  $ P_Child         : num [1:1600] 0 0 2 2 0 0 2 0 3 2 ...
##  $ P_Youth         : num [1:1600] 1 2 0 0 0 0 3 2 0 3 ...
##  $ P_Cook          : num [1:1600] 0 3 1 0 0 0 2 0 3 0 ...
##  $ P_DIY           : num [1:1600] 0 2 1 1 1 0 1 0 0 0 ...
##  $ P_Art           : num [1:1600] 0 3 2 1 2 0 2 0 2 1 ...
```

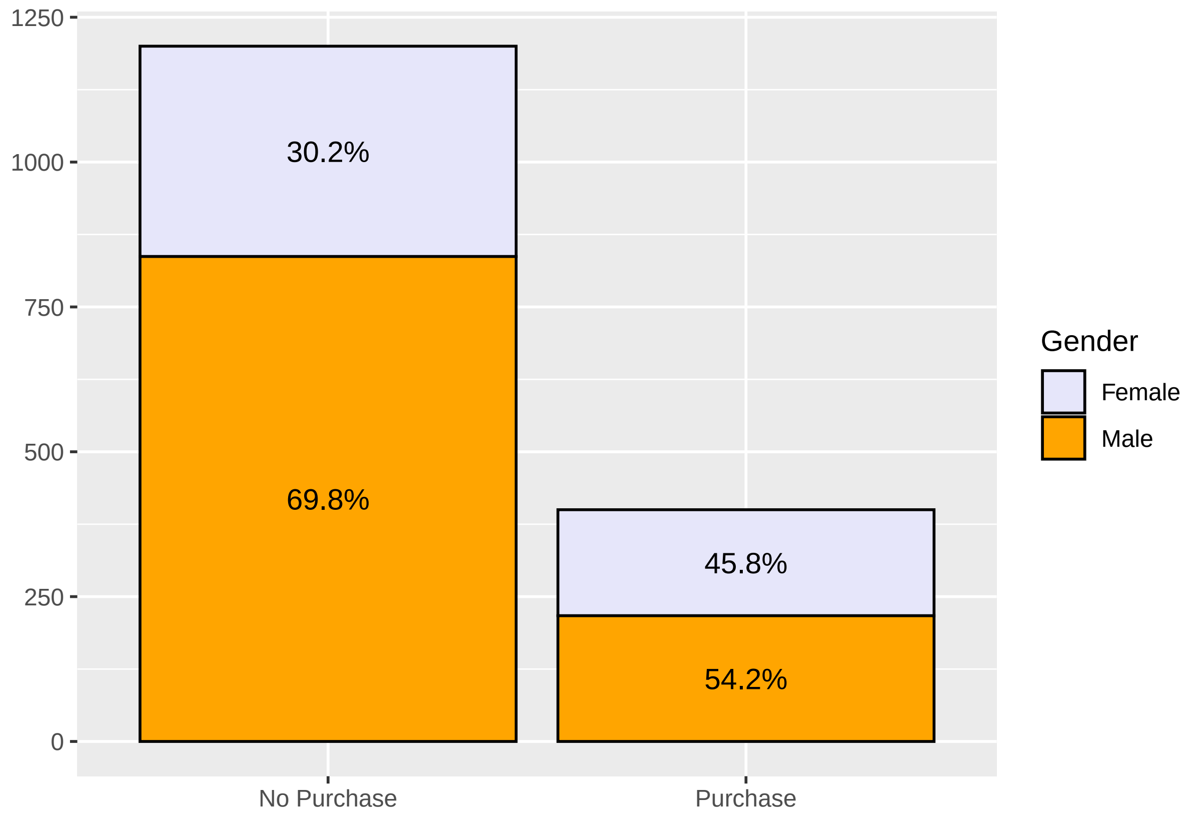
Remove `Observation` variable and convert `Choice` to factor

```r
BBBC_Train = subset(BBBC_Train, select = -Observation)
BBBC_Test = subset(BBBC_Test, select = -Observation)

BBBC_Train$Choice = as.factor(BBBC_Train$Choice)
BBBC_Test$Choice = as.factor(BBBC_Test$Choice)
```

```r
pairs.test = subset(BBBC_Train, select = -Choice)
pairs.test = subset(pairs.test, select = -Gender)

pairs(pairs.test,
      upper.panel = panel.cor,
      diag.panel = panel.hist)
```



```
ggplot(data = BBBC_Train,
       aes(x = factor(ifelse(Choice == 1, "Purchase", "No Purchase")),
           fill = factor(ifelse(Gender == 0, "Female", "Male")))) +
    geom_bar(alpha =1, color = "black", stat = "count") +
    scale_fill_manual(values = c("lavender", "orange")) +
    geom_text(aes(label = scales::percent(..count.. / tapply(..count.., ..x.., sum)[as.character(..x..)
    labs(fill = "Gender", y="", x="")
```



Linear Regression Model

```
lm.book <- lm(as.numeric(Choice) ~ ., data = BBBC_Train)
summary(lm.book)
```

```
##
## Call:
## lm(formula = as.numeric(Choice) ~ ., data = BBBC_Train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.9603 -0.2462 -0.1161  0.1622  1.0588
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        1.3642284  0.0307411  44.378  < 2e-16 ***
## Gender            -0.1309205  0.0200303  -6.536 8.48e-11 ***
## Amount_purchased   0.0002736  0.0001110   2.464   0.0138 *
## Frequency         -0.0090868  0.0021791  -4.170 3.21e-05 ***
## Last_purchase      0.0970286  0.0135589   7.156 1.26e-12 ***
## First_purchase    -0.0020024  0.0018160  -1.103   0.2704
## P_Child           -0.1262584  0.0164011  -7.698 2.41e-14 ***
## P_Youth           -0.0963563  0.0201097  -4.792 1.81e-06 ***
## P_Cook            -0.1414907  0.0166064  -8.520  < 2e-16 ***
## P_DIY             -0.1352313  0.0197873  -6.834 1.17e-11 ***
## P_Art              0.1178494  0.0194427   6.061 1.68e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3788 on 1589 degrees of freedom
## Multiple R-squared:  0.2401, Adjusted R-squared:  0.2353
## F-statistic:  50.2 on 10 and 1589 DF,  p-value: < 2.2e-16
```

Logistic Regression Model

Change gender to factor for logistic regression model

```
BBBC_Train_Logit = BBBC_Train
BBBC_Test_Logit = BBBC_Test

BBBC_Train_Logit$Gender = as.factor(BBBC_Train_Logit$Gender)

BBBC_Test_Logit$Gender = as.factor(BBBC_Test_Logit$Gender)
```

Create initial logistic regression model

```
glm.train = glm(Choice ~ ., data = BBBC_Train_Logit, family = "binomial")
summary(glm.train)
```

```
##
## Call:
## glm(formula = Choice ~ ., family = "binomial", data = BBBC_Train_Logit)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.38586  -0.66728  -0.43696  -0.02242   2.72238
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -0.3515281  0.2143839  -1.640   0.1011
## Gender1          -0.8632319  0.1374499  -6.280 3.38e-10 ***
## Amount_purchased  0.0018641  0.0007918   2.354   0.0186 *
## Frequency        -0.0755142  0.0165937  -4.551 5.35e-06 ***
## Last_purchase     0.6117713  0.0938127   6.521 6.97e-11 ***
## First_purchase   -0.0147792  0.0128027  -1.154   0.2483
## P_Child          -0.8112489  0.1167067  -6.951 3.62e-12 ***
## P_Youth          -0.6370422  0.1433778  -4.443 8.87e-06 ***
## P_Cook           -0.9230066  0.1194814  -7.725 1.12e-14 ***
## P_DIY            -0.9058697  0.1437025  -6.304 2.90e-10 ***
## P_Art             0.6861124  0.1270176   5.402 6.60e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1799.5  on 1599  degrees of freedom
## Residual deviance: 1392.2  on 1589  degrees of freedom
## AIC: 1414.2
##
## Number of Fisher Scoring iterations: 5
```

Remove variables with high multicollinearity

```r
vif(glm.train)
```

```
##            Gender Amount_purchased        Frequency    Last_purchase
##          1.023359         1.232172         2.490447        17.706670
##    First_purchase          P_Child          P_Youth           P_Cook
##          9.247748         2.992269         1.761546         3.229097
##             P_DIY            P_Art
##          1.992698         1.938089
```

```r
glm.train = glm(Choice ~ .-Last_purchase, data = BBBC_Train_Logit, family = "binomial")

vif(glm.train)
```

```
##            Gender Amount_purchased        Frequency   First_purchase
##          1.021977         1.220305         2.173240         6.886806
##           P_Child          P_Youth           P_Cook            P_DIY
##          1.904631         1.320305         2.060140         1.462770
##             P_Art
##          1.603865
```

```r
glm.train = glm(Choice ~ .-Last_purchase-First_purchase, data = BBBC_Train_Logit, family = "binomial")

vif(glm.train)
```

```
##            Gender Amount_purchased        Frequency          P_Child
##          1.020217         1.213528         1.015899         1.215500
##           P_Youth           P_Cook            P_DIY            P_Art
##          1.081019         1.228798         1.179821         1.229491
```

Build stepwise model

```r
glm.null = glm(Choice ~ 1, data = BBBC_Train_Logit, family = "binomial")
glm.full = glm(Choice ~ .-Last_purchase-First_purchase, data = BBBC_Train_Logit, family = "binomial")

glm.step1 = step(glm.null, scope = list(upper = glm.full), direction = "both", test = "Chisq", trace = F

summary(glm.step1)
```

```
##
## Call:
## glm(formula = Choice ~ P_Art + Frequency + Gender + P_Cook +
##     P_DIY + Amount_purchased + P_Child, family = "binomial",
##     data = BBBC_Train_Logit)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.30792  -0.69156  -0.47311  -0.02466   2.84228
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -0.2894506  0.2026211  -1.429  0.15314
## P_Art             1.2441330  0.0988603  12.585  < 2e-16 ***
## Frequency        -0.0885491  0.0103772  -8.533  < 2e-16 ***
## Gender1          -0.8120440  0.1345723  -6.034  1.6e-09 ***
## P_Cook           -0.2940503  0.0727520  -4.042  5.3e-05 ***
## P_DIY            -0.2823065  0.1076089  -2.623  0.00870 **
## Amount_purchased  0.0023859  0.0007678   3.108  0.00189 **
## P_Child          -0.1964495  0.0720116  -2.728  0.00637 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1799.5  on 1599  degrees of freedom
## Residual deviance: 1445.1  on 1592  degrees of freedom
## AIC: 1461.1
##
## Number of Fisher Scoring iterations: 5
```

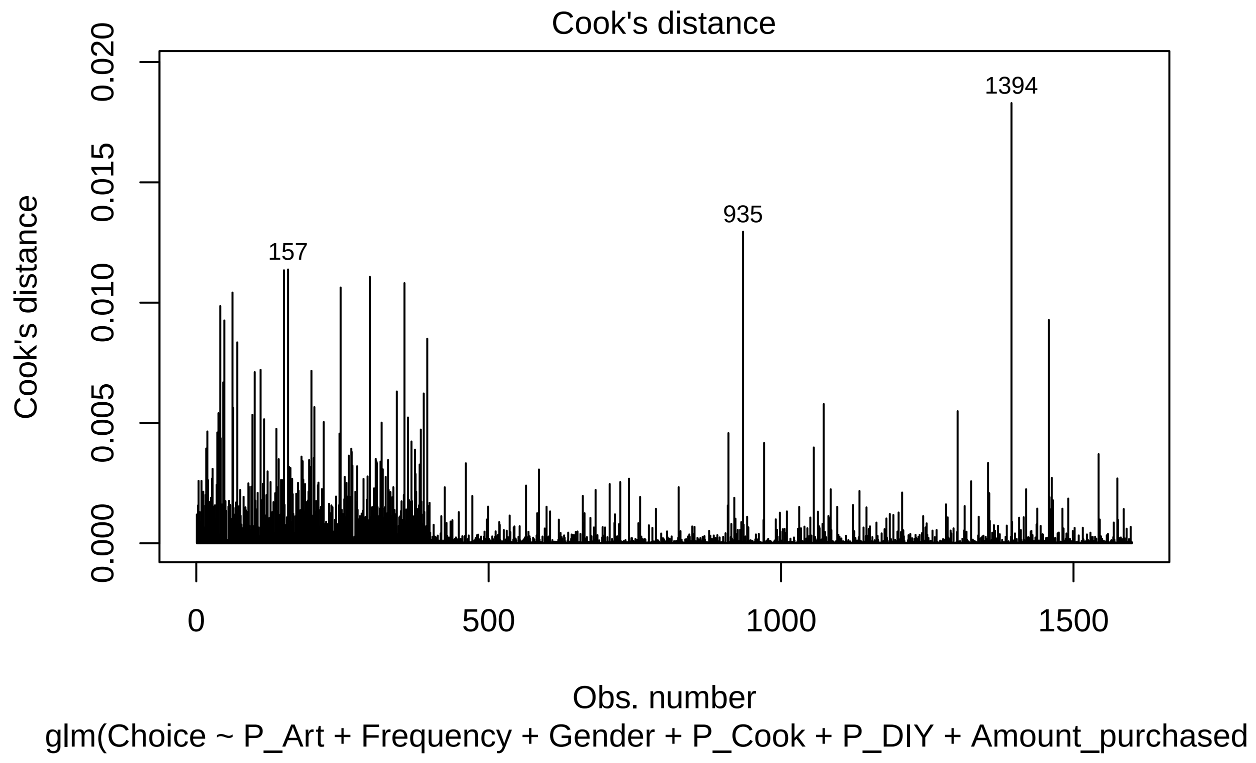
Goodness of fit

```
hoslem.test(glm.step1$y, fitted(glm.step1), g = 10)
```

```
##
##  Hosmer and Lemeshow goodness of fit (GOF) test
##
## data:  glm.step1$y, fitted(glm.step1)
## X-squared = 3.8263, df = 8, p-value = 0.8724
```

Plot

```
plot(glm.step1, which = 4)
```



```
BBBC_Train_Logit$PredProb = predict.glm(glm.step1, BBBC_Train_Logit, type = "response")
BBBC_Train_Logit$PredChoice = ifelse(BBBC_Train_Logit$PredProb >= 0.5, 1, 0)
caret::confusionMatrix(as.factor(BBBC_Train_Logit$Choice), as.factor(BBBC_Train_Logit$PredChoice))
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    0    1
##          0 1134   66
##          1  259  141
##
##                Accuracy : 0.7969
##                  95% CI : (0.7763, 0.8163)
##     No Information Rate : 0.8706
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3545
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.8141
##             Specificity : 0.6812
##          Pos Pred Value : 0.9450
##          Neg Pred Value : 0.3525
##              Prevalence : 0.8706
##          Detection Rate : 0.7087
##    Detection Prevalence : 0.7500
##       Balanced Accuracy : 0.7476
##
##        'Positive' Class : 0
##
```

The accuracy of this model is not the best, but the positive predictive value, which is what we care about, is pretty good.

Run model with test data;

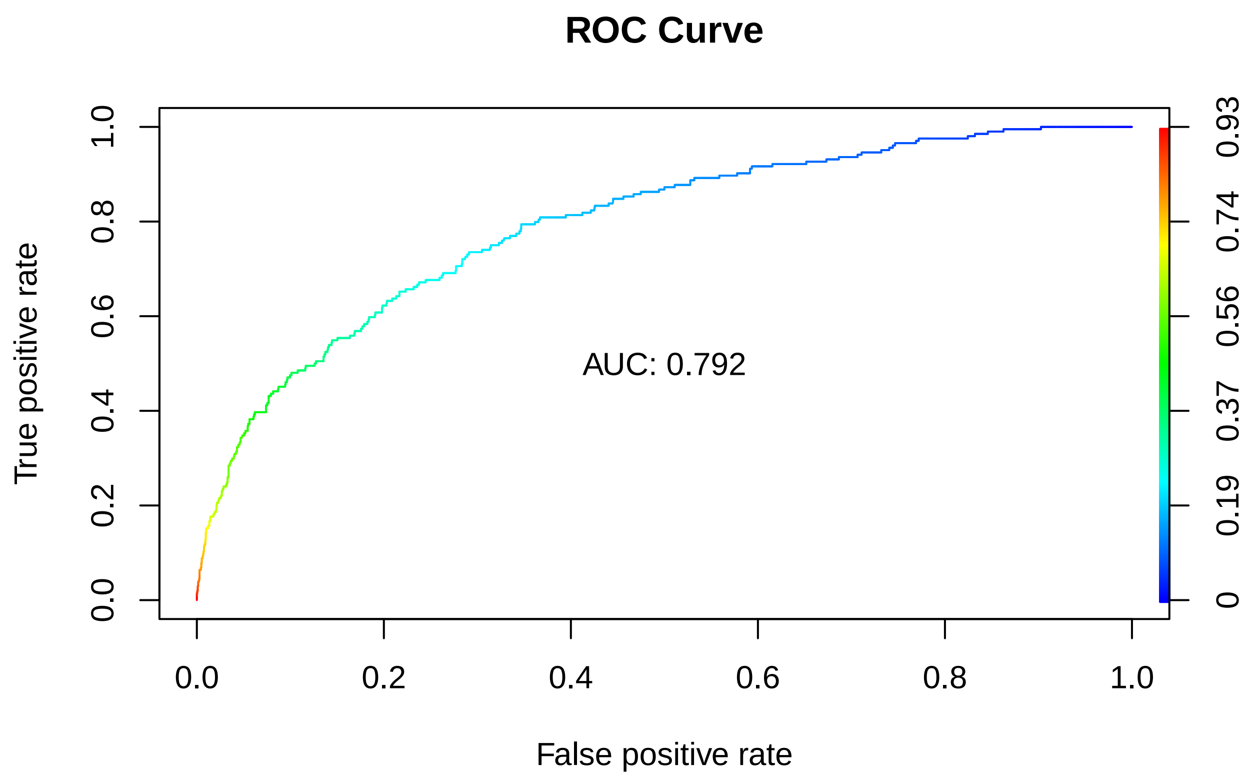
```
BBBC_Test_Logit$PredProb = predict.glm(glm.step1, BBBC_Test_Logit, type = "response")
BBBC_Test_Logit$PredChoice = ifelse(BBBC_Test_Logit$PredProb >= 0.5, 1, 0)
caret::confusionMatrix(as.factor(BBBC_Test_Logit$Choice), as.factor(BBBC_Test_Logit$PredChoice))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1986  110
##          1  131   73
##
##                Accuracy : 0.8952
##                  95% CI : (0.882, 0.9074)
##     No Information Rate : 0.9204
##     P-Value [Acc > NIR] : 1.0000
##
##                   Kappa : 0.3202
##
##  Mcnemar's Test P-Value : 0.1976
##
##             Sensitivity : 0.9381
##             Specificity : 0.3989
##          Pos Pred Value : 0.9475
##          Neg Pred Value : 0.3578
##               Prevalence : 0.9204
##           Detection Rate : 0.8635
##     Detection Prevalence : 0.9113
##        Balanced Accuracy : 0.6685
##
##         'Positive' Class : 0
##
```

The model has better accuracy with the test data, and still has a high positive predictive value. It could get better by calibrating the sensitivity and specificity.

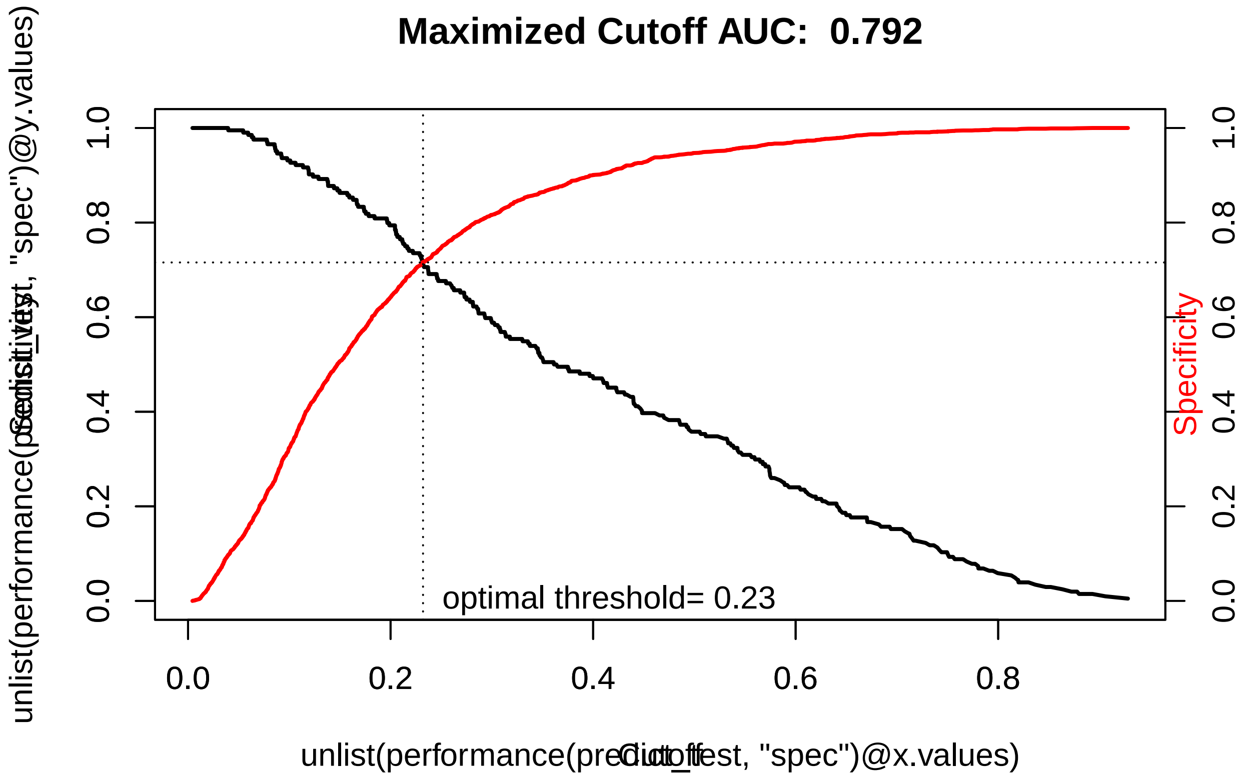
Calculate and plot AUC

```
pred_test = predict(glm.step1, BBBC_Test_Logit, type = "response")
response_test = BBBC_Test_Logit$Choice
predict_test = prediction(pred_test, response_test)
auc_test = round(as.numeric(performance(predict_test, measure = "auc")@y.values),3)
perform = performance(predict_test, "tpr","fpr")
plot(perform, colorize = T, main = "ROC Curve")
text(0.5,0.5, paste("AUC:", auc_test))
```



```
plot(unlist(performance(predict_test, "sens")@x.values),
     unlist(performance(predict_test, "sens")@y.values),
     type = "l",
     lwd = 2,
     ylab = "Sensitivity",
     xlab = "Cutoff",
     main = paste("Maximized Cutoff", "AUC: ", auc_test))

par(new = TRUE)
```

```r
plot(unlist(performance(predict_test, "spec")@x.values),
     unlist(performance(predict_test, "spec")@y.values),
     type = "l",
     lwd = 2,
     col = "red")

axis(4, at=seq(0, 1, 0.2))
mtext("Specificity", side = 4, col = 'red')

min.diff.glm = which.min(abs(unlist(performance(predict_test, "sens")@y.values) -
                              unlist(performance(predict_test, "spec")@y.values)))
min.x.glm = unlist(performance(predict_test, "sens")@x.values)[min.diff.glm]
min.y.glm = unlist(performance(predict_test, "sens")@y.values)[min.diff.glm]
optimal.glm = min.x.glm

abline(h = min.y.glm, lty = 3)
abline(v = min.x.glm, lty = 3)
text(min.x.glm,0,paste("optimal threshold=",round(optimal.glm,2)), pos = 4)
```



So now we know that the optimal cutoff threshold is 0.23, so we can refit the predictions to optimize the sensitivity and specificity.

Refit predictions and confusion matrix;

```r
BBBC_Test_Logit$PredProb = predict.glm(glm.step1, BBBC_Test_Logit, type = "response")
BBBC_Test_Logit$PredChoice = ifelse(BBBC_Test_Logit$PredProb >= 0.23, 1, 0)
caret::confusionMatrix(as.factor(BBBC_Test_Logit$Choice), as.factor(BBBC_Test_Logit$PredChoice))
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction    0    1
##          0 1490  606
##          1   55  149
##
##                Accuracy : 0.7126
##                  95% CI : (0.6936, 0.731)
##     No Information Rate : 0.6717
##     P-Value [Acc > NIR] : 1.353e-05
##
##                   Kappa : 0.1989
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9644
##             Specificity : 0.1974
##          Pos Pred Value : 0.7109
##          Neg Pred Value : 0.7304
##              Prevalence : 0.6717
##          Detection Rate : 0.6478
##    Detection Prevalence : 0.9113
##       Balanced Accuracy : 0.5809
##
##        'Positive' Class : 0
##
```

This doesn't actually look that good. The model's accuracy has dropped, as has the specificity and positive predictive value. Based on these numbers, it would most likely be better to go with the model that has a cutoff of 0.5.

The best Logit model therefore, is;

log(1/(1-p)) = -0.289 + 1.244*P_Art - 0.088*Frequency - 0.812*Gender1 - 0.294*P_Cook - 0.282*P_DIY + 0.002*Amount_Purchased - 0.196*P_Child

The most influential covariates then, are P_Art(number of art books purchased), Frequency(total number of purchases), Gender, P_Cook(number of cookbooks purchased), P_DIY (number of DIY books purchased), Amount_Purchased (total money spent), and P_Child.

Breaking down how each covariate influences the model;

The odds of a customer buying The Art History of Florence change by a factor of 3.46 with each additional art book purchased, assuming other variables remain constant.

The odds of a customer buying The Art History of Florence change by a factor of 0.915 with each additional book purchased, assuming other variables remain constant.

The odds of a male customer are .443 times that of a female customer, assuming other variables remain constant.

The odds of a customer buying The Art History of Florence change by a factor of 0.745 with each additional cook book purchased, assuming other variables remain constant.

The odds of a customer buying The Art History of Florence change by a factor of 1.002 with each additional dollar spent, assuming other variables remain constant.

The odds of a customer buying The Art History of Florence change by a factor of 1.21 with each additional children's book purchased, assuming other variables remain constant.

```
exp(1.244)
```

```
## [1] 3.469464
```

```r
exp(-0.088)
```

```
## [1] 0.9157609
```

```r
exp(-0.812)
```

```
## [1] 0.4439692
```

```r
exp(-0.294)
```

```
## [1] 0.7452765
```

```r
exp(0.002)
```

```
## [1] 1.002002
```

```r
exp(0.196)
```

```
## [1] 1.216527
```

Midwest Mailing Campaign

Data for the proposed mailing campaign; 50,000 customers cost of mailing = $0.65 / addressee cost of book = $15 Selling price of book = $31.95 overhead = 0.45*bookcost

SVM Model

```r
BBBC_Train_SVM = BBBC_Train
BBBC_Test_SVM = BBBC_Test
```

```r
str(BBBC_Train_SVM)
```

```
## tibble [1,600 x 11] (S3: tbl_df/tbl/data.frame)
##  $ Choice           : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Gender           : num [1:1600] 1 1 1 1 0 1 1 0 1 1 ...
##  $ Amount_purchased: num [1:1600] 113 418 336 180 320 268 198 280 393 138 ...
##  $ Frequency        : num [1:1600] 8 6 18 16 2 4 2 6 12 10 ...
##  $ Last_purchase    : num [1:1600] 1 11 6 5 3 1 12 2 11 7 ...
##  $ First_purchase   : num [1:1600] 8 66 32 42 18 4 62 12 50 38 ...
##  $ P_Child          : num [1:1600] 0 0 2 2 0 0 2 0 3 2 ...
##  $ P_Youth          : num [1:1600] 1 2 0 0 0 0 3 2 0 3 ...
##  $ P_Cook           : num [1:1600] 0 3 1 0 0 0 2 0 3 0 ...
##  $ P_DIY            : num [1:1600] 0 2 1 1 1 0 1 0 0 0 ...
##  $ P_Art            : num [1:1600] 0 3 2 1 2 0 2 0 2 1 ...
```

```r

# Splitting data into training and testing sets 70/30 Split
set.seed(1)
tr_ind = sample(nrow(BBBC_Train_SVM), 0.7*nrow(BBBC_Train_SVM), replace=FALSE)
book.train.split = BBBC_Train_SVM[tr_ind,]
book.test.split = BBBC_Train_SVM[-tr_ind,]
```

Use training split on BBBC_Train

```
svm_form = Choice ~ .

tuned.linear = tune.svm(svm_form, data = book.train.split,
                 kernel = "linear",
                 gamma = seq(.01, .1, by = .01),
                 cost = seq(.1, 1, by = .1))
```

```

mysvm.linear = svm(formula = svm_form,
            data = book.train.split,
            gamma =tuned.linear$best.parameters$gamma,
            cost = tuned.linear$best.parameters$cost)

summary(mysvm.linear)
```

SVM with Linear Kernel

```
##
## Call:
## svm(formula = svm_form, data = book.train.split, gamma = tuned.linear$best.parameters$gamma,
##     cost = tuned.linear$best.parameters$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.3
##
## Number of Support Vectors:  590
##
##  ( 299 291 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
# Predict on the test split
predict.test.linear = predict(mysvm.linear,
                     book.test.split,
                     type = "response")

caret::confusionMatrix(predict.test.linear, book.test.split$Choice)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 371 100
##          1   0   9
##
##                Accuracy : 0.7917
##                  95% CI : (0.7525, 0.8271)
##     No Information Rate : 0.7729
##     P-Value [Acc > NIR] : 0.1776
##
##                   Kappa : 0.1221
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.00000
##             Specificity : 0.08257
##          Pos Pred Value : 0.78769
##          Neg Pred Value : 1.00000
##              Prevalence : 0.77292
##          Detection Rate : 0.77292
##    Detection Prevalence : 0.98125
##       Balanced Accuracy : 0.54128
##
##        'Positive' Class : 0
##
```

```
predict.test.linear = predict(mysvm.linear,
                     BBBC_Test_SVM,
                     type = "response")

caret::confusionMatrix(predict.test.linear, BBBC_Test_SVM$Choice)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2088  192
##          1    8   12
##
##                Accuracy : 0.913
##                  95% CI : (0.9008, 0.9242)
##     No Information Rate : 0.9113
##     P-Value [Acc > NIR] : 0.4024
##
##                   Kappa : 0.0928
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99618
##             Specificity : 0.05882
##          Pos Pred Value : 0.91579
##          Neg Pred Value : 0.60000
##              Prevalence : 0.91130
##          Detection Rate : 0.90783
##    Detection Prevalence : 0.99130
##       Balanced Accuracy : 0.52750
##
##        'Positive' Class : 0
##
```

```

svm_form = Choice ~ .
```

```
tuned.radial = tune.svm(svm_form, data = book.train.split,
                 gamma = seq(.01, .1, by = .01),
                 cost = seq(.1, 1, by = .1))
```

```

mysvm.radial = svm(formula = svm_form,
            data = book.train.split,
            gamma =tuned.radial$best.parameters$gamma,
            cost = tuned.radial$best.parameters$cost)

summary(mysvm.radial)
```

SVM with Radial Kernel

```
##
## Call:
## svm(formula = svm_form, data = book.train.split, gamma = tuned.radial$best.parameters$gamma,
##     cost = tuned.radial$best.parameters$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.8
##
## Number of Support Vectors:  564
##
##  ( 293 271 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
# Predict on the test split
svmpredict.radial = predict(mysvm.radial,
                     book.test.split,
                     type = "response")

caret::confusionMatrix(svmpredict.radial, book.test.split$Choice)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 363  85
##          1   8  24
##
##                Accuracy : 0.8062
##                  95% CI : (0.768, 0.8407)
##     No Information Rate : 0.7729
##     P-Value [Acc > NIR] : 0.04375
##
##                   Kappa : 0.2646
```

```
##
##  Mcnemar's Test P-Value : 3.252e-15
##
##             Sensitivity : 0.9784
##             Specificity : 0.2202
##          Pos Pred Value : 0.8103
##          Neg Pred Value : 0.7500
##              Prevalence : 0.7729
##          Detection Rate : 0.7562
##    Detection Prevalence : 0.9333
##       Balanced Accuracy : 0.5993
##
##        'Positive' Class : 0
##
```

```
predict.test.radial = predict(mysvm.radial,
                     BBBC_Test_SVM,
                     type = "response")

caret::confusionMatrix(predict.test.radial, BBBC_Test_SVM$Choice)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2047  160
##          1   49   44
##
##                Accuracy : 0.9091
##                  95% CI : (0.8966, 0.9206)
##     No Information Rate : 0.9113
##     P-Value [Acc > NIR] : 0.6597
##
##                   Kappa : 0.2549
##
##  Mcnemar's Test P-Value : 2.765e-14
##
##             Sensitivity : 0.9766
##             Specificity : 0.2157
##          Pos Pred Value : 0.9275
##          Neg Pred Value : 0.4731
##              Prevalence : 0.9113
##          Detection Rate : 0.8900
##    Detection Prevalence : 0.9596
##       Balanced Accuracy : 0.5962
##
##        'Positive' Class : 0
##
```

```
svm_form = Choice ~ .

tuned.poly = tune.svm(svm_form, data = book.train.split,
                      kernel = "polynomial",
                      degree = seq(2, 4, by = 1),
                      coef0 = c(0.01, 0.1, 0.25, 0.5, 0.75, 1, 2, 3),
                      cost = seq(.1, 1, by = .1))
```

```

mysvm.poly = tuned.poly$best.model

summary(mysvm.poly)
```

SVM with Polynomial Kernel

```
##
## Call:
## best.svm(x = svm_form, data = book.train.split, degree = seq(2, 4,
##     by = 1), coef0 = c(0.01, 0.1, 0.25, 0.5, 0.75, 1, 2, 3), cost = seq(0.1,
##     1, by = 0.1), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.5
##      degree:  2
##      coef.0:  3
##
## Number of Support Vectors:  518
##
##  ( 263 255 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
# Predict on the test split
svmpredict.poly = predict(mysvm.poly,
                     book.test.split,
                     type = "response")

caret::confusionMatrix(svmpredict.poly, book.test.split$Choice)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 354  74
##          1  17  35
##
##                Accuracy : 0.8104
##                  95% CI : (0.7724, 0.8445)
##     No Information Rate : 0.7729
##     P-Value [Acc > NIR] : 0.02648
##
##                   Kappa : 0.3376
```

```
##
##  Mcnemar's Test P-Value : 4.348e-09
##
##             Sensitivity : 0.9542
##             Specificity : 0.3211
##          Pos Pred Value : 0.8271
##          Neg Pred Value : 0.6731
##              Prevalence : 0.7729
##          Detection Rate : 0.7375
##    Detection Prevalence : 0.8917
##       Balanced Accuracy : 0.6376
##
##        'Positive' Class : 0
##
```

```
predict.test.poly = predict(mysvm.poly,
                     BBBC_Test_SVM,
                     type = "response")

caret::confusionMatrix(predict.test.poly, BBBC_Test_SVM$Choice)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2018  137
##          1   78   67
##
##                Accuracy : 0.9065
##                  95% CI : (0.8939, 0.9181)
##     No Information Rate : 0.9113
##     P-Value [Acc > NIR] : 0.8013
##
##                   Kappa : 0.3349
##
##  Mcnemar's Test P-Value : 7.635e-05
##
##             Sensitivity : 0.9628
##             Specificity : 0.3284
##          Pos Pred Value : 0.9364
##          Neg Pred Value : 0.4621
##              Prevalence : 0.9113
##          Detection Rate : 0.8774
##    Detection Prevalence : 0.9370
##       Balanced Accuracy : 0.6456
##
##        'Positive' Class : 0
##
```