Analysis of Sales Information of Orange Juice Using SVMs

Gap Kim

INTRODUCTION

In this study, the OJ dataset from the ISLR package has been analyzed. The dataset contains 1070 sales information for the Citrus Hill (CH) and Minute Maid (MM) brands of orange juice. After running a basic exploratroy data analysis, support vector based classification has been performed to predict which brand of the orange juice the customer purchased. Due to unbalanced categorical classes, stratified sampling using four-fold cross validation has been performed. Three classifiers, support vector classifier (SVC), support vector machine (SVM) with radial kernal, and SVM with second order polynomial, have been used for the analysis, and their performance has been compared using the test error rate, ROC curve, and area under the ROC curve.

EXPLORATORY DATA ANALYSIS

The OJ dataset contains 1070 sales information on following 18 variables:

```
  Purchase: A factor with levels CH and MM indicating customer purchased Citrus Hill or Minute M
aid Orange Juice
  WeekofPurchase: Week of purchase
  StoreID: Store ID
  PriceCH: Price charged for CH
  PriceMM: Price charged for MM
  DiscCH: Discount offered for CH
  DiscMM: Discount offered for MM
  SpecialCH: Indicator of special on CH
  SpecialMM: Indicator of special on MM
  LoyalCH: Customer brand loyalty for CH
  SalePriceMM: Sale price for MM
  SalePriceCH: Sale price for CH
  PriceDiff: Sale price of MM less sale price of CH
  Store7: A factor with levels No and Yes indicating whether the sale is at Store 7
  PctDiscMM: Percentage discount for MM
  PctDiscCH: Percentage discount for CH
  ListPriceDiff: List price of MM less list price of CH
  STORE: Which of 5 possible stores the sale occured at
```

Upon inspection, following variables were treated as categorical variables: Purchase, SpecialCH, StoreID, SpecialMM, Store7, and STORE. The total counts of factors in each of the variables are summarized in Table 1. It is noted that some variables are severely skewed in class proportions.

Table 1: Total count of factors in categorical variables

Table 1: Total count of factors in categorical variables

Purchase		SpecialCH		SpecialMM		Store7	
CH	MM	0	1	0	1	No	Yes
Purchase		**SpecialCH**		**SpecialMM**		**Store7**	
CH	**MM**	**0**	**1**	**0**	**1**	**No**	**Yes**
653	417	912	158	897	173	714	356

Table 1(continued): Total count of factors in categorical variables

StoreID					STORE				
1	**2**	**3**	**4**	**7**	**0**	**1**	**2**	**3**	**4**
157	222	196	139	356	356	157	222	196	139

In Figure 1, boxplots are provided for the remaining quantitative variables in the dataset. There are quite a few variables with outliers. In particular, the distributions of DiscCH and PctDiscMM are severely right-skewed with many outliers.

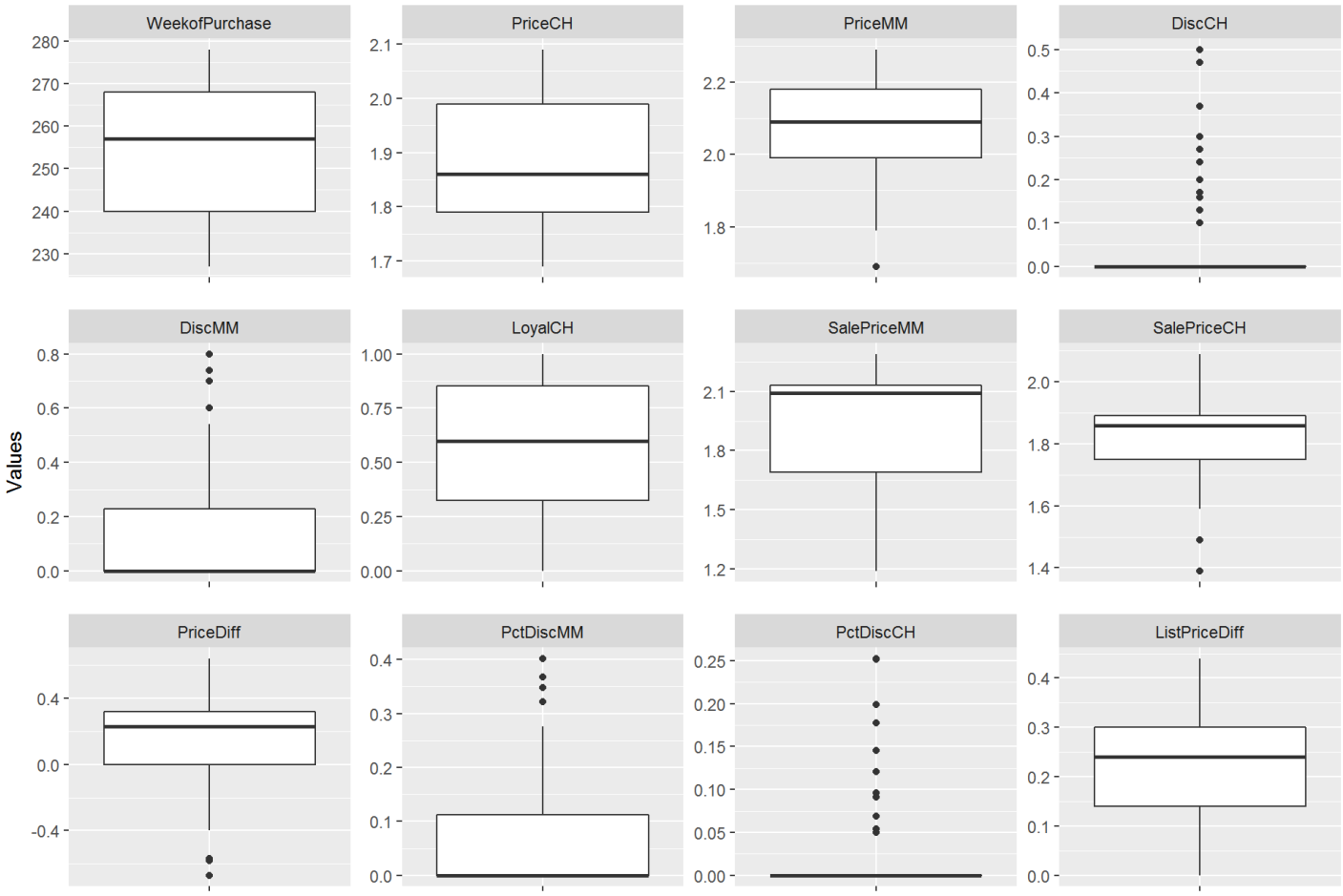

Figure 1: Boxplots of quantitative variables

In Figure 2, correlations between the variables are summarized. There are variables with strong corrleation, which seems natural considering what they measure and their relationship with other variables. For example, when there is a Discount offered for CH (DiscCH), the Sale price for CH (salePriceCH) will be lower. For the same reason, Percentage discount for CH (PctDiscCH) will be correlated. The correlation table shows a strong negative correlation between DiscCH~SalePriceCH and a strong positive correlation between DiscCH~PctDiscCH.

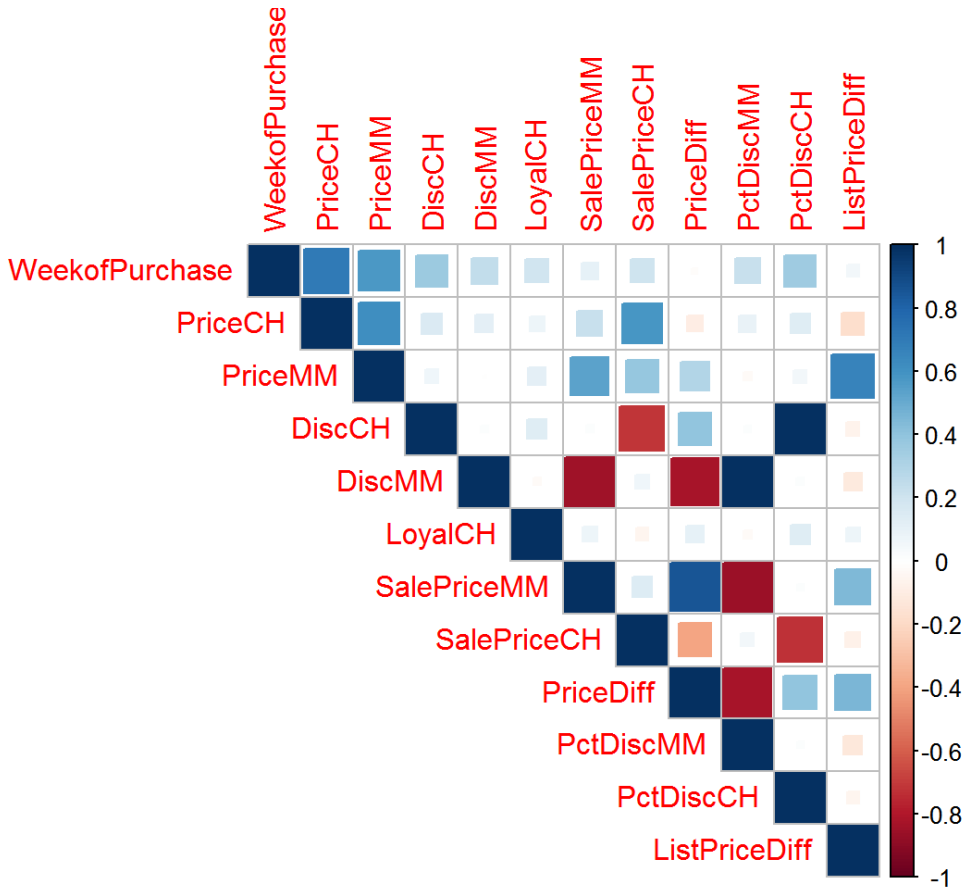

Figure 2: Correlation matrix plot of all quantitative variables

ANALYSIS USING SUPPORT VECTOR MACHINES

First, the dataset has been split into training and test set using a four-fold cross validation with stratified sampling. The dataset was analyzed with Support Vector Classifier (SVC), Support Vector Machine (SVM) with radial and second degree polynomial kernels. In each of the classifier, parameter optimzation has been performed and the performance characteristics of the classifiers have been analyzed.

Stratified Sampling for 4-fold Cross Validation

Unbalanced class in the test dataset can lead to inaccurate test performance measures. Thus, four-fold cross validation with stratified sampling has been applied to all the classifiers used in this study. Within each fold, sample stratification has been performed based on all categorical variables (Purchase, SpecialCH, SpecialMM, StoreID, Store7, and STORE). Hence each fold contains roughly 800 training datapoints and 270 test datapoints where each training and test fold contains similar proportions of the classes compared with the full dataset. The class proportions from the full dataset and each fold are summarized in Table 2.

Table 2: Class ratios in the full dataset and each of the fold in the test dataset

	Purchase		**SpecialCH**		**SpecialMM**		**Store7**	
Dataset	**CH**	**MM**	**0**	**1**	**0**	**1**	**No**	**Yes**
Fullset	0.61	0.39	0.852	0.148	0.838	0.162	0.667	0.333
Fold1	0.611	0.389	0.853	0.147	0.842	0.158	0.668	0.332
Fold2	0.61	0.39	0.854	0.146	0.839	0.161	0.67	0.33

Table 2: Class ratios in the full dataset and each of the fold in the test dataset

	Purchase		SpecialCH		SpecialMM		Store7	
Dataset	**CH**	**MM**	**0**	**1**	**0**	**1**	**No**	**Yes**
Fold3	0.608	0.392	0.847	0.153	0.84	0.16	0.664	0.336
Fold4	0.611	0.389	0.856	0.144	0.833	0.167	0.667	0.333

Table 2(continued): Class ratios in the full dataset and 4-fold test dataset

	StoreID					STORE				
Dataset	**1**	**2**	**3**	**4**	**7**	**0**	**1**	**2**	**3**	**4**
Fullset	0.147	0.207	0.183	0.13	0.333	0.333	0.147	0.207	0.183	0.13
Fold1	0.151	0.208	0.181	0.128	0.332	0.332	0.151	0.208	0.181	0.128
Fold2	0.142	0.21	0.187	0.131	0.33	0.33	0.142	0.21	0.187	0.131
Fold3	0.142	0.209	0.183	0.131	0.336	0.336	0.142	0.209	0.183	0.131
Fold4	0.152	0.204	0.181	0.13	0.333	0.333	0.152	0.204	0.181	0.13

Support Vector Classifier (SVC)

Support vector classifier is basically SVM with “linear” kernel. First, training and test error rates have been calculated before tuning of any parameter is performed. Then, non-negative tuning parameter C was optimized through the ‘cost’ parameter included in ‘svm’ function in R package ‘e1071’. The peformance characteristic of the SVC was analyzed using the ROC curve.

Training and Test Error before Parameter Tuning

The SVC has been run with cost=0.01, and the resulting summary ouput of the SVC based on the fourth fold is the following:

```
Call:
svm(formula = Purchase ~ ., data = OJ[-folds.id[[i]], ], kernel = "linear",
    cost = 0.01, scale = TRUE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01
      gamma:  0.04167

Number of Support Vectors:  433

 ( 217 216 )


Number of Classes:  2

Levels:
 CH MM
```

The fourth fold contains 800 training datapoints and 270 test datapoints. Among 800 training datapoints, 433 are support vectors with 217 of them from CH margin, and 216 from MM margin. More than half of the training datapoints are used in determining the decision hyperplane and the margin in this fold.

The confusion matrix from the training data has been calculated by aggregating results from all four folds. The results are summarized in Table 3 where the training error rate is 0.1626. Note that the total counts in the confusion matrix is 3210 (= 1070 x 3), which is equivalent to three times the total observations.

Table 3: Confusion matrix based on training data aggregated from all folds

	Actual	
Predicted	**CH**	**MM**
CH	1736	299
MM	223	952

The confusion matrix aggregated from all folds based on the test data is provided in Table 4 where the test error rate is 0.1673. Note that the total counts in the confusion matrix is 1070, which equals the total number of datapoints. This means that each datapoint has been evaluated exactly once in the test performance.

Table 4: Confusion matrix based on test data aggregated from all folds

	Actual	
Predicted	**CH**	**MM**
CH	575	101
MM	78	316

Optimal Parameters and Test Error

The ‘cost’ parameter has been optimized with 10-fold cross validation within the training set. Since each fold produces a best tuning parameter, the optimal value for the tuning parameter was decided by averaging the four best values.

```
Best 'cost' parameter for each fold: 0.04 0.05 0.04 0.05
```

```
Selected optimal value for 'cost' parameter = 0.045
```

The optimal ‘cost’ value of 0.045 has been used to fit the training set from the 4 folds. For each fold, the fitted SVC has been evaluated using the test dataset for that respective fold. The resulting confusion matrix is summarized in Table 5 where the test error rate is 0.1692.

Table 5: Confusion matrix based on test data aggregated from all folds using optimized SVC

	Actual	
Predicted	**CH**	**MM**
CH	573	101
MM	80	316

Performance Characteristics of SVC

The cutoff value that decides the class of a datapoint can be varied to understand the performance behavior of the SVC. The ROC curve (sensitivity vs. Type I error rate) as well as the test error rate have been plotted in Figure 3. The verical line at Cutoff=0 indicates the default cutoff when the class is decided by whether the fitted values are postiive or negative. The ROC curve of the classifier is favorable since it hugs the upper-left corner of the plot indicating high sensitivity with low Type I error rates.

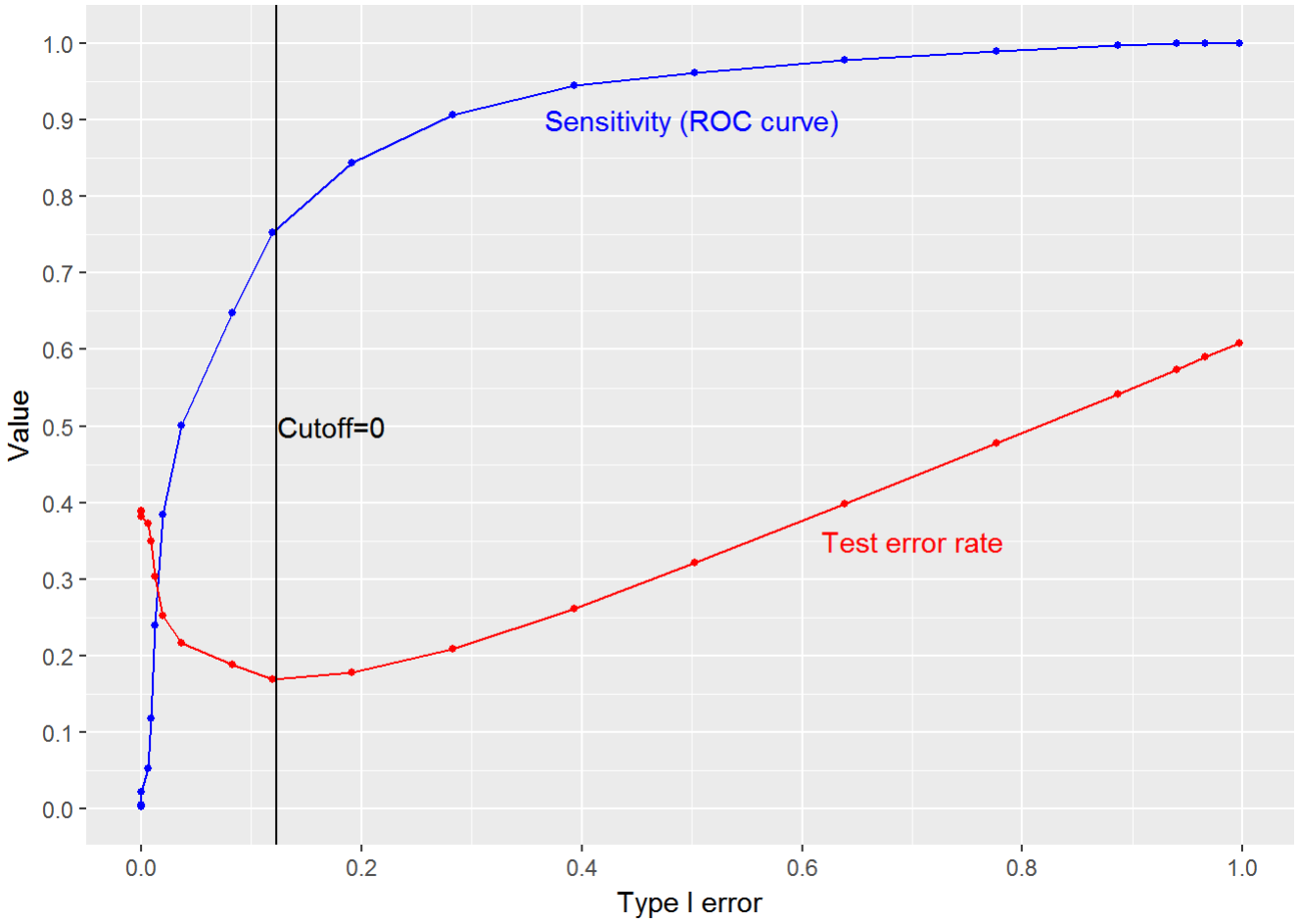

Figure 3: Performance characteristics of SVC on test dataset

Support Vector Machine with Radial Kernel

In this section, SVM with radial kernal has been analyzed in predicting the customer purchase behavior. Similarly, the training and test error rates have been calculated before any parameter tuning. The 'cost' parameter has been tuned using the default setting for gamma parameter. The peformance characteristics of the SVM was analyzed using the ROC curve.

Training and Test Error before Parameter Tuning

The SVM has been run with cost=0.01, and the summary ouput of the classifier based on the fourth fold is the following:

```
Call:
svm(formula = Purchase ~ ., data = OJ[-folds.id[[i]], ], kernel = "radial",
    cost = 0.01, scale = TRUE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.01
      gamma:  0.04167

Number of Support Vectors:  627

 ( 315 312 )


Number of Classes:  2

Levels:
 CH MM
```

With cost=0.01, there are 627 support vectors out of 800 training datapoints. The confusion matrix from the training data is summarized in Table 6 where the training error rate is 0.3897. However, upon inspection of the confusion matrix, no prediction was made for MM. This may be due to the model having too much bias. One can imagine a big multidimensional envelop encompassing all the data and the prediction resulting in a single class.

Table 6: Confusion matrix based on training data aggregated from all folds

	Actual	
Predicted	**CH**	**MM**
CH	1959	1251
MM	0	0

As shown in Table 7, the confusion matrix from the aggregated test dataset also shows no prediction for the MM label.

Table 7: Confusion matrix based on test data aggregated from all folds

	Actual	
Predicted	**CH**	**MM**
CH	653	417
MM	0	0

Optimal Parameters and Test Error

The parameter tuning was performed for the ‘cost’ while using the default values for the gamma parameter:

```
Best 'cost' parameter for each fold: 0.5 0.08 0.1 0.3
```

```
Selected optimal value for 'cost' parameter= 0.245
```

The optimal ‘cost’ value of 0.245 has been used to fit the training sets from the 4 folds. For each fold, the fitted SVM has been evaluated using the test dataset for that respective fold. The resulting aggregated confusion matrix is provided in Table 8 where the test error rate is 0.1682.

Table 8: Confusion matrix based on test data aggregated from all folds using optimized SVM with radial kernel

	Actual	
Predicted	**CH**	**MM**
CH	576	103
MM	77	314

Performance Characteristics of SVM with Radial Kernel

The ROC curve (sensitivity vs. Type I error rate) plotted based on the test dataset is shown in Figure 4, which is similar to that of SVC. The verical line at Cutoff=0 indicates the default cutoff when the class is decided by whether the fitted values are postiive or negative.

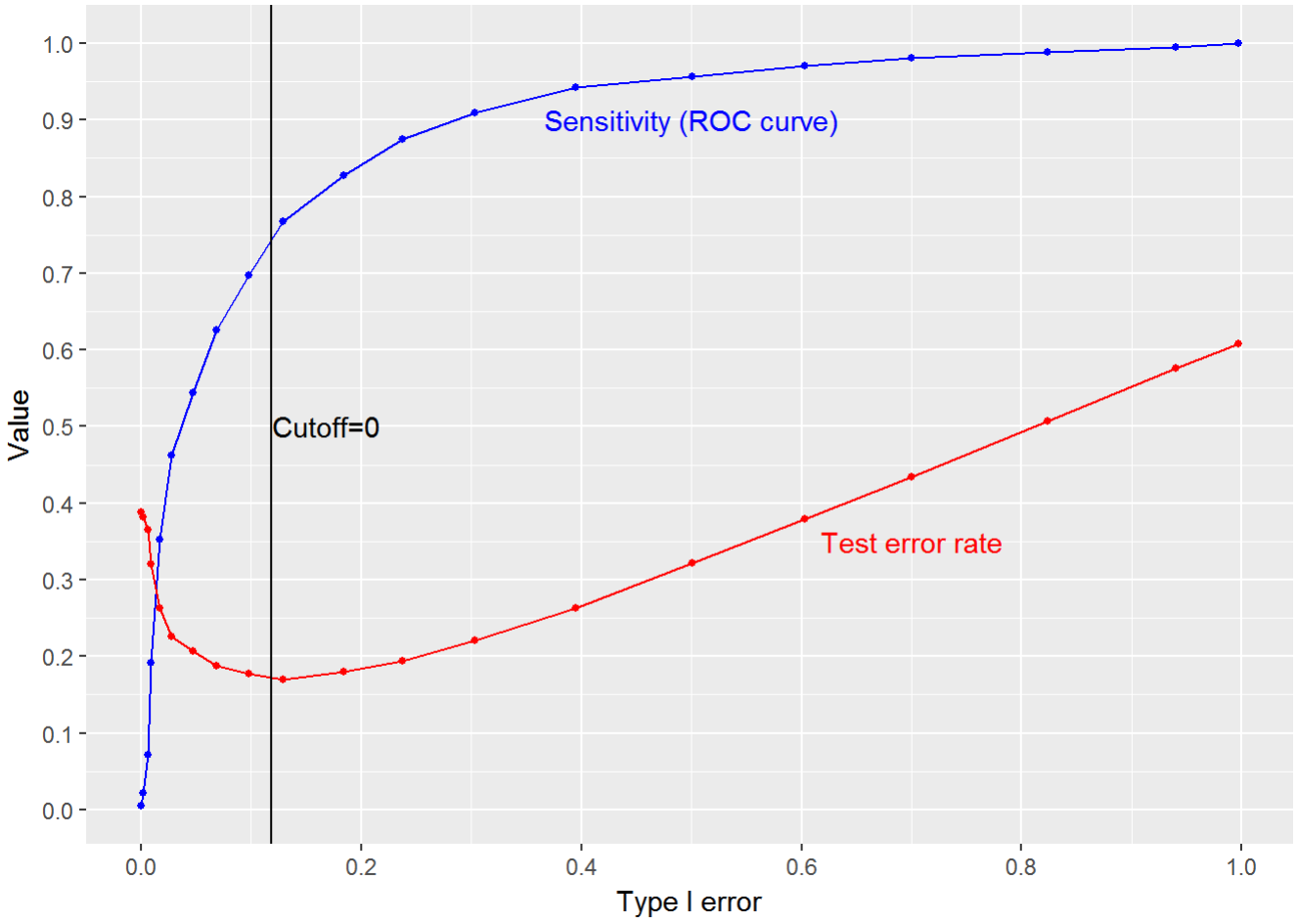

Figure 4: Performance characteristics of SVM with radial kernal on the test dataset

Support Vector Machine with Polynomial Kernel

In this section, SVM with second order polynomial kernal has been analyzed following a similar procedure as the two previous classifiers. The ‘cost’ parameter has been tuned using the default setting for gamma parameter, and the peformance characteristics of the SVM was analyzed using the ROC curve.

Training and Test Error before Parameter Tuning

The SVM has been run with cost=0.01, and the summary ouput of the classifier from the fourth fold is the following:

```
Call:
svm(formula = Purchase ~ ., data = OJ[-folds.id[[i]], ], kernel = "polynomial",
    degree = 2, cost = 0.01, scale = TRUE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  0.01
     degree:  2
      gamma:  0.04167
     coef.0:  0

Number of Support Vectors:  627

 ( 315 312 )


Number of Classes:  2

Levels:
 CH MM
```

The SVM with second degree polynomial kernel resulted in the same number of support vectors as the radial kernel, and produced the same confusion matrix as summarized in Table 9.

Table 9: Confusion matrix based on training data aggregated from all folds

	Actual	
Predicted	**CH**	**MM**
CH	1959	1251
MM	0	0

The resulting confusion matrix from the aggregated test dataset is also the same as the SVM with the radial kernel. The cost=0.01 resulted in highly biased model with too many support vectors involved in the calculation of margins for both SVMs.

Table 10: Confusion matrix based on test data aggregated from all folds

	Actual	
Predicted	**CH**	**MM**

Table 10: Confusion matrix based on test data aggregated from all folds

	Actual	
Predicted	**CH**	**MM**
CH	653	417
MM	0	0

Optimal Parameters and Test Error

The parameter tuning was performed for the ‘cost’ while using the default values for gamma paramter. One can observe that the best ‘cost’ parameter is different for each fold due to variations in the sampling fold. The average value has been used when evaluating the test dataset.

```
Best 'cost' parameter for each fold: 7 8 5 4
```

```
Selected optimal value for 'cost' parameter= 6
```

The resulting confusion matrix is provided in Table 11 where the test error rate is 0.1888.

Table 11: Confusion matrix based on test data aggregated from all folds using optimized SVM with second order polynomial kernel

	Actual	
Predicted	**CH**	**MM**
CH	595	144
MM	58	273

Performance Characteristics of SVM with Polynomial Kernel

The ROC curve (sensitivity vs. Type I error rate) plotted based on the test dataset is shown in Figure 5, which shows a similar trend compared with SVC and SVM with radial kernel.

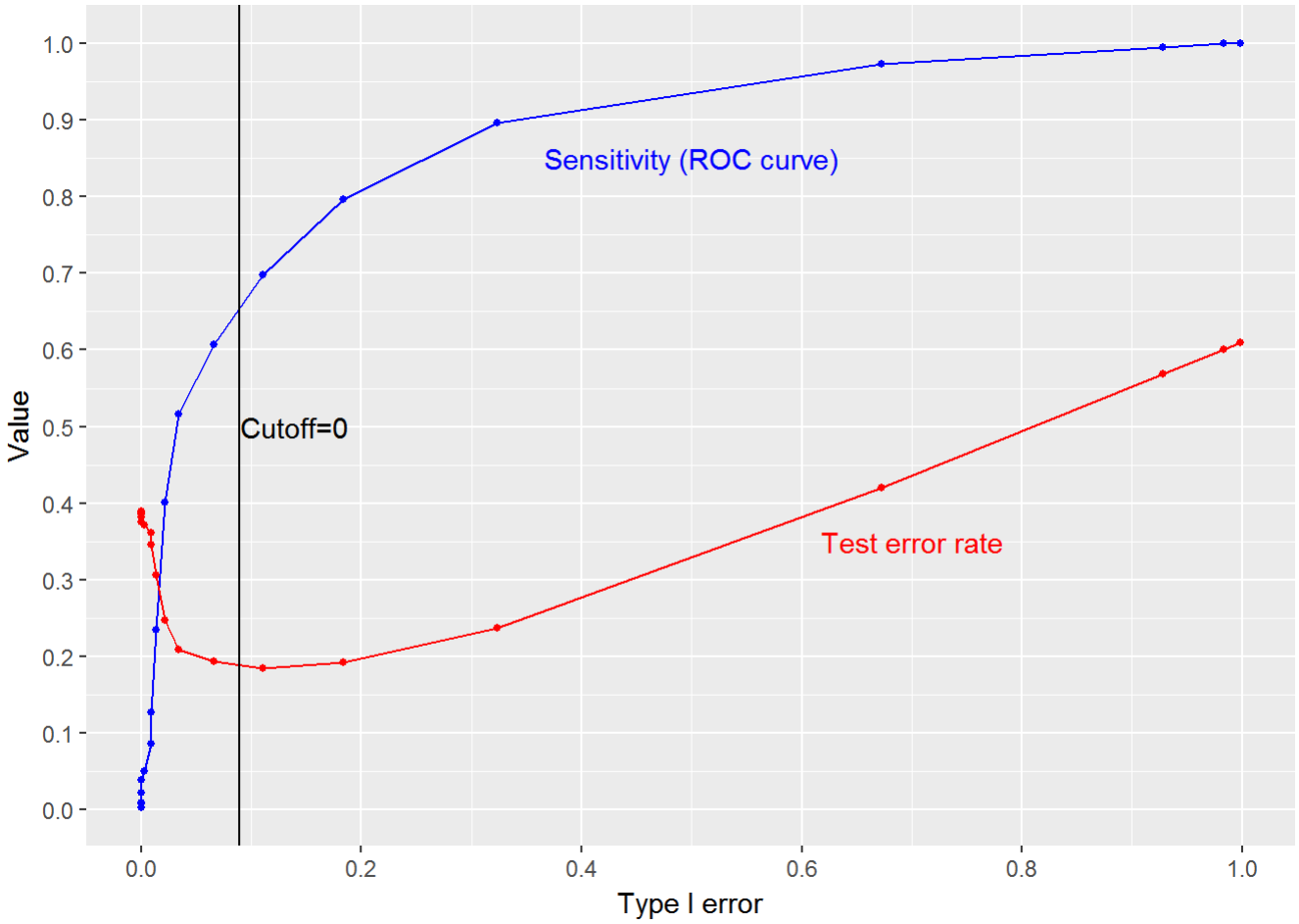

Figure 5: Performance characteristics of SVM with second order polynomial kernal on the test dataset

DISCUSSION

The performance metric from the three classifiers are summarized in Table 11. The SVC and SVM with radial kernel had very similar performance and had lower test error rate than the SVM with second degree polynomial. The specificity of the SVM with second degree polynomial was higher than the other two classifiers.

The area under the curve (AUC) of ROC metric has been computed for all three classifiers. The AUC of SVC and SVM with radial kernel were almost equivalent, and the AUC of the SVM with second degree polynomial came out the lowest.

Table 11: Performance comparsion of the SVC and SVM with radial and polynoimial kernel

	Accuracy	Error rate	Type I error	Type II error	Sensitivity	Specificity	AUC of ROC
SVC(linear)	0.8308	0.1692	0.1225	0.2422	0.7578	0.8775	0.8943
SVM(radial)	0.8318	0.1682	0.1179	0.247	0.753	0.8821	0.8922
SVM(poly2)	0.8112	0.1888	0.0888	0.3453	0.6547	0.9112	0.8779

With the AUC of SVC being the highest, visual analysis of the classifier has been performed. From the decision tree analysis in the previous report, predictors LoyalCH and PriceDiff were the two most influential variables in predicting the customer purchase behavior. Hence the customer purchase of CH and MM were plotted in these two dimensions in Figure 6 for all 1070 datapoints. The red color indicates customers who actually purchased CH and the blue color indicates those purchased MM. It can be observed that customers who mostly purchased CH are at upper-right corner and those who purchased MM are at the lower-left corner.

On the same plot, the support vectors identified from all four folds have been indicated with “X” mark. It is readily seen that support vectors form a disgonal band separating the customers who purchased CH and MM. One can visualize the amount of support vectors that were involved in deciding the decision hyperplane and the margin.

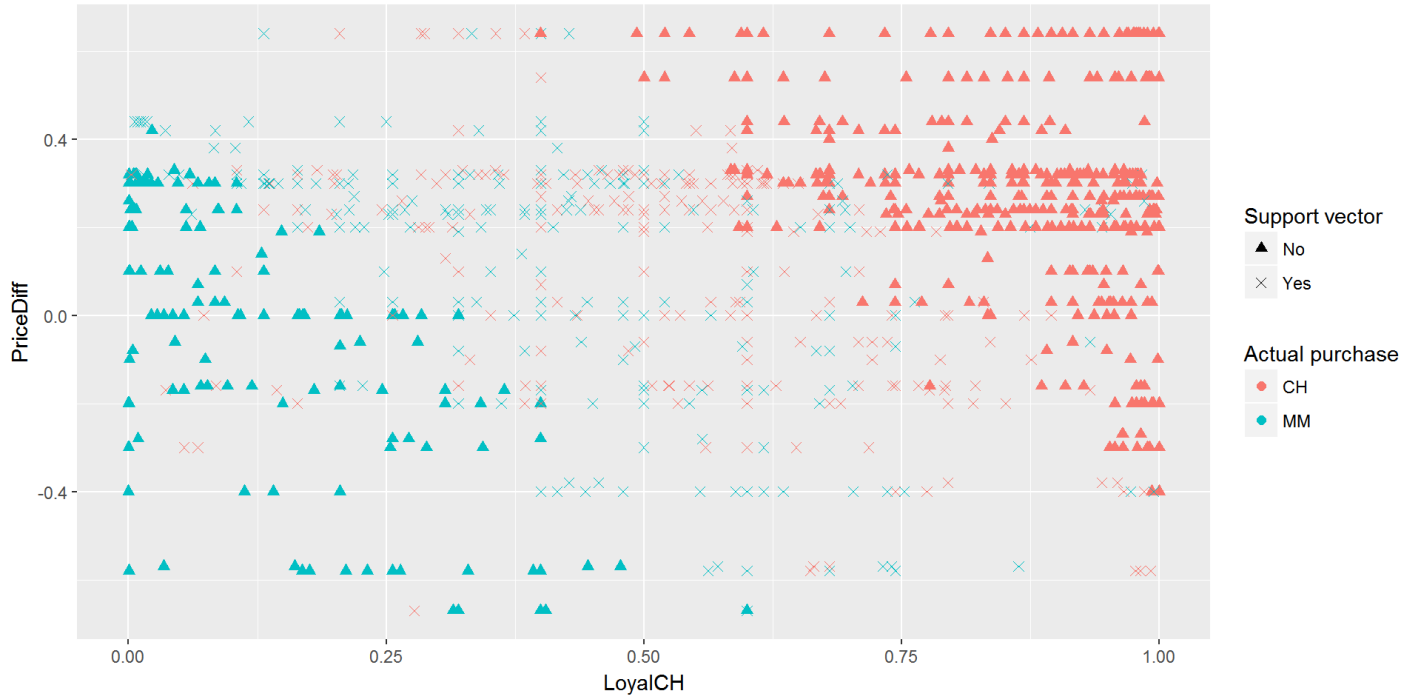

Figure 6: Visualization of SVC classifier on all 1070 observations

Based on the performance metric and the visualization of the classifier, SVC or SVM with radial kernel seems preferable. Based on Figure 6, however, it is hard to visualize whether radial kernel has advantage over linear kernel. Therefore, the simpler classifier, SVC, would be the better choice due to computational advantages.

CONCLUSIONS

In this study, the OJ dataset from the ISLR package has been analyzed, which contains 1070 sales information for the Citrus Hill (CH) and Minute Maid (MM) brands of orange juice. Three support vector based classifiers, SVC, SVM with radial kernel, and SVM with second degree polynomial, have been evaluated in predicting whether a customer purchased CH or MM based on 17 variables. Due to unbalance of classes in the categorical variables, stratified sampling with four-fold cross validation has been performed. Following conclusions may be drawn from this study:

All three classifiers performed well in predicting the customer purchase behavior with accuracies above 80% where SVC and SVM with radial kernel performed slightly better than SVM with second degree polynomial. The performance metric of the three classifiers are summarized in the table below:

Performance comparsion of the SVC and SVM with radial and polynoimial kernel

	Accuracy	Error rate	Type I error	Type II error	Sensitivity	Specificity	AUC of ROC
SVC(linear)	0.8308	0.1692	0.1225	0.2422	0.7578	0.8775	0.8943
SVM(radial)	0.8318	0.1682	0.1179	0.247	0.753	0.8821	0.8922
SVM(poly2)	0.8112	0.1888	0.0888	0.3453	0.6547	0.9112	0.8779

The performance of SVC and SVM with radial kernel are comparable to each other in all metrics and outperformed SVM with second degree polynomial kernal. Since SVC is simpler than SVM with radial kernel, SVC is preferable in terms of computational time and efficiency.

APPENDIX

All R codes used in producing the results are included below:

```
##############################
### Initial Setup

knitr::opts_chunk$set(comment=NA, echo=FALSE, warning=TRUE, message=FALSE,
                      fig.align="center")
options(digits=4)
rm(list=ls())

library(ISLR)
data(OJ)

##################################
### EXPLORATORY DATA ANALYSIS
##################################

OJ$StoreID = as.factor(OJ$StoreID)
OJ$SpecialCH = as.factor(OJ$SpecialCH)
OJ$SpecialMM = as.factor(OJ$SpecialMM)
OJ$STORE = as.factor(OJ$STORE)

### Table 1: Total count of factors in categorical variables

library(htmlTable)

tab1 = c(table(OJ$Purchase), table(OJ$SpecialCH), table(OJ$SpecialMM), table(OJ$Store7))
tab2 = c(table(OJ$StoreID), table(OJ$STORE))

htmlTable(tab1,
          caption="Table 1: Total count of factors in categorical variables",
          header = names(tab1),
          cgroup = c("Purchase","SpecialCH","SpecialMM", "Store7"),
          n.cgroup = c(2,2,2,2),
          css.cell = "width: 60px;")

htmlTable(tab2,
          caption="Table 1(continued): Total count of factors in categorical variables",
          header = names(tab2),
          cgroup = c("StoreID","STORE"),
          n.cgroup = c(5,5),
          css.cell = "width: 60px;")

### Figure 1: Boxplots of quantitative variables

library(reshape2)
library(ggplot2)
melt.OJ = melt(OJ)

ggplot(data=melt.OJ) +
   geom_boxplot(aes(x="", y=value)) +
   facet_wrap(~variable, scale="free") +
   labs(x="", y="Values")
```

```
### Figure 2: Correlation matrix plot of all quantitative variables

library(corrplot)
OJ.cor = cor(OJ[,c(2,4,5,6,7,10,11,12,13,15,16,17)])
corrplot(OJ.cor, method="square", type="upper")

########################################
### ANALYSIS USING SVM
########################################

#################################
### User function definition

### Calculates confusion matrix
print.cm = function(cm, cap) {
   table.cm = unname(cm)
   colnames(table.cm) = colnames(cm)
   rownames(table.cm) = colnames(cm)

   out.cm = htmlTable(table.cm,
             caption=paste(cap),
             cgroup = c("Actual"),
             rowlabel = "Predicted",
             css.cell = "width: 100px;")
   return(out.cm)
}

### Calucate performance metric from confusion matrix
my.metric = function(cm){
   accuracy = round((cm[1,1]+cm[2,2])/sum(cm),4)
   error.rate = round((cm[1,2]+cm[2,1])/sum(cm),4)
   type.1 = round(cm[2,1]/sum(cm[,1]),4)
   type.2 = round(cm[1,2]/sum(cm[,2]),4)
   sensitivity = round(cm[2,2]/sum(cm[,2]),4)
   specificity = round(cm[1,1]/sum(cm[,1]),4)

   metric = c(accuracy, error.rate, type.1, type.2, sensitivity, specificity)
   return(metric)
}

### Calculate area under curve
get.AUC = function(x, y) {
   dx = diff(x)
   y.mean = diff(y)/2 + y[c(1:length(y)-1)]

   auc = sum(dx * y.mean)
   return(auc)
}

#####################################
### Training and test datasets

###
### Sample stratification
```

```
###

library(caret)

feature.str = names(OJ)
feature.groups = paste(OJ$Purchase,
                    OJ$SpecialCH,
                    OJ$SpecialMM,
                    OJ$StoreID,
                    OJ$Store7,
                    OJ$STORE)

set.seed(21)
folds.id = createFolds(feature.groups, k=4)

Purchase = OJ$Purchase

feature.ratio = function(feature, data, folds.id){

   n = length(folds.id)
   f = length(unique(data[, feature]))

   ratio = matrix(rep(NA, n*f), ncol=f)

   for(i in c(1:n)){

      ratio[i,] = table(data[folds.id[[i]], feature]) /
            sum(table(data[folds.id[[i]], feature]))
   }

   ratio = rbind(ratio)
   return(ratio)
}

get.ratio = function(feature, OJ, folds.id) {
   full.rt = table(OJ[,feature]) / sum(table(OJ[,feature]))
   rt = feature.ratio(feature, OJ, folds.id)
   ratio = rbind(full.rt, rt)

   rownames(ratio) = c("Fullset", "Fold1", "Fold2", "Fold3", "Fold4")
   return(ratio)
}

Purchase.ratio = get.ratio("Purchase", OJ, folds.id)
SpecialCH.ratio = get.ratio("SpecialCH", OJ, folds.id)
SpecialMM.ratio = get.ratio("SpecialMM", OJ, folds.id)
StoreID.ratio = get.ratio("StoreID", OJ, folds.id)
Store7.ratio = get.ratio("Store7", OJ, folds.id)
STORE.ratio = get.ratio("STORE", OJ, folds.id)

### Table 2: Class ratios in the full dataset and each of the fold in the test dataset

tab.ratio1 = round(cbind(Purchase.ratio,SpecialCH.ratio,SpecialMM.ratio,Store7.ratio),3)
tab.ratio2 = round(cbind(StoreID.ratio,STORE.ratio),3)
```

```
htmlTable(tab.ratio1,
          caption="Table 2: Class ratios in the full dataset and each of the fold in the test da
taset",
          cgroup = c("Purchase","SpecialCH","SpecialMM","Store7"),
          n.cgroup = c(2,2,2,2),
          rowlabel = "Dataset",
          css.cell = "width: 70px;")

htmlTable(tab.ratio2,
          caption="Table 2(continued): Class ratios in the full dataset and 4-fold test dataset"
,
          cgroup = c("StoreID","STORE"),
          n.cgroup = c(5,5),
          rowlabel = "Dataset",
          css.cell = "width: 70px;")

#####################################
### SVC

library(e1071)

n.fold = 4

ypred.train = list()
ypred.test = list()
cm.train = matrix(rep(0,n.fold), ncol=2)
cm.test = matrix(rep(0,n.fold), ncol=2)

for(i in c(1:n.fold)) {
   svmfit = svm(Purchase ~ ., data=OJ[-folds.id[[i]],],
                kernel="linear", cost=0.01, scale=TRUE)

   ypred.train[[i]] = predict(svmfit)
   ypred.test[[i]] = predict(svmfit, newdata=OJ[folds.id[[i]],])

   cm.train = cm.train + table(Predict=ypred.train[[i]], Actual=Purchase[-folds.id[[i]]])
   cm.test = cm.test + table(Predict=ypred.test[[i]], Actual=Purchase[folds.id[[i]]])
}

train.error = (cm.train[1,2]+cm.train[2,1]) / sum(cm.train)
test.error = (cm.test[1,2]+cm.test[2,1]) / sum(cm.test)

summary(svmfit)

### Table 3: Confusion matrix based on training data <br> aggregated from all folds

print.cm(cm.train, "Table 3: Confusion matrix based on training data <br> aggregated from all fo
lds")

### Table 4: Confusion matrix based on test data <br> aggregated from all folds

print.cm(cm.test, "Table 4: Confusion matrix based on test data <br> aggregated from all folds")
```

```
bestmod = rep(NA, n.fold)

for(i in c(1:n.fold)) {
   set.seed(10)
   tune.out = tune(svm, Purchase ~ ., data=OJ[-folds.id[[i]],], kernel="linear",
                   ranges=list(cost=c(0.01, 0.03, 0.04, 0.05, 0.06, 1)))
   bestmod[i] = tune.out$best.model$cost
}

bestmod.opt = mean(bestmod)

cat("Best 'cost' parameter for each fold:", bestmod)
cat("Selected optimal value for 'cost' parameter =", bestmod.opt)


ypred.test = list()
cm.test = matrix(rep(0,n.fold), ncol=2)
fitted = rep(NA, nrow(OJ))   # decision values
svmfit.SVC = list()

for(i in c(1:n.fold)) {
   svmfit = svm(Purchase ~ ., data=OJ[-folds.id[[i]],],
                kernel="linear", cost=bestmod.opt, scale=TRUE)
   svmfit.SVC[[i]] = svmfit
   ypred.test[[i]] = predict(svmfit, newdata=OJ[folds.id[[i]],], decision.values=T)
   cm.test = cm.test + table(Predict=ypred.test[[i]], Actual=Purchase[folds.id[[i]]])

   fitted[folds.id[[i]]] = attributes(ypred.test[[i]])$decision.values
}

SVC.cm.test = cm.test
SVC.test.error = (cm.test[1,2]+cm.test[2,1]) / sum(cm.test)
SVC_test.typeI.ref = cm.test[2,1] / sum(cm.test[,1])
#SVC.test.error

### Table 5: Confusion matrix based on test data <br> aggregated from all folds using optimized
 SVC

print.cm(SVC.cm.test, "Table 5: Confusion matrix based on test data <br> aggregated from all fol
ds using optimized SVC")

cutoff = seq(min(fitted)+0.01, max(fitted)-0.01, length.out=20)

sensitivity = rep(NA, 20)
typeI = rep(NA, 20)
test.error = rep(NA, 20)

for(j in c(1:20)) {
   pred.label = rep("MM", nrow(OJ))
   pred.label[which(fitted > cutoff[j])] = "CH"

   cm = table(pred.label, OJ$Purchase)

   sensitivity[j] = cm[2,2] / sum(cm[,2])
```

```
   typeI[j] = cm[2,1] / sum(cm[,1])
   test.error[j] = (cm[1,2]+cm[2,1]) / sum(cm)
}

auc.SVC = get.AUC(typeI, sensitivity)
#auc.SVC

### Figure 3: Performance characteristics of SVC on test dataset

library(ggplot2)

ggplot() +
   geom_line(aes(x=typeI, y=sensitivity), color="blue") +
   geom_point(aes(x=typeI, y=sensitivity), color="blue", size=1) +
   geom_line(aes(x=typeI, y=test.error), color="red") +
   geom_point(aes(x=typeI, y=test.error), color="red", size=1) +
   scale_y_continuous(breaks=seq(0,1,length.out=11)) +
   scale_x_continuous(breaks=seq(0,1,length.out=6)) +
   geom_vline(xintercept=SVC_test.typeI.ref) +
   annotate("text", x=SVC_test.typeI.ref+0.05, y=0.5, label="Cutoff=0") +
   annotate("text", x=0.5, y=0.9, color="blue", label="Sensitivity (ROC curve)") +
   annotate("text", x=0.7, y=0.35, color="red", label="Test error rate") +
   labs(x="Type I error", y="Value")

#######################################
### SVM radial kernel

ypred.train = list()
ypred.test = list()
cm.train = matrix(rep(0,n.fold), ncol=2)
cm.test = matrix(rep(0,n.fold), ncol=2)

for(i in c(1:n.fold)) {
   svmfit = svm(Purchase ~ ., data=OJ[-folds.id[[i]],],
                kernel="radial", cost=0.01, scale=TRUE)

   ypred.train[[i]] = predict(svmfit)
   ypred.test[[i]] = predict(svmfit, newdata=OJ[folds.id[[i]],])

   cm.train = cm.train + table(Predict=ypred.train[[i]], Actual=Purchase[-folds.id[[i]]])
   cm.test = cm.test + table(Predict=ypred.test[[i]], Actual=Purchase[folds.id[[i]]])
}

train.error = (cm.train[1,2]+cm.train[2,1]) / sum(cm.train)
test.error = (cm.test[1,2]+cm.test[2,1]) / sum(cm.test)

summary(svmfit)

### Table 6: Confusion matrix based on training data <br> aggregated from all folds

print.cm(cm.train, "Table 6: Confusion matrix based on training data <br> aggregated from all fo
lds")

### Table 7: Confusion matrix based on test data <br> aggregated from all folds
```

```
print.cm(cm.test, "Table 7: Confusion matrix based on test data <br> aggregated from all folds")

bestmod = rep(NA, n.fold)

for(i in c(1:n.fold)) {
   set.seed(10)
   tune.out = tune(svm, Purchase ~ ., data=OJ[-folds.id[[i]],], kernel="radial",
                   ranges=list(cost=c(0.06, 0.08, 0.1, 0.3, 0.5, 1)))
   bestmod[i] = tune.out$best.model$cost
}

bestmod.opt = mean(bestmod)

cat("Best 'cost' parameter for each fold:", bestmod)
cat("Selected optimal value for 'cost' parameter=", bestmod.opt)

ypred.test = list()
cm.test = matrix(rep(0,n.fold), ncol=2)
fitted = rep(NA, nrow(OJ))   # decision values

for(i in c(1:n.fold)) {
   svmfit = svm(Purchase ~ ., data=OJ[-folds.id[[i]],],
                kernel="radial", cost=bestmod.opt, scale=TRUE)
   ypred.test[[i]] = predict(svmfit, newdata=OJ[folds.id[[i]],], decision.values=T)
   cm.test = cm.test + table(Predict=ypred.test[[i]], Actual=Purchase[folds.id[[i]]])

   fitted[folds.id[[i]]] = attributes(ypred.test[[i]])$decision.values
}

SVM_rad.cm.test = cm.test
SVM_rad.test.error = (cm.test[1,2]+cm.test[2,1]) / sum(cm.test)
SVM_rad.typeI.ref = cm.test[2,1] / sum(cm.test[,1])
#SVM_rad.test.error

### Table 8: Confusion matrix based on test data <br> aggregated from all folds using optimized
 SVM with radial kernel

print.cm(SVM_rad.cm.test, "Table 8: Confusion matrix based on test data <br> aggregated from all
 folds using optimized SVM <br> with radial kernel")

cutoff = seq(min(fitted)+0.01, max(fitted)-0.01, length.out=20)

sensitivity = rep(NA, 20)
typeI = rep(NA, 20)
test.error = rep(NA, 20)

for(j in c(1:20)) {
   pred.label = rep("MM", nrow(OJ))
   pred.label[which(fitted > cutoff[j])] = "CH"

   cm = table(pred.label, OJ$Purchase)

   sensitivity[j] = cm[2,2] / sum(cm[,2])
```

```
   typeI[j] = cm[2,1] / sum(cm[,1])
   test.error[j] = (cm[1,2]+cm[2,1]) / sum(cm)
}

auc.SVM_rad = get.AUC(typeI, sensitivity)
#auc.SVM_rad

### Figure 4: Performance characteristics of SVM with radial kernal on the test dataset

library(ggplot2)

ggplot() +
   geom_line(aes(x=typeI, y=sensitivity), color="blue") +
   geom_point(aes(x=typeI, y=sensitivity), color="blue", size=1) +
   geom_line(aes(x=typeI, y=test.error), color="red") +
   geom_point(aes(x=typeI, y=test.error), color="red", size=1) +
   scale_y_continuous(breaks=seq(0,1,length.out=11)) +
   scale_x_continuous(breaks=seq(0,1,length.out=6)) +
   geom_vline(xintercept=SVM_rad.typeI.ref) +
   annotate("text", x=SVM_rad.typeI.ref+0.05, y=0.5, label="Cutoff=0") +
   annotate("text", x=0.5, y=0.9, color="blue", label="Sensitivity (ROC curve)") +
   annotate("text", x=0.7, y=0.35, color="red", label="Test error rate") +
   labs(x="Type I error", y="Value")

#######################################
### SVM 2nd polynomial

ypred.train = list()
ypred.test = list()
cm.train = matrix(rep(0,n.fold), ncol=2)
cm.test = matrix(rep(0,n.fold), ncol=2)

for(i in c(1:n.fold)) {
   svmfit = svm(Purchase ~ ., data=OJ[-folds.id[[i]],],
                kernel="polynomial", degree=2, cost=0.01, scale=TRUE)

   ypred.train[[i]] = predict(svmfit)
   ypred.test[[i]] = predict(svmfit, newdata=OJ[folds.id[[i]],])

   cm.train = cm.train + table(Predict=ypred.train[[i]], Actual=Purchase[-folds.id[[i]]])
   cm.test = cm.test + table(Predict=ypred.test[[i]], Actual=Purchase[folds.id[[i]]])
}

train.error = (cm.train[1,2]+cm.train[2,1]) / sum(cm.train)
test.error = (cm.test[1,2]+cm.test[2,1]) / sum(cm.test)

summary(svmfit)

### Table 9: Confusion matrix based on training data <br> aggregated from all folds

print.cm(cm.train, "Table 9: Confusion matrix based on training data <br> aggregated from all fo
lds")

### Table 10: Confusion matrix based on test data <br> aggregated from all folds
```

```
print.cm(cm.test, "Table 10: Confusion matrix based on test data <br> aggregated from all folds"
)

bestmod = rep(NA, n.fold)

for(i in c(1:n.fold)) {
   set.seed(10)
   tune.out = tune(svm, Purchase ~ ., data=OJ[-folds.id[[i]],],
                   kernel="polynomial", degree=2,
                   ranges=list(cost=c(1, 3, 4, 5, 7, 8, 9, 10)))
   bestmod[i] = tune.out$best.model$cost
}

bestmod.opt = mean(bestmod)

cat("Best 'cost' parameter for each fold:", bestmod)
cat("Selected optimal value for 'cost' parameter=", bestmod.opt)


ypred.test = list()
cm.test = matrix(rep(0,n.fold), ncol=2)
fitted = rep(NA, nrow(OJ))   # decision values

for(i in c(1:n.fold)) {
   svmfit = svm(Purchase ~ ., data=OJ[-folds.id[[i]],],
                kernel="polynomial", degree=2, cost=bestmod.opt, scale=TRUE)
   ypred.test[[i]] = predict(svmfit, newdata=OJ[folds.id[[i]],], decision.values=T)
   cm.test = cm.test + table(Predict=ypred.test[[i]], Actual=Purchase[folds.id[[i]]])

   fitted[folds.id[[i]]] = attributes(ypred.test[[i]])$decision.values
}

SVM_poly.cm.test = cm.test
SVM_poly.test.error = (cm.test[1,2]+cm.test[2,1]) / sum(cm.test)
SVM_poly.typeI.ref = cm.test[2,1] / sum(cm.test[,1])
#SVM_poly.typeI.ref
#SVM_poly.test.error

### Table 11: Confusion matrix based on test data <br> aggregated from all folds using optimized
 SVM with second order polynomial kernel

print.cm(SVM_poly.cm.test, "Table 11: Confusion matrix based on test data <br> aggregated from a
ll folds using optimized SVM <br> with second order polynomial kernel")

cutoff = seq(min(fitted)+0.01, max(fitted)-0.01, length.out=20)

sensitivity = rep(NA, 20)
typeI = rep(NA, 20)
test.error = rep(NA, 20)

for(j in c(1:20)) {
   pred.label = rep("MM", nrow(OJ))
   pred.label[which(fitted > cutoff[j])] = "CH"
```

```
   cm = table(pred.label, OJ$Purchase)

   sensitivity[j] = cm[2,2] / sum(cm[,2])
   typeI[j] = cm[2,1] / sum(cm[,1])
   test.error[j] = (cm[1,2]+cm[2,1]) / sum(cm)
}

auc.SVM_poly = get.AUC(typeI, sensitivity)
#auc.SVM_poly

### Figure 5: Performance characteristics of SVM with second order polynomial kernal on the test
 dataset

library(ggplot2)

ggplot() +
   geom_line(aes(x=typeI, y=sensitivity), color="blue") +
   geom_point(aes(x=typeI, y=sensitivity), color="blue", size=1) +
   geom_line(aes(x=typeI, y=test.error), color="red") +
   geom_point(aes(x=typeI, y=test.error), color="red", size=1) +
   scale_y_continuous(breaks=seq(0,1,length.out=11)) +
   scale_x_continuous(breaks=seq(0,1,length.out=6)) +
   geom_vline(xintercept=SVM_poly.typeI.ref) +
   annotate("text", x=SVM_poly.typeI.ref+0.05, y=0.5, label="Cutoff=0") +
   annotate("text", x=0.5, y=0.85, color="blue", label="Sensitivity (ROC curve)") +
   annotate("text", x=0.7, y=0.35, color="red", label="Test error rate") +
   labs(x="Type I error", y="Value")

############################
### DISCUSSION
############################

### Table 11: Performance comparsion of the SVC and SVM with radial and polynoimial kerne

auc = round(c(auc.SVC, auc.SVM_rad, auc.SVM_poly),4)

metric = rbind(my.metric(SVC.cm.test), my.metric(SVM_rad.cm.test),
               my.metric(SVM_poly.cm.test))
colnames(metric) = c("  Accuracy  ", "Error rate", "Type I error", "Type II error", "Sensitivit
y", "Specificity")
rownames(metric) = c("SVC(linear)", "SVM(radial)", "SVM(poly2)")

metric = cbind(metric, auc)
colnames(metric)[7] = "AUC of ROC"

htmlTable(metric,
          caption = "Table 11: Performance comparsion of the SVC and SVM with radial and polynoi
mial kernel",
          css.cell="width: 100px;")

ind = c(1:1070)

f1 = ind[-folds.id[[1]]]
```

```
SV1.id = f1[svmfit.SVC[[1]]$index]

f2 = ind[-folds.id[[2]]]
SV2.id = f2[svmfit.SVC[[2]]$index]

f3 = ind[-folds.id[[3]]]
SV3.id = f3[svmfit.SVC[[3]]$index]

f4 = ind[-folds.id[[4]]]
SV4.id = f4[svmfit.SVC[[4]]$index]

SV = union(
   union(SV1.id, SV2.id),
   union(SV3.id, SV4.id)
)

SV.id = rep(0, 1070)
SV.id[SV] = 1

### Figure 6: Visualization of SVC classifier on all 1070 observations

ggplot() +
   geom_point(aes(x=OJ$LoyalCH, y=OJ$PriceDiff, color=OJ$Purchase,
                  shape=as.factor(SV.id)), size=2) +
   scale_shape_manual(values=c(17,4), labels=c("No", "Yes")) +
   labs(x="LoyalCH", y="PriceDiff", color="Actual purchase", shape="Support vector")

htmlTable(metric,
          caption = "Performance comparsion of the SVC and SVM with radial and polynoimial kerne
l",
          css.cell="width: 100px;")
```