Analysis of Sales Information of Orange Juice Using Tree-based Methods

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 and Minute Maid brands of orange juice. After running a basic exploratroy data analysis, tree-based classification has been performed to predict which brand of the orange juice the customer purchased. First, a decision tree with unpruned and pruned results are compared. Next, three ensemble methods, Bagging, Random Forest and Gradient Boosting, have been compared.

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 variable are summarized in Table 1.

Table 1: Total count of factors in categorical variables

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

Boxplots are provided in Figure 1 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.

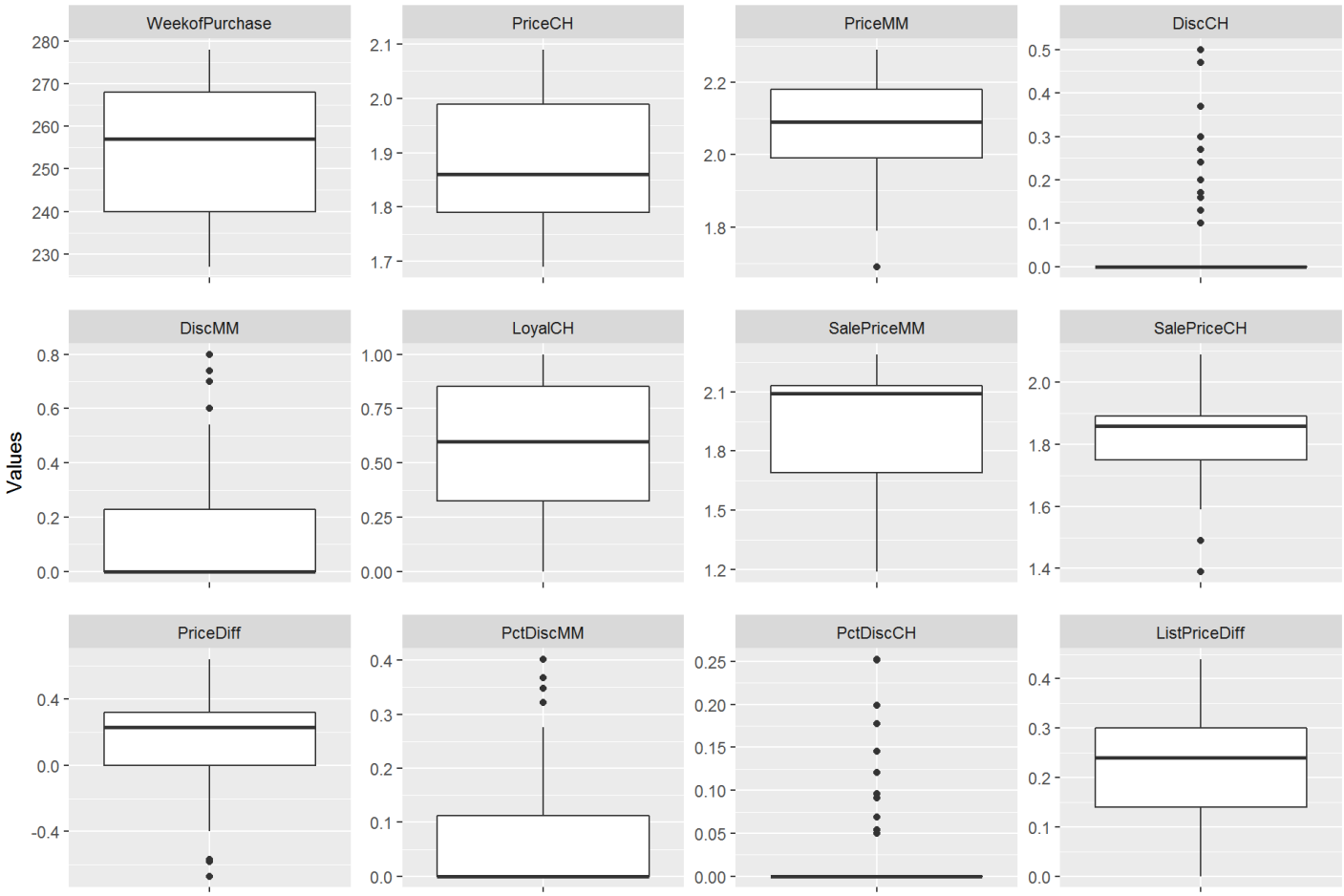

Figure 1: Boxplots of quantitative variables

In Figure 2, correlations between the variables are summarized. One can notice that some of the strong correlations are natural. 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.

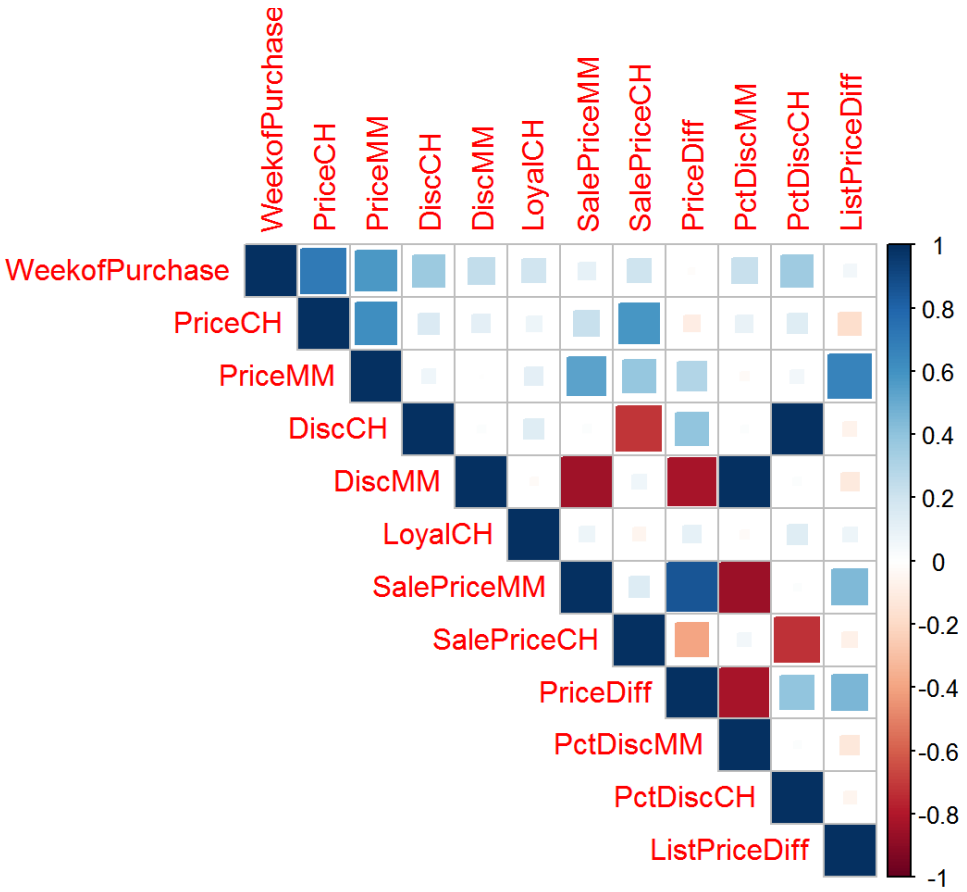

Figure 2: Correlation matrix plot of all quantitative variables

ANALYSIS BY DECISION TREE

Training and Test Dataset

The dataset has been divided into training and test datasets where 800 observations have been randomly assigned to the training set and the remaining 270 to the test set. Although variables SpecialCH and SpecialMM have unbalanced class, the proportions of the class would be reasonably retained with 800 observations in the training set.

In the test dataset, 0.637 of the total datapoints are CH. Thus, if we were to blindly guess CH, the test error rate would be 0.363.

Unpruned Decision Tree

A decision tree classifier has been used to predict whether a customer purchased Citrus Hill (CM) or Minute Maid Orange Juice (MM) based on remaining 17 variables.

First, the decision tree classifier has been fitted with the 800 training observations. The resulting unpruned single tree is shown in Figure 3 followed by summary information of the tree. The resulting tree has 7 terminal leaves. It is surprising that only four predictors, “LoyalCH”, “SalePriceMM”, “PriceDiff” and “SalePriceCH”, are used among 17 predictors included in the analysis. The misclassification training error rate is 0.201.

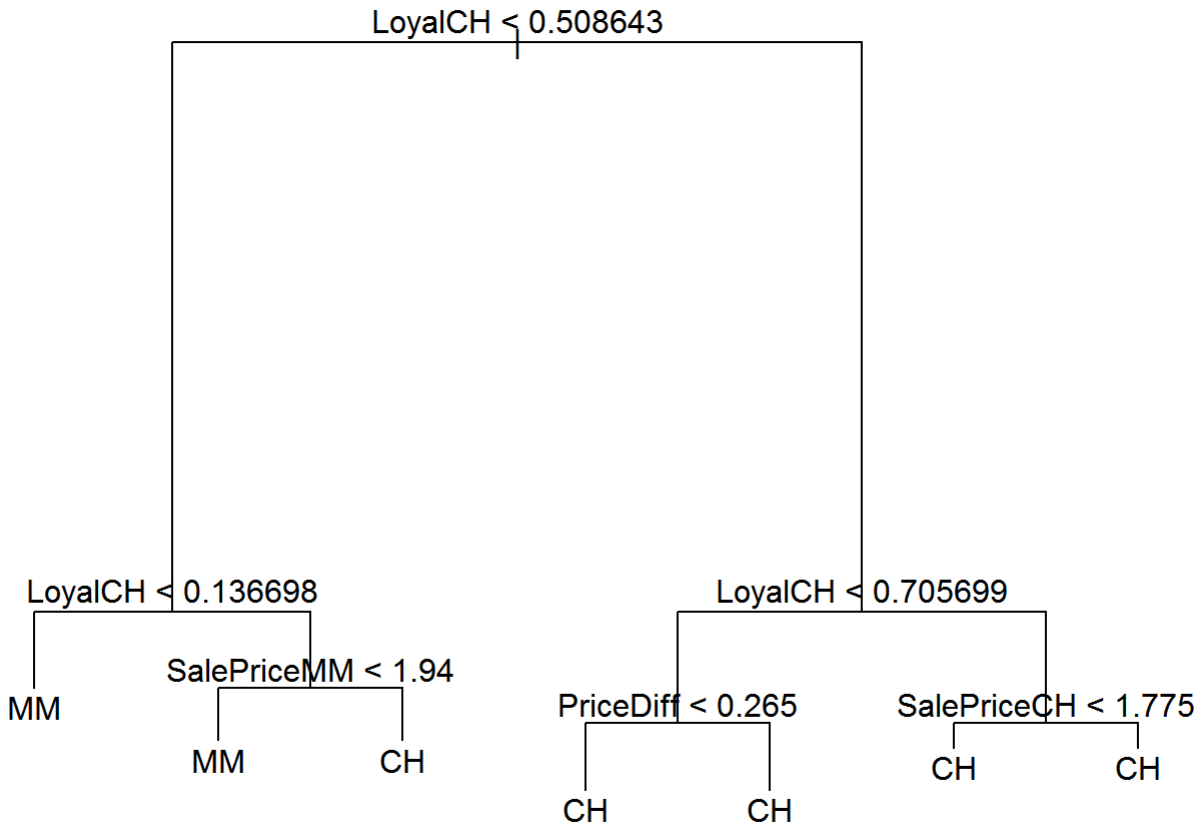

Figure 3: Unpruned decision tree resulting from the training set

```
Classification tree:
tree(formula = Purchase ~ ., data = OJ, subset = train.id)
Variables actually used in tree construction:
[1] "LoyalCH"     "SalePriceMM" "PriceDiff"   "SalePriceCH"
Number of terminal nodes:  7
Residual mean deviance:  0.786 = 623 / 793
Misclassification error rate: 0.201 = 161 / 800
```

The detailed node information of the tree are the following:

```
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 800 1000 CH ( 0.60 0.40 )
   2) LoyalCH < 0.508643 353  400 MM ( 0.28 0.72 )
     4) LoyalCH < 0.136698 102   50 MM ( 0.07 0.93 ) *
     5) LoyalCH > 0.136698 251  300 MM ( 0.37 0.63 )
      10) SalePriceMM < 1.94 118  100 MM ( 0.20 0.80 ) *
      11) SalePriceMM > 1.94 133  200 CH ( 0.51 0.49 ) *
   3) LoyalCH > 0.508643 447  400 CH ( 0.85 0.15 )
     6) LoyalCH < 0.705699 151  200 CH ( 0.68 0.32 )
      12) PriceDiff < 0.265 88  100 CH ( 0.50 0.50 ) *
      13) PriceDiff > 0.265 63   30 CH ( 0.92 0.08 ) *
     7) LoyalCH > 0.705699 296  100 CH ( 0.95 0.05 )
      14) SalePriceCH < 1.775 93    0 CH ( 1.00 0.00 ) *
      15) SalePriceCH > 1.775 203  100 CH ( 0.92 0.08 ) *
```

The top split assigns observations with LoyalCH < 0.508643 to the left side, which contains 353 observations with 72% of them being Minute Maid. The terminal leaf on the far right side of the tree is labeled as Citrus Hill, which contains 203 observations where 187 are indeed Citrus Hill (92%) and 16 are Minute Maid (8%).

The performance of the unpruned decision tree in Figure 3 has been evaluated using the test dataset. The resulting confusion matrix is given in Table 2, where the misclassification test error rate is 0.237.

Table 2: Confusion matrix resulting from prediction of unpruned tree using test set

	Actual	
Predicted	**CH**	**MM**
CH	147	39
MM	25	59

Pruning of Tree

Since there is a potential of overfitting the tree, pruning has been considered. To find the optimal tree size, 30-fold cross validation has been applied for each of the tree size. The number of misclassified labels versus the tree size is shown in Figure 4. Based on the plot, the optimal tree size is 2, which has the lowest number of misclassified labels.

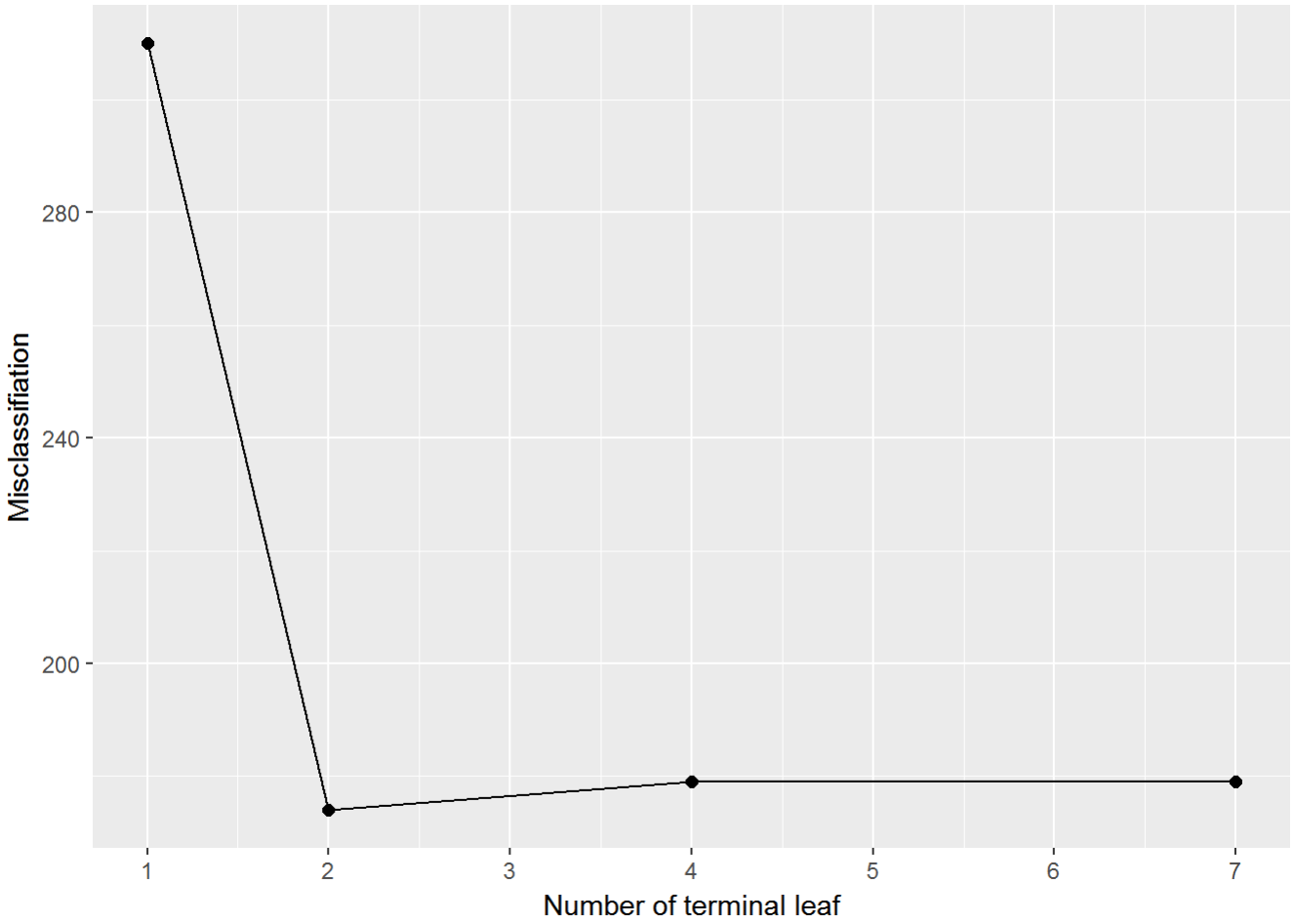

Figure 4: Cross validation from the training set to find optimal tree size

Thus, the tree has been pruned with 2 terminal leaves as shown in Figure 5. Compared with the unpruned tree of Figure 3, the tree has only one split where LoyalCH < 0.508643 is assigned MM and LoyalCH > 0.508643 is assigned CH. From the node information, the estimated misclassifiation rate of a test point can be compared between the two terminal leaves. The estimated misclassification rate on the left leaf is 0.3, which is the posterior probability of a datapoint being classified as CH given the datapoint is in the left leaf (predicted as MM). On the other hand, the misclassification rate on the right leaf is 0.1, which is the posterior probability of a datapoint classified as MM given the datapoint is in the right leaf. The overall misclassification training error on the pruned tree is 0.205.

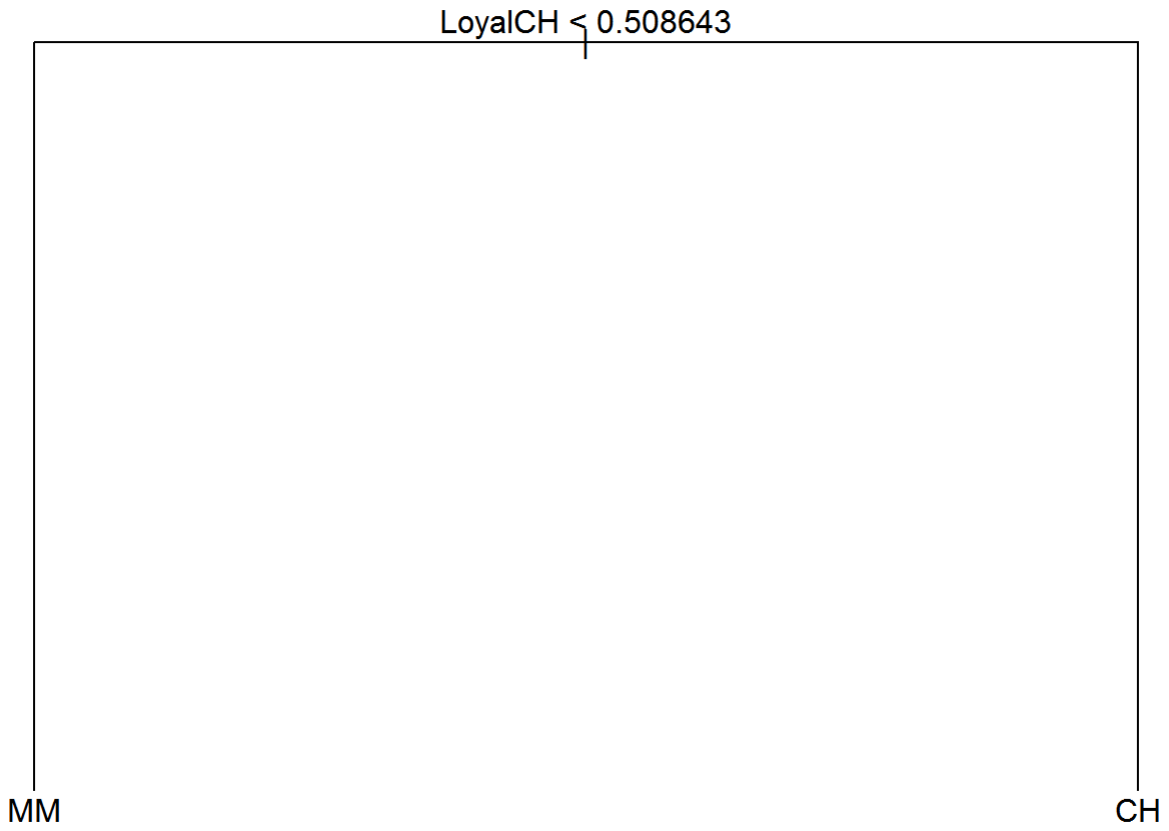

Figure 5: Pruned tree with optimal terminal leaf number of 2

```
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

1) root 800 1000 CH ( 0.6 0.4 )
  2) LoyalCH < 0.508643 353  400 MM ( 0.3 0.7 ) *
  3) LoyalCH > 0.508643 447  400 CH ( 0.9 0.1 ) *
```

The confusion matrix resulting from the pruned tree with test dataset is provided in Table 3 with overall misclassification test error rate of 0.1926. The misclassification rate for CH is 0.105 (= 16/(16+136)), and the misclassification rate for MM is 0.305 (= 36/(36+82)). The misclassification rate for each class from the test dataset agrees well with estimated values from the training set.

Table 3: Confusion matrix resulting from the pruned tree using the test dataset

	Actual	
Predicted	**CH**	**MM**
CH	136	16
MM	36	82

The training and test errors of unpruned and pruned trees are summarized and compared in Figure 5. For the unpruned tree, the test error is greater than the training error, while the test error is smaller than the training error for the pruned tree. The training errors from unpruned and pruned trees are very close. It is speculated that the

significant improvement in the test error of the pruned tree is due to reduction of overfitting when compared with the unpruned tree. By pruning the tree, the variance of the model has been reduced resulting in a model with a better balance between variance and bias.

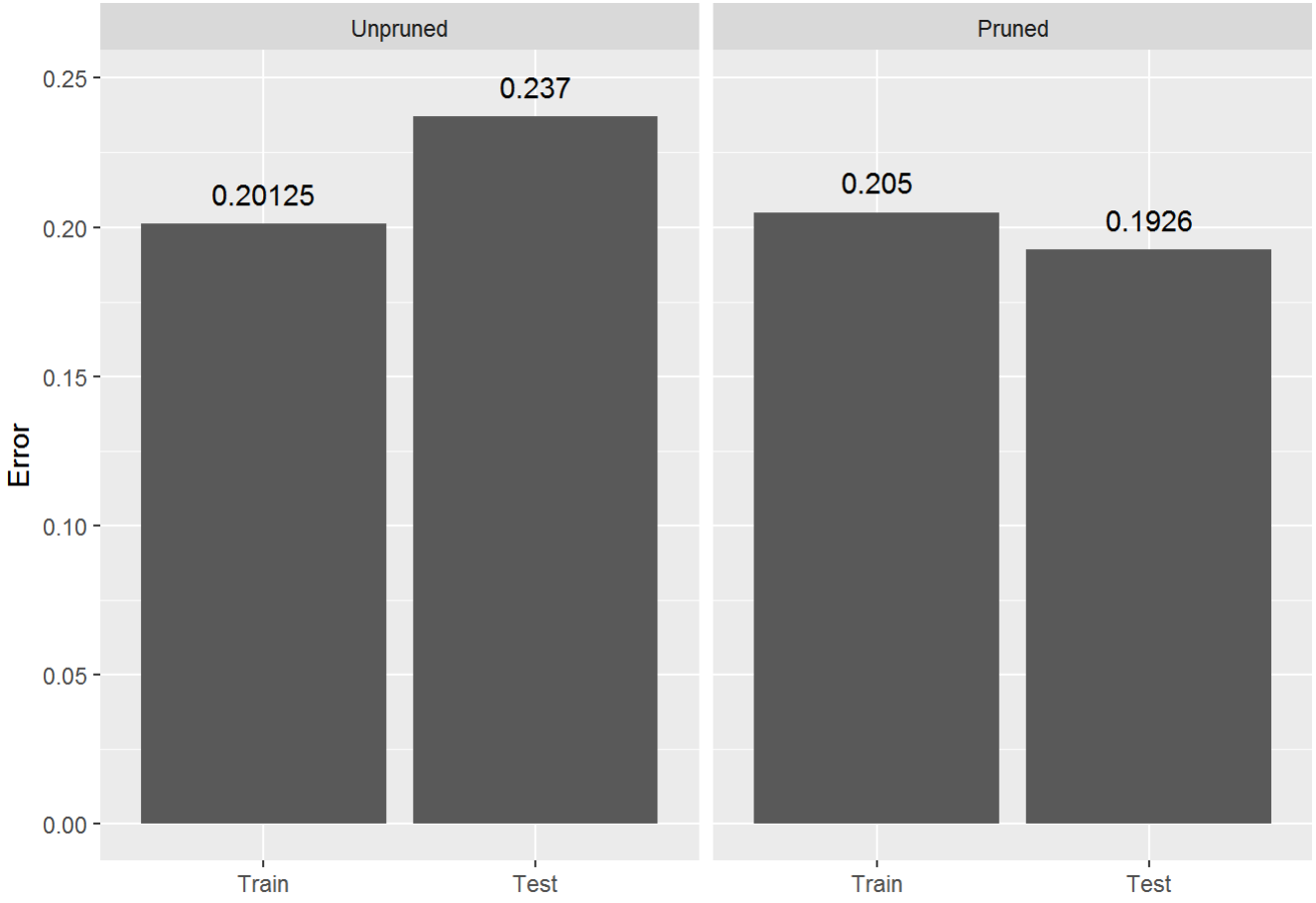

Figure 5: Comparison of errors from unpruned and pruned decision tree

BAGGING, RANDOM FOREST and BOOSTING

To improve the performance of the decision tree, three ensemble methods (Bagging, Random Forest, and Boosting) have been compared in this section.

Bagging

As shown in Figure 5, pruning the tree actually improved the accuracy of the classifier indicating that the unpruned tree may be overfitting the training dataset. Bagging can reduce the overfitting by lowering the variance coming from an indiviudal tree with aggregating fitted values from a large number of bootstrap samples when training the model.

For this dataset, 500 trees were selected, which is sufficiently large for the errors to be stablized. Bagging is a special case of the Random Forest with all predictors considered during the boostrapping from the training dataset. After fitting the Bagging classifier with training set, test set was used to analyze the performance. The resulting confusion matrix is provided in Table 4 where the misclassification test error rate is 0.1815.

Table 4: Confusion matrix resulting from Bagging classifier

	Actual	
Predicted	**CH**	**MM**
CH	147	24
MM	25	74

The top 8 most important variables based on node purity from the Bagging classifier are plotted in Figure 6.

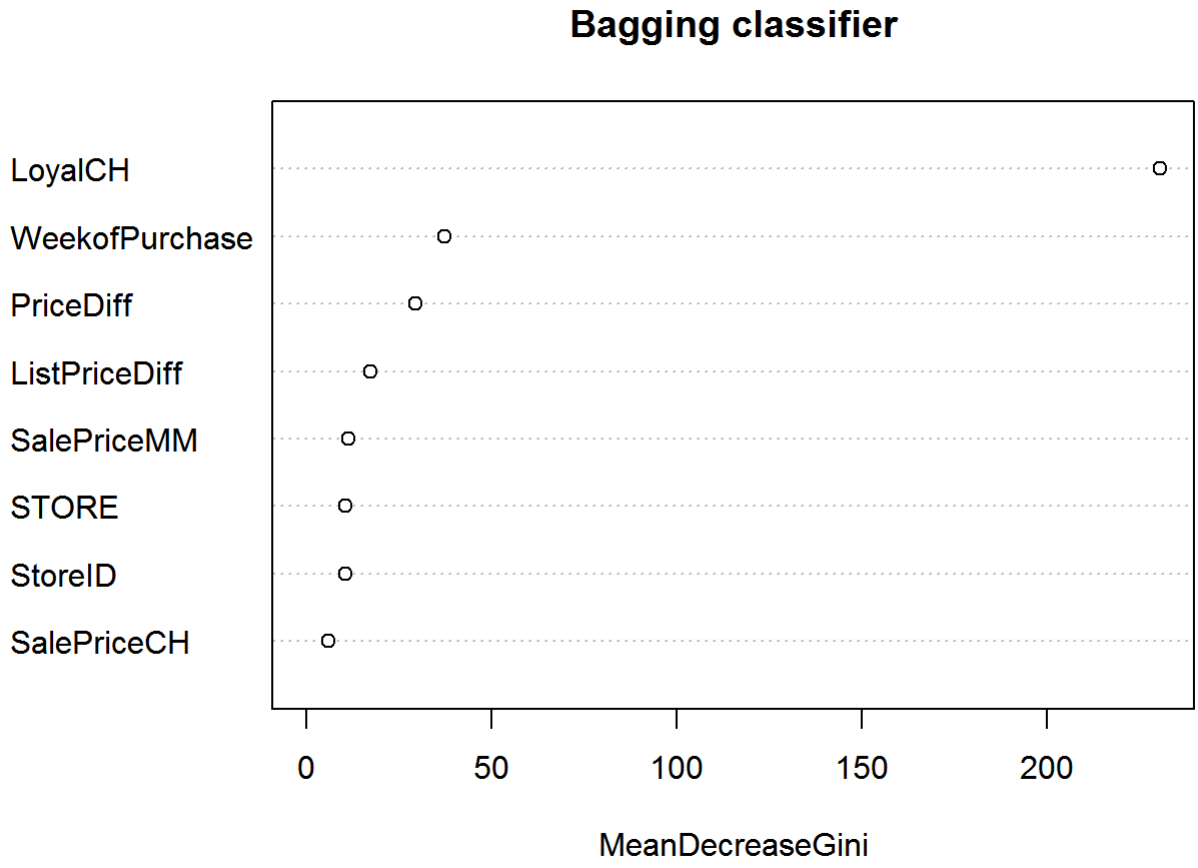

Figure 6: Variable importance plot from Bagging classifier

Random Forest

The Random Forest classifier combines bootstrapping of samples and random draws of predictors. By using the random sample of predictors at each split, the fitted values across trees are more independent. Thus, the strategy would be particularly helpful when there are several strong predictors that could dominate the structure of trees during bootstrapping. With random forest, the weak predictors may still contribute when those strong predictors are absent during the bootstrapping, and thus, the trees would be decorrelated and more indpendent.

The random forest has been run with 3 predictors sampled at each split. Total of 500 trees were used, which was sufficiently large enough so the errors became stable. The resulting confusion matrix from the test dataset is provided in Table 5 where the misclassification test error was 0.1741.

Table 5: Confusion matrix resulting from Random Forest classifier

	Actual

Table 5: Confusion matrix resulting from Random Forest classifier

	Actual	
Predicted	**CH**	**MM**
CH	150	25
MM	22	73

The top 8 most important variables from the Random Forest classifier are plotted in Figure 7.

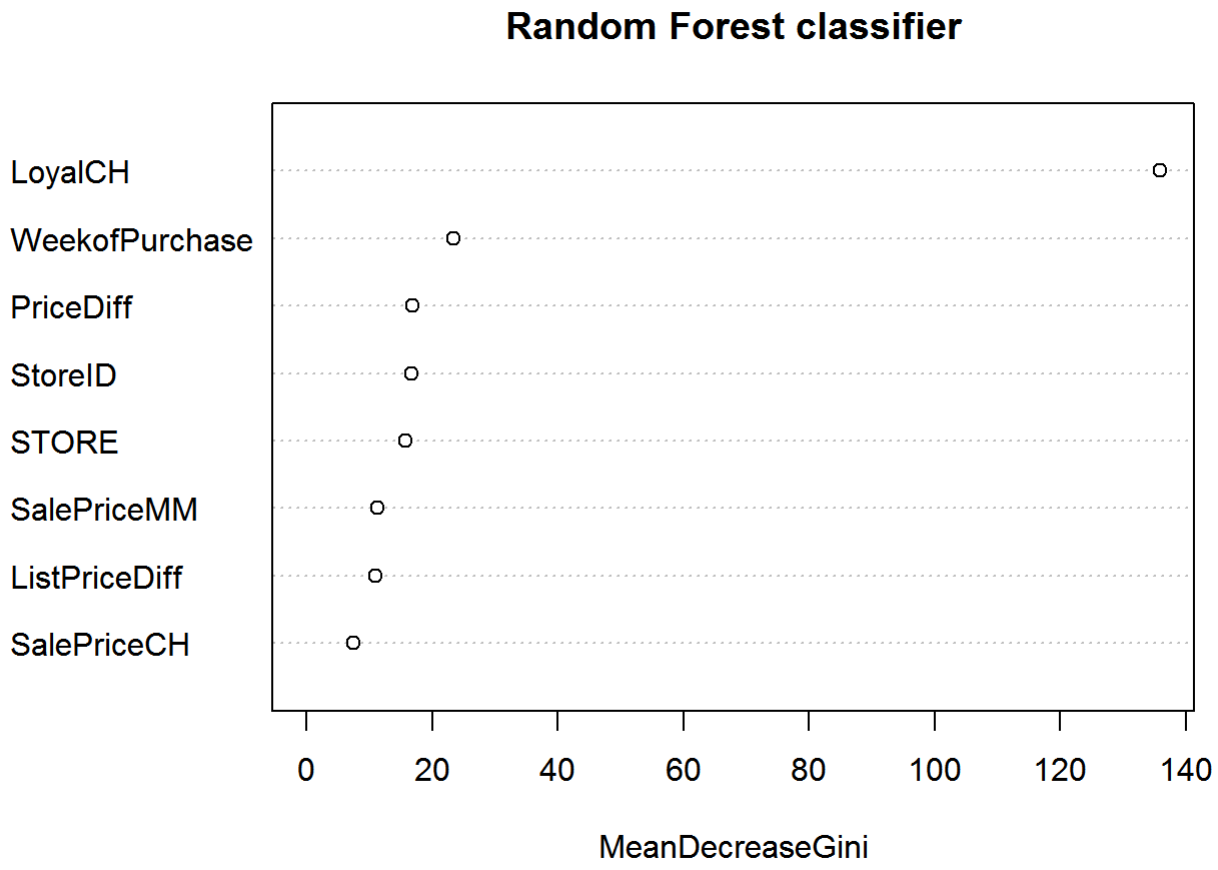

Figure 7: Variable importance plot from Random Forest classifier

Boosting

In Boosting, the trees are grown sequentially where a new tree is fitted using the residuals. By fitting small trees to the residuals, the Boosting method can capture areas where a model might not perform well.

The Boosting was performed with 5000 trees where interaction depth of 1 and shrinkage parameter of 0.001 were used. The resulting confusion matrix from the test dataset are given in Table 6 with misclassification error rate of 0.137.

Table 6: Confusion matrix resulting from Gradient Boosting classifier

	Actual	
Predicted	**CH**	**MM**
CH	156	21

Table 6: Confusion matrix resulting from Gradient Boosting classifier

	Actual	
Predicted	**CH**	**MM**
MM	16	77

The resulting variable importance from Boosting is shown in Figure 8. Comparing the feature importance plots, Figures 6, 7 and 8, LoyalCH was the most important feature selected by all ensemble classifiers. The Bagging and Random Forest selected WeekofPurchase as the second most important predictor whereas Boosting selected PriceDiff as the second most important. It is interesting to note that the WeekofPurchase was ranked 6th in Boosting classifier.

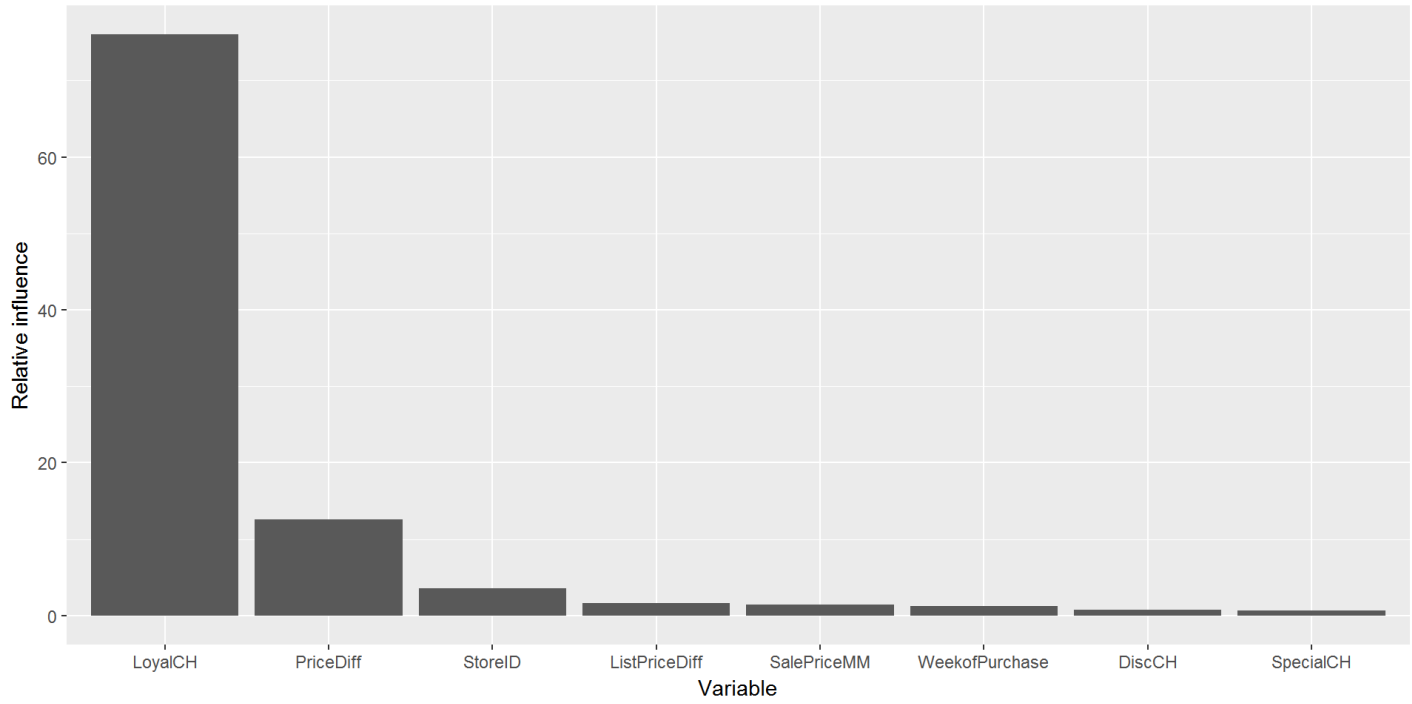

Figure 8: Variable importance plot from Gradient Boosting classifier

The comparison of misclassified error rates of all classifiers are summarized in Figure 9. For the given training set, the unpruned tree showed the highest test error rate. The test error of the pruned tree improved, which indicated that the pruning removed some of the overfitting in the unpruned tree. The three ensemble methods (Bagging, Random Forest, and Boosting) all outperformed the two single tree classifiers. Overall, the Gradient Boosting showed the best performance of all methods considered in this study.

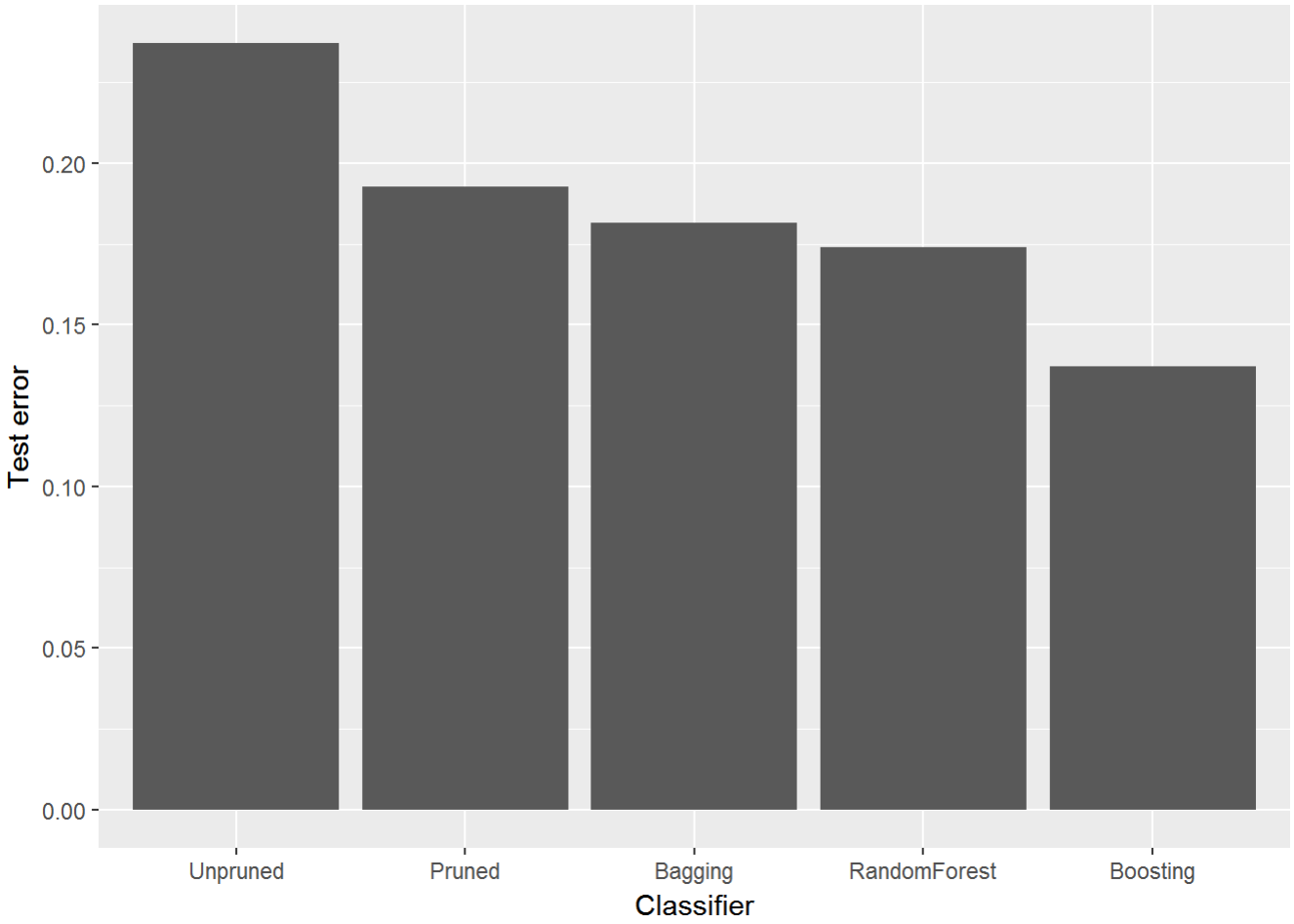

Figure 9: Comparison of test errors of all classifiers used in this study

The tree-based methods are known to be robust against outliers and multicollinearity issues. The analysis from OJ dataset also supports the robustness of tree-based learners. As observed earlier, the dataset had quite a few variables with outliers and strong corrleations. Nonetheless, the tree-based learners performed well with test error rate siginifcantly lower than just blindly guessing “CH”, which gives error rate of 0.363.

For further reliability regarding the test performace, it may be necessary to perform cross validation to assess the variablity in the test error rate. In particular, the decision tree learner may be more vulnerable with large variance in the model.

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. Tree-based classifiers have been evaluated in predicting whether a customer purchased CH or MM based on 17 variables. Following conclusions may be drawn from this study:

- The unpruned tree resulting from the training dataset had 7 terminal leaves with test error rate of 0.237. By pruning the tree to only 2 terminal leaves, the test error rate improved to 0.1926. The pruning procedure decreased the overfitting in the unpruned tree by reducing the variance in the classifier.
- The emsemble method considered in the study, Bagging, Random Forest and Boosting, all outperformed the single tree classifiers (unpruned and pruned). The highest performing classifier was Gradient Boosting

with test error rate of 0.137. This is significant improvement over just blindly guessing “CH”, which gives error rate of 0.363.

- The most important variable identified by all methods was LoyalCH, which was the customer brand loyalty for CH. The Gradient Boosting resulted with PriceDiff and StoreID as the second and third important variables.
- The tree-based methods were robust and performed well where the dataset had outliers and strong correlations among variables.

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)

sum(is.na(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")
```

```
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")

############################################################
### Sample stratification (Not used in the report)

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 = createDataPartition(feature.groups, times=1, p=0.732)

########################################
### ANALYSIS WITH DECISION TREE
########################################

### User function definition

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)
}

print.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)

   cat(paste("Overall accuracy =", metric[1]),
      paste("Error rate =", metric[2]),
      paste("Type I error =", metric[3]),
      paste("Type II error =", metric[4]),
```

```
      paste("Sensitivity =", metric[5]),
      paste("Specificity =", metric[6]),
      sep="\n")
}

print.cm = function(cm, lab.name, cap) {
   table.cm = unname(cm)
   colnames(table.cm) = lab.name
   rownames(table.cm) = lab.name


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

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

set.seed(23)
train.id = sample(1:nrow(OJ), 800)
#train.id = folds.id[[1]]     # Goes with stratified sampling (not used)
test.id = -train.id

Purchase = OJ$Purchase
OJ.test = OJ[test.id,]
Purchase.test = Purchase[test.id]

CH.ratio = sum(Purchase.test == "CH") / length(Purchase.test)

library(tree)

tree.OJ = tree(Purchase ~ ., data=OJ, subset=train.id)
summary_tree.OJ = summary(tree.OJ)
train.error.tree = summary_tree.OJ$misclass[1] / summary_tree.OJ$misclass[2]

### Figure 3: Unpruned decision tree resulting from the training set

par(mar=c(2,2,2,2))
plot(tree.OJ)
text(tree.OJ)

summary(tree.OJ)

tree.OJ
tree.pred = predict(tree.OJ, newdata=OJ.test, type="class")
tree1.cm = table(tree.pred, Purchase.test)
test.error.tree = my.metric(tree1.cm)[2]
#test.error.tree

print.cm(tree1.cm, colnames(tree1.cm), "Table 2: Confusion matrix resulting from <br> prediction
```

```
 of unpruned tree using test set")

### Figure 4: Cross validation from the training set to find optimal tree size

library(ggplot2)

set.seed(30)
cv.OJ = cv.tree(tree.OJ, FUN=prune.misclass, K=30)
opt.size = cv.OJ$size[which.min(cv.OJ$dev)]
#opt.size

ggplot() +
   geom_point(aes(x=cv.OJ$size, y=cv.OJ$dev), size=2) +
   geom_line(aes(x=cv.OJ$size, y=cv.OJ$dev)) +
   scale_x_continuous(breaks=c(1:8)) +
   labs(x="Number of terminal leaf", y="Misclassifiation")

prune.OJ = prune.misclass(tree.OJ, best=opt.size)
summary_prune.OJ = summary(prune.OJ)
train.error.pruned = summary_prune.OJ$misclass[1] / summary_prune.OJ$misclass[2]

### Figure 5: Pruned tree with optimal terminal leaf number of 2

par(mar=c(2,2,2,2))
plot(prune.OJ)
text(prune.OJ)

prune.OJ

tree.prune.pred = predict(prune.OJ, newdata=OJ.test, type="class")
tree.prune.cm = table(tree.prune.pred, Purchase.test)
test.error.pruned = my.metric(tree.prune.cm)[2]

### Table 3: Confusion matrix resulting from <br> the pruned tree using the test dataset

print.cm(tree.prune.cm, colnames(tree.prune.cm), "Table 3: Confusion matrix resulting from <br>
 the pruned tree using the test dataset")

### Figure 5: Comparison of errors from unpruned and pruned decision tree

error.unpruned = c(train.error.tree, test.error.tree)
error.pruned = c(train.error.pruned, test.error.pruned)

tab3 = cbind(error.unpruned, error.pruned)
rownames(tab3) = c("Train.Error","Test.Error")
colnames(tab3) = c("Unpruned", "Pruned")

tab3.melt = melt(tab3)

ggplot(data=tab3.melt) +
   geom_col(aes(x=Var1, y=value)) +
   geom_text(aes(x=Var1, y=value+0.01, label=as.vector(tab3))) +
   facet_wrap(~ Var2) +
   scale_x_discrete(labels=c("Train","Test")) +
```

```
   labs(x="", y="Error")

library(randomForest)

set.seed(30)
bag.OJ = randomForest(Purchase ~ ., data=OJ, subset=train.id,
                      mtry=ncol(OJ)-1, ntree=500, importance=T)
#bag.OJ

bag.OJ.pred = predict(bag.OJ, newdata=OJ.test, type="class")
bag.OJ.cm = table(bag.OJ.pred, Purchase.test)
test.error.bag = my.metric(bag.OJ.cm)[2]
#test.error.bag

### Table 4: Confusion matrix resulting <br/> from Bagging classifier

print.cm(bag.OJ.cm, colnames(bag.OJ.cm), "Table 4: Confusion matrix resulting <br/> from Bagging
 classifier")

varImpPlot(bag.OJ, sort=T, n.var=8, type=2, main="Bagging classifier")

set.seed(30)
rf.OJ = randomForest(Purchase ~ ., data=OJ, subset=train.id,
                     mtry=3, ntree=500, importance=T)
#rf.OJ

rf.OJ.pred = predict(rf.OJ, newdata=OJ.test, type="class")
rf.OJ.cm = table(rf.OJ.pred, Purchase.test)
test.error.rf = my.metric(rf.OJ.cm)[2]
#test.error.rf

### Table 5: Confusion matrix resulting <br/> from Random Forest classifier

print.cm(rf.OJ.cm, colnames(rf.OJ.cm), "Table 5: Confusion matrix resulting <br/> from Random Fo
rest classifier")

varImpPlot(rf.OJ, sort=T, n.var=8, type=2, main="Random Forest classifier")

library(gbm)

OJ.boost = OJ
OJ.boost$Purchase = factor(OJ.boost$Purchase, levels=c("CH","MM"), labels=c(0,1))
OJ.boost$Purchase = as.numeric(OJ.boost$Purchase)-1

set.seed(30)
boost.OJ = gbm(Purchase ~ ., data=OJ.boost[train.id, ], distribution="bernoulli",
               n.trees=5000, interaction.depth=2, shrinkage=0.001)

boost.OJ.pred = predict(boost.OJ, newdata=OJ.boost[test.id, ], n.trees=5000, type="response")

#summary(boost.OJ.pred)

boost.pred = rep("CH", length(boost.OJ.pred))
boost.pred[which(boost.OJ.pred > 0.5)] = "MM"
```

```
boost.OJ.cm = table(boost.pred, Purchase.test)

test.error.boost = my.metric(boost.OJ.cm)[2]

### Table 6: Confusion matrix resulting <br/> from Gradient Boosting classifier

print.cm(boost.OJ.cm, colnames(boost.OJ.cm), "Table 6: Confusion matrix resulting <br/> from Gra
dient Boosting classifier")

### Figure 8: Variable importance plot from Gradient Boosting classifier

summary.boost = summary(boost.OJ, plotit=F)[1:8,]
summary.boost$var = factor(summary.boost$var, levels=summary.boost$var)

ggplot(data=summary.boost) +
   geom_col(aes(x=var, y=rel.inf)) +
   labs(x="Variable", y="Relative influence")

### Figure 9: Comparison of test errors of all classifiers used in this study

test.error = c(test.error.tree, test.error.pruned, test.error.bag, test.error.rf, test.error.boo
st)
names(test.error) = c("Unpruned", "Pruned", "Bagging", "RandomForest", "Boosting")

test.error = melt(test.error)
test.error$classifier = factor(rownames(test.error), levels=rownames(test.error))

ggplot(data=test.error) +
   geom_col(aes(x=classifier, y=value)) +
   labs(x="Classifier", y="Test error")
```