BIOS 664 HW3

Meng-Ni Ho

4/8/2019

Background and Introduction

In this assignment, we will solve a classification problem using the `Caravan` dataset including in the R library `ILSR`. This data set includes 85 predictors that measure demographic characteristics for 5,822 individuals. The binary response variable (last column of the data matrix) is `Purchase`, which indicates whether or not a given individual purchases a caravan insurance policy. In this data set, only 6% of people purchased caravan insurance.

Load data:

```
require(ISLR)
```

```
## Loading required package: ISLR
```

```
attach(Caravan)
dim(Caravan)
```

```
## [1] 5822   86
```

```
names(Caravan)
```

```
##  [1] "MOSTYPE"  "MAANTHUI" "MGEMOMV"  "MGEMLEEF" "MOSHOOFD" "MGODRK"  
##  [7] "MGODPR"   "MGODOV"   "MGODGE"   "MRELGE"   "MRELSA"   "MRELOV"  
## [13] "MFALLEEN" "MFGEKIND" "MFWEKIND" "MOPLHOOG" "MOPLMIDD" "MOPLLAAG"
## [19] "MBERHOOG" "MBERZELF" "MBERBOER" "MBERMIDD" "MBERARBG" "MBERARBO"
## [25] "MSKA"     "MSKB1"    "MSKB2"    "MSKC"     "MSKD"     "MHHUUR"  
## [31] "MHKOOP"   "MAUT1"    "MAUT2"    "MAUT0"    "MZFONDS"  "MZPART"  
## [37] "MINKM30"  "MINK3045" "MINK4575" "MINK7512" "MINK123M" "MINKGEM" 
## [43] "MKOOPKLA" "PWAPART"  "PWABEDR"  "PWALAND"  "PPERSAUT" "PBESAUT" 
## [49] "PMOTSCO"  "PVRAAUT"  "PAANHANG" "PTRACTOR" "PWERKT"   "PBROM"   
## [55] "PLEVEN"   "PPERSONG" "PGEZONG"  "PWAOREG"  "PBRAND"   "PZEILPL" 
## [61] "PPLEZIER" "PFIETS"   "PINBOED"  "PBYSTAND" "AWAPART"  "AWABEDR" 
## [67] "AWALAND"  "APERSAUT" "ABESAUT"  "AMOTSCO"  "AVRAAUT"  "AAANHANG"
## [73] "ATRACTOR" "AWERKT"   "ABROM"    "ALEVEN"   "APERSONG" "AGEZONG" 
## [79] "AWAOREG"  "ABRAND"   "AZEILPL"  "APLEZIER" "AFIETS"   "AINBOED" 
## [85] "ABYSTAND" "Purchase"
```

```
data = Caravan
set.seed(109)
```

Feature selection:

Since there are 86 variables, use VIF and correlation to filter out relevant variables for training
1. VIF: include variables with VIF < 9
2. Spearman correlation: include variables with coefficients < 0.01

filter step 1: VIF

```
full_model = glm(factor(Purchase) ~ ., data = data, family = binomial(link = 'logit'))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
VIF = vif(full_model)
VIF = as.data.frame(VIF)
# select VIF < 9
VIF_select = VIF %>%
  tibble::rownames_to_column('parameters') %>%
  dplyr::filter(VIF < 9)

# extract parameters
vif_par_selected = VIF_select['parameters']
vif_par_selected = as.vector(vif_par_selected$parameters)
# create new data using extracted parameters
data_vif_filter = data %>% dplyr::select(Purchase, vif_par_selected)
print("dimensions for data_vif_filter: ")
```

```
## [1] "dimensions for data_vif_filter: "
```

```
print(dim(data_vif_filter))
```

```
## [1] 5822   35
```

```
print("column names for data_cor_filter: ")
```

```
## [1] "column names for data_cor_filter: "
```

```
print(names(data_vif_filter))
```

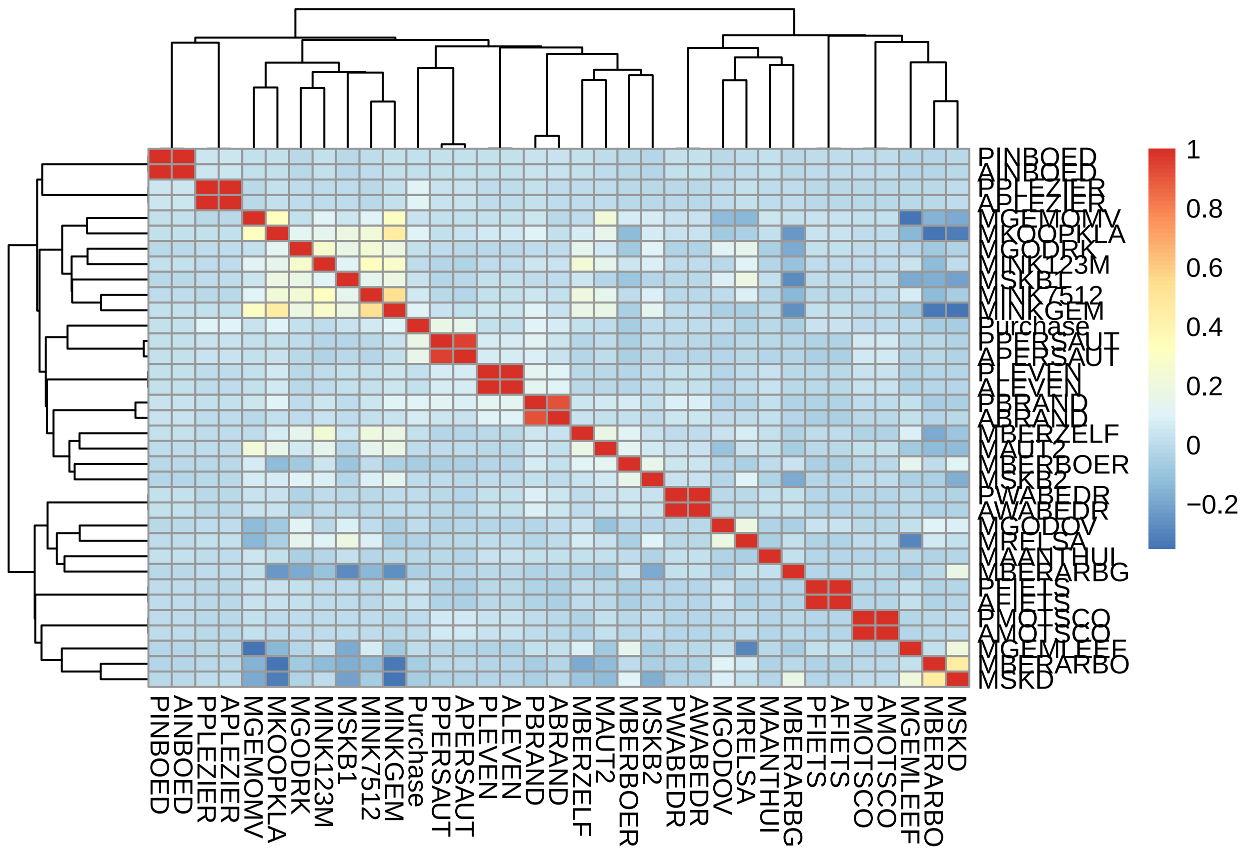
```
##  [1] "Purchase" "MAANTHUI" "MGEMOMV"  "MGEMLEEF" "MGODRK"   "MGODOV"
##  [7] "MRELSA"   "MBERZELF" "MBERBOER" "MBERARBG" "MBERARBO" "MSKB1"
## [13] "MSKB2"    "MSKD"     "MAUT2"    "MINK7512" "MINK123M" "MINKGEM"
## [19] "MKOOPKLA" "PWABEDR"  "PPERSAUT" "PMOTSCO"  "PLEVEN"   "PBRAND"
## [25] "PPLEZIER" "PFIETS"   "PINBOED"  "AWABEDR"  "APERSAUT" "AMOTSCO"
## [31] "ALEVEN"   "ABRAND"   "APLEZIER" "AFIETS"   "AINBOED"
```

filter step 2: Spearman Correlation

```
# transform Purchase from factor to numeric to perform correlation
levels(data_vif_filter$Purchase)[1] = 0
levels(data_vif_filter$Purchase)[2] = 1
data_vif_filter = transform(data_vif_filter, Purchase = as.numeric(Purchase))
# correlation heatmap for "data_vif_filter"
print("correlation heatmap for data_vif_filter")
```

```
## [1] "correlation heatmap for data_vif_filter"
```

```
pheatmap(cor(data_vif_filter, method = 'spearman'))
```



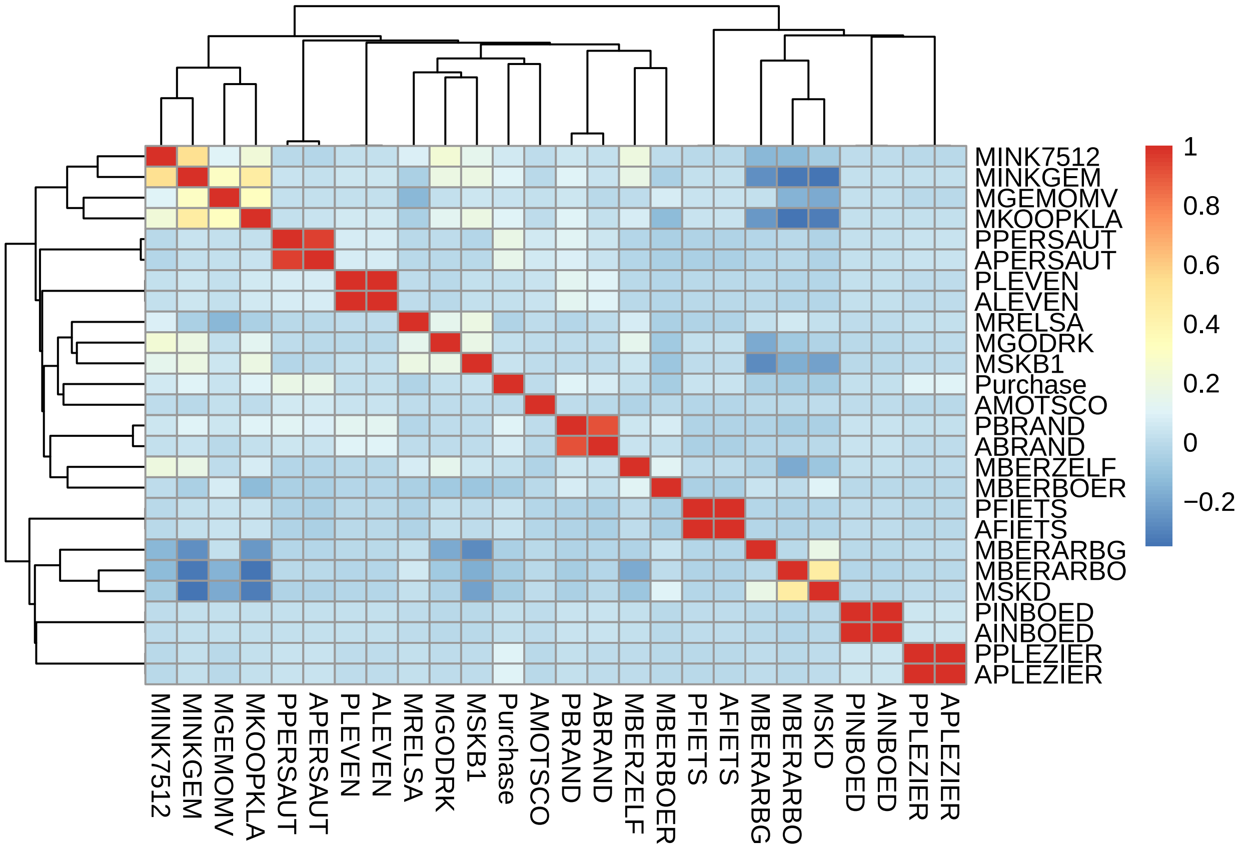
```
# Spearman correlation
corr = cor(data_vif_filter, method = 'spearman')
# select correlation > 0.01
corr_select = as.data.frame(corr) %>%
  tibble::rownames_to_column('parameters') %>%
  dplyr::filter(abs(Purchase) > 0.01) %>%
  dplyr::select(parameters, Purchase)

# extract parameters
corr_par_selected = corr_select['parameters']
corr_par_selected = as.vector(corr_par_selected$parameters)
# create new data with selected parameters
data_cor_filter = data_vif_filter %>% dplyr::select(Purchase, corr_par_selected)

# correlation heatmap for "data_cor_filter"
print("correlation heatmap for data_cor_filter")
```

```
## [1] "correlation heatmap for data_cor_filter"
```

```
pheatmap(cor(data_cor_filter, method = 'spearman'))
```



```r
print("dimensions for data_cor_filter: ")
```

```
## [1] "dimensions for data_cor_filter: "
```

```r
dim(data_cor_filter)
```

```
## [1] 5822   26
```

```r
print("column names for data_cor_filter: ")
```

```
## [1] "column names for data_cor_filter: "
```

```r
names(data_cor_filter)
```

```
##  [1] "Purchase" "MGEMOMV"  "MGODRK"   "MRELSA"   "MBERZELF" "MBERBOER"
##  [7] "MBERARBG" "MBERARBO" "MSKB1"    "MSKD"     "MINK7512" "MINKGEM"
## [13] "MKOOPKLA" "PPERSAUT" "PLEVEN"   "PBRAND"   "PPLEZIER" "PFIETS"
## [19] "PINBOED"  "APERSAUT" "AMOTSCO"  "ALEVEN"   "ABRAND"   "APLEZIER"
## [25] "AFIETS"   "AINBOED"
```

- `data_cor_filter` will be our final data for training

Create Training and Testing Datasets

Use the following R code to standardize the covariate data, and split the complete the data matrix into the testing (first 1000 observations) and training datasets (remaining observations)

```r
standardized.X=scale(data_cor_filter[,-1])
test =1:1000
```

```
train.X=standardized.X[-test ,]
test.X=standardized.X[test ,]
train.Y=Purchase [-test]
test.Y=Purchase [test]

# training set
train = cbind(as.data.frame(train.Y), train.X)
colnames(train)[colnames(train) == "train.Y"] = "Purchase"
# testing set
test = cbind(as.data.frame(test.Y), test.X)
colnames(test)[colnames(test) == "test.Y"] = "Purchase"
```

Problems

Question 1: Train the following ML algorithms on the training datasets, compute the expected classification errors for each method using cross-validation.

- logistic regression
- linear discriminant analysis
- support vector machine with linear kernel
- support vector machine with radial kernel
- feed-forward neural network (single or multiple hidden layers)
- AdaBoost

```
## 10-fold CV
fitControl = trainControl(method = "cv", number = 10, savePredictions = TRUE, classProbs=TRUE)

log_reg = train(factor(Purchase) ~ ., data = train, method = "glm", family = binomial(), trControl = fi

LDA = train(factor(Purchase) ~ ., data = train, method = "lda", trControl = fitControl)

svmLinear = train(factor(Purchase) ~ ., data = train, method = "svmLinear", trControl = fitControl)

svmRadial = train(factor(Purchase) ~ ., data = train, method = "svmRadial", trControl = fitControl)

nn = train(factor(Purchase) ~ ., data = train, method = "nnet", trControl = fitControl)

ada = train(factor(Purchase) ~ ., data = train, method = "ada", trControl = fitControl)

log_reg_err = 1 - log_reg$results['Accuracy'][[1]]
LDA_err = 1 - LDA$results['Accuracy'][[1]]
svmLinear_err = 1 - svmLinear$results['Accuracy'][[1]]
svmRadial_err = 1 - svmRadial$results['Accuracy'][[1]][1]
nn_err = 1 - nn$results['Accuracy'][[1]][1]
ada_err = 1 - ada$results['Accuracy'][[1]][1]

# Error
error = as.data.frame(cbind("-", format(log_reg_err, digits = 3),
                            format(LDA_err, digits = 3),
                            format(svmLinear_err, digits = 3),
                            format(svmRadial_err, digits = 3),
                            format(nn_err, digits = 3), format(ada_err, digits = 3)))
colnames(error) = c('classification error', 'logistic regression', 'LDA', 'SVM-Linear','SVM-Radial', 'N
```

- Results from training:

```
log_reg
```

```
## Generalized Linear Model
##
## 4822 samples
##   25 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4340, 4340, 4340, 4340, 4340, 4340, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9394444  0.01653153
```

```
LDA
```

```
## Linear Discriminant Analysis
##
## 4822 samples
##   25 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4340, 4340, 4340, 4340, 4341, 4340, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9369552  0.0492915
```

```
svmLinear
```

```
## Support Vector Machines with Linear Kernel
##
## 4822 samples
##   25 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4339, 4340, 4340, 4340, 4340, 4340, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9398593  -0.0004027218
##
```

```
## Tuning parameter 'C' was held constant at a value of 1
```

```
svmRadial
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 4822 samples
##   25 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4339, 4341, 4340, 4339, 4340, 4340, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.9400668   0.0000000000
##   0.50  0.9398589  -0.0004030807
##   1.00  0.9400668   0.0000000000
##
## Tuning parameter 'sigma' was held constant at a value of 0.03696486
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.03696486 and C = 0.25.
```

```
nn
```

```
## Neural Network
##
## 4822 samples
##   25 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4340, 4340, 4340, 4341, 4340, 4339, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##   1     0e+00  0.9400668  0.000000000
##   1     1e-04  0.9400668  0.000000000
##   1     1e-01  0.9400668  0.000000000
##   3     0e+00  0.9404817  0.012221773
##   3     1e-04  0.9400668  0.010823200
##   3     1e-01  0.9381970  0.002613017
##   5     0e+00  0.9369560  0.010357634
##   5     1e-04  0.9315627  0.019919278
##   5     1e-01  0.9384062  0.035644726
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 3 and decay = 0.
```

```
ada
```

```
## Boosted Classification Trees
##
## 4822 samples
##   25 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4340, 4340, 4340, 4340, 4339, 4340, ...
## Resampling results across tuning parameters:
##
##   maxdepth  iter  Accuracy   Kappa
##   1          50   0.9400664  0
##   1         100   0.9400664  0
##   1         150   0.9400664  0
##   2          50   0.9400664  0
##   2         100   0.9400664  0
##   2         150   0.9400664  0
##   3          50   0.9400664  0
##   3         100   0.9400664  0
##   3         150   0.9400664  0
##
## Tuning parameter 'nu' was held constant at a value of 0.1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were iter = 50, maxdepth = 1 and nu = 0.1.
```

- Classification error:

```
kable(error)
```

classification error	logistic regression	LDA	SVM-Linear	SVM-Radial	Neural Network	AdaBoost
-	0.0606	0.063	0.0601	0.0599	0.0599	0.0599

- Answer: svmLinear, Neural Network, Adaboost has the smaller error rate.

Question 2: Evaluate the classification algorithms on the testing dataset. Are the expected classification errors computed above accurate?

```
# logistic regression
log_reg_pred = predict(log_reg, test)
confusionMatrix(log_reg_pred,test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  937  58
##        Yes   4   1
##
##                Accuracy : 0.938
##                  95% CI : (0.9212, 0.9521)
##     No Information Rate : 0.941
##     P-Value [Acc > NIR] : 0.6862
##
##                   Kappa : 0.0222
##
##  Mcnemar's Test P-Value : 1.685e-11
```

```
##
##             Sensitivity : 0.99575
##             Specificity : 0.01695
##          Pos Pred Value : 0.94171
##          Neg Pred Value : 0.20000
##              Prevalence : 0.94100
##          Detection Rate : 0.93700
##    Detection Prevalence : 0.99500
##       Balanced Accuracy : 0.50635
##
##        'Positive' Class : No
##
```

```
log_reg_pred_err = 1-confusionMatrix(log_reg_pred,test$Purchase)$overall['Accuracy'][[1]]

# LDA
LDA_pred = predict(LDA, test)
confusionMatrix(log_reg_pred,test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  937  58
##        Yes   4   1
##
##                Accuracy : 0.938
##                  95% CI : (0.9212, 0.9521)
##     No Information Rate : 0.941
##     P-Value [Acc > NIR] : 0.6862
##
##                   Kappa : 0.0222
##
##  Mcnemar's Test P-Value : 1.685e-11
##
##             Sensitivity : 0.99575
##             Specificity : 0.01695
##          Pos Pred Value : 0.94171
##          Neg Pred Value : 0.20000
##              Prevalence : 0.94100
##          Detection Rate : 0.93700
##    Detection Prevalence : 0.99500
##       Balanced Accuracy : 0.50635
##
##        'Positive' Class : No
##
```

```
LDA_pred_err = 1-confusionMatrix(LDA_pred,test$Purchase)$overall['Accuracy'][[1]]

# svmLinear
svmLinear_pred = predict(svmLinear, test)
confusionMatrix(svmLinear_pred,test$Purchase)
```

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

```
##           Reference
## Prediction  No Yes
##        No  941  59
##        Yes   0   0
##
##                Accuracy : 0.941
##                  95% CI : (0.9246, 0.9548)
##     No Information Rate : 0.941
##     P-Value [Acc > NIR] : 0.5346
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 4.321e-14
##
##             Sensitivity : 1.000
##             Specificity : 0.000
##          Pos Pred Value : 0.941
##          Neg Pred Value :   NaN
##              Prevalence : 0.941
##          Detection Rate : 0.941
##    Detection Prevalence : 1.000
##       Balanced Accuracy : 0.500
##
##        'Positive' Class : No
##
```

```
svmLinear_pred_err = 1-confusionMatrix(svmLinear_pred,test$Purchase)$overall['Accuracy'][[1]]

# svmRadial
svmRadial_pred = predict(svmRadial, test)
confusionMatrix(svmRadial_pred,test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  940  58
##        Yes   1   1
##
##                Accuracy : 0.941
##                  95% CI : (0.9246, 0.9548)
##     No Information Rate : 0.941
##     P-Value [Acc > NIR] : 0.5346
##
##                   Kappa : 0.029
##
##  Mcnemar's Test P-Value : 3.086e-13
##
##             Sensitivity : 0.99894
##             Specificity : 0.01695
##          Pos Pred Value : 0.94188
##          Neg Pred Value : 0.50000
##              Prevalence : 0.94100
##          Detection Rate : 0.94000
##    Detection Prevalence : 0.99800
##       Balanced Accuracy : 0.50794
##
##        'Positive' Class : No
##
```

```
svmRadial_pred_err = 1-confusionMatrix(svmRadial_pred,test$Purchase)$overall['Accuracy'][[1]]

# NN
nn_pred = predict(nn, test)
confusionMatrix(nn_pred,test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  941  59
##        Yes   0   0
##
##                Accuracy : 0.941
##                  95% CI : (0.9246, 0.9548)
##     No Information Rate : 0.941
##     P-Value [Acc > NIR] : 0.5346
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 4.321e-14
##
##             Sensitivity : 1.000
##             Specificity : 0.000
##          Pos Pred Value : 0.941
##          Neg Pred Value :   NaN
##              Prevalence : 0.941
##          Detection Rate : 0.941
##    Detection Prevalence : 1.000
##       Balanced Accuracy : 0.500
##
##        'Positive' Class : No
##
```

```
nn_pred_err = 1-confusionMatrix(nn_pred,test$Purchase)$overall['Accuracy'][[1]]

# AdaBoost
ada_pred = predict(ada, test)
confusionMatrix(ada_pred,test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  941  59
##        Yes   0   0
##
##                Accuracy : 0.941
##                  95% CI : (0.9246, 0.9548)
##     No Information Rate : 0.941
##      P-Value [Acc > NIR] : 0.5346
##
##                    Kappa : 0
##
##  Mcnemar's Test P-Value : 4.321e-14
##
##             Sensitivity : 1.000
##             Specificity : 0.000
##          Pos Pred Value : 0.941
##          Neg Pred Value :   NaN
##              Prevalence : 0.941
##          Detection Rate : 0.941
##    Detection Prevalence : 1.000
##       Balanced Accuracy : 0.500
##
##        'Positive' Class : No
##
```

```
ada_pred_err = 1-confusionMatrix(ada_pred,test$Purchase)$overall['Accuracy'][[1]]



# Error
err_pred = as.data.frame(cbind("-", format(log_reg_pred_err, digits = 3), format(LDA_pred_err, digits =
                               format(svmRadial_pred_err, digits = 3), format(nn_pred_err, digits = 3),
colnames(err_pred) = c('classification error', 'logistic regression', 'LDA', 'SVM-Linear','SVM-Radial',
```

- Classification error for testing set:

```
kable(err_pred)
```

classification error	logistic regression	LDA	SVM-Linear	SVM-Radial	Neural Network	AdaBoost
-	0.062	0.06	0.059	0.059	0.059	0.059

- Error rate is similar to training data.

Question 3: Build a best ensemble classifier using the existing built classifiers from question 1. Is the performance better?

1. use all the algorithms for voting

```
# change level
log_reg_pred = mapvalues(log_reg_pred, from = c("No", "Yes"), to = c(0, 1))
LDA_pred = mapvalues(LDA_pred, from = c("No", "Yes"), to = c(0, 1))
svmLinear_pred = mapvalues(svmLinear_pred, from = c("No", "Yes"), to = c(0, 1))
svmRadial_pred = mapvalues(svmRadial_pred, from = c("No", "Yes"), to = c(0, 1))
nn_pred = mapvalues(nn_pred, from = c("No", "Yes"), to = c(0, 1))
ada_pred = mapvalues(ada_pred, from = c("No", "Yes"), to = c(0, 1))

# adding up prediction results using all the algorithms
vote_vec1 = as.vector(unfactor(log_reg_pred)) + as.vector(unfactor(LDA_pred)) + as.vector(unfactor(svmL
  as.vector(unfactor(svmRadial_pred)) + as.vector(unfactor(nn_pred)) + as.vector(unfactor(ada_pred))
table(vote_vec1)
```

```
## vote_vec1
##   0   1   2   3
## 993   1   5   1
```

```
# create empty ensemble prediction list
ensemble_pred1 = rep(0, length(vote_vec1))
# sum up predictions, voting >= 3, code ensemble_pred1 to 1, < 3 will be code to 0
ensemble_pred1[vote_vec1 >= 3] = 1
table(ensemble_pred1, test$Purchase)
```

```
##
## ensemble_pred1  No Yes
##              0 940  59
##              1   1   0
```

```
ensemble_accuracy1 = (937+1)/1000
ensemble_error1 = 1 - ensemble_accuracy1
```

2. use only svmLinear, NN, Adaboost (smaller error) for voting

```
# if only using three algorithms with lowest error: svmLinear, nn, adaboost
vote_vec2 = as.vector(unfactor(svmLinear_pred)) + as.vector(unfactor(nn_pred)) + as.vector(unfactor(ada
table(vote_vec2)
```

```
## vote_vec2
##    0
## 1000
```

```
ensemble_pred2 = rep(0, length(vote_vec2))
ensemble_pred2[vote_vec2 >= 3] = 1
table(ensemble_pred2, test$Purchase)
```

```
##
## ensemble_pred2  No Yes
##              0 941  59
```

```
ensemble_accuracy2 = (941)/1000
ensemble_error2 = 1 - ensemble_accuracy2
```

```
print(paste("ensemble error using all algorithms: ", format(ensemble_error1, digits = 3)))
```

```
## [1] "ensemble error using all algorithms:  0.062"
```

```
print(paste("ensemble error using svmLinear, NN, AdaBoost: ", format(ensemble_error2, digits = 3)))
```

```
## [1] "ensemble error using svmLinear, NN, AdaBoost:  0.059"
```

- Using svmLinear, NN, AdaBoost to build ensemble classifier produced smaller error than using all algorithm; however, the error is the same as using svmLinear, NN, AdaBoost alone. Therefore using ensemble classifier didn't reduce error.

Question 4: Comment on/propose possible approaches to improve the classifier.

One way that may improve classifier performance is feature selection (reduce dimension). Since the dataset contains 86 variables, we will only need the variables that are most relevant and with less degrees of noises. Using Pearson correlation can help filter out the features that are most correlated with the outcome ("Purchase"). Additionally, accessing collinearity can also filter out features that are highly correlated with each other. Above methods are the strategies I used before conducting ML training. However, most effective strategy is to conduct *principle component analysis (PCA)*, which can help reduce dimension but still retain the variation presented in the training data.