Homework Week 5

Kai Li

2/8/2017

## R Markdown

#### (a) Create training and test sets

After loading the data, it is realized that not only there are three factor elements in the dataset, STORE, Store7, and StoreID, the information in these three variables is repetitive too: the codes in STORE and StoreID can be directly translated into each other; both code systems can also be mapped to the binary coding system in Store7. As a result, only one element, STORE, is kept in the dataset.

# Read OJ dataset  
  
OJ <- OJ[,-c(3, 14)]  
OJ$STORE <- as.factor(OJ$STORE)

# Random sampling  
set.seed(1)  
train <- sample(nrow(OJ), 800)  
OJ.train <- OJ[train,]  
OJ.test <- OJ[-train,]

#### (b) Fit support vector classifier with training set

svmfit <- svm(formula = Purchase ~ .,  
 data = OJ.train,  
 kernel = "linear",  
 cost = 0.01)  
  
summary(svmfit)

##   
## Call:  
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "linear",   
## cost = 0.01)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 0.01   
## gamma: 0.05263158   
##   
## Number of Support Vectors: 434  
##   
## ( 217 217 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## CH MM

The results suggest that the svm classifier is based on cost = 0.01 and gamma = 0.056. Out of all the 800 data points, 628 are support vectors. As suggested by the low cost, the margin is quite wide.

#### (c) Test error rates on both sets.

Below are the results when the model is applied on the training set.

predict.training <- predict(svmfit, newdata = OJ.train)  
  
table(predict = predict.training,  
 original = OJ.train$Purchase)

## original  
## predict CH MM  
## CH 441 75  
## MM 53 231

accuracy\_rate.1 <- round(mean(predict.training == OJ.train$Purchase),  
 digits = 4) \* 100

The overall accuracy rate for the training set is 84%.

Below are the results of applying the model on the test set.

predict.test <- predict(svmfit, newdata = OJ.test)  
  
table(predict = predict.test,  
 original = OJ.test$Purchase)

## original  
## predict CH MM  
## CH 141 33  
## MM 18 78

accuracy\_rate.2 <- round(mean(predict.test == OJ.test$Purchase),  
 digits = 4) \* 100

The overall accuracy rate for the test set is 81.11%.

#### (d) Model tune

svmfit.tune <- tune(svm,   
 Purchase ~ .,  
 data = OJ.train,  
 kernel = "linear",  
 ranges = list(cost = c(0.01,  
 0.05,  
 0.1,  
 0.5,  
 1,  
 5,  
 10)))  
  
summary(svmfit.tune)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 1  
##   
## - best performance: 0.1575   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 0.01 0.16000 0.04743416  
## 2 0.05 0.16250 0.04859127  
## 3 0.10 0.16125 0.04387878  
## 4 0.50 0.15875 0.05001736  
## 5 1.00 0.15750 0.05177408  
## 6 5.00 0.16250 0.04965156  
## 7 10.00 0.16375 0.04803428

According to the results, cost = 0.5 is the most optimal cost among all the candidates given.

#### (e) Test new model on both sets

Below are the results of applying the new model on the training set.

best.model <- svmfit.tune$best.model  
  
predict.training.2 <- predict(best.model, newdata = OJ.train)  
  
table(predict = predict.training.2,  
 original = OJ.train$Purchase)

## original  
## predict CH MM  
## CH 438 72  
## MM 56 234

accuracy\_rate.3 <- round(mean(predict.training.2 == OJ.train$Purchase),  
 digits = 4) \* 100

The accuracy rate on the training test is 84%.

Below are the results of applying the new model on the test set.

predict.test.2 <- predict(best.model, newdata = OJ.test)  
  
table(predict = predict.test.2,  
 original = OJ.test$Purchase)

## original  
## predict CH MM  
## CH 140 30  
## MM 19 81

accuracy\_rate.4 <- round(mean(predict.test.2 == OJ.test$Purchase),  
 digits = 4) \* 100

The accuracy rate on the test test is 81.85%, slightly higher than the first model.

#### (f) radial kernel

The same model using radial kernal as part I was applied to the training and test sets. However, the model doesn't seen to return reasonable results. This might due to the very low cost parameter passing to the model, which result a too loose model. For example, if the cost is raised to 0.05, the results seem to be more meaningful.

svmfit.radial <- svm(Purchase ~ .,  
 data = OJ,  
 subset = train,  
 cost = 0.01,  
 kernel = 'radial')  
  
summary(svmfit.radial)

##   
## Call:  
## svm(formula = Purchase ~ ., data = OJ, cost = 0.01, kernel = "radial",   
## subset = train)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 0.01   
## gamma: 0.05263158   
##   
## Number of Support Vectors: 616  
##   
## ( 306 310 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## CH MM

predict.training.radial <- predict(svmfit.radial, OJ.train)  
  
table(predict = predict.training.radial,  
 original = OJ.train$Purchase)

## original  
## predict CH MM  
## CH 494 306  
## MM 0 0

accuracy\_rate.5 <- round(mean(predict.training.radial == OJ.train$Purchase),  
 digits = 4) \* 100  
  
predict.test.radial <- predict(svmfit.radial, OJ.test)  
  
table(predict = predict.test.radial,  
 original = OJ.test$Purchase)

## original  
## predict CH MM  
## CH 159 111  
## MM 0 0

accuracy\_rate.6 <- round(mean(predict.test.radial == OJ.test$Purchase),  
 digits = 4) \* 100

The model is tuned, and the best result is the one with cost = 0.5 and gamma = 0.1. The new model is applied to both the training and test sets. The results of radial model is summarized below.

svmfit.radial.tune <- tune(svm,   
 Purchase ~ .,  
 data = OJ.train,  
 kernel = "radial",  
 ranges = list(cost = c(0.01,  
 0.05,  
 0.1,  
 0.5,  
 1,  
 5,  
 10),  
 gamma = c(0.1,  
 0.5,  
 1,  
 2,  
 4)))  
  
summary(svmfit.radial.tune)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost gamma  
## 0.5 0.1  
##   
## - best performance: 0.18   
##   
## - Detailed performance results:  
## cost gamma error dispersion  
## 1 0.01 0.1 0.38250 0.06800735  
## 2 0.05 0.1 0.25500 0.06101002  
## 3 0.10 0.1 0.18250 0.03395258  
## 4 0.50 0.1 0.18000 0.03961621  
## 5 1.00 0.1 0.18375 0.03634805  
## 6 5.00 0.1 0.18875 0.03197764  
## 7 10.00 0.1 0.18250 0.03736085  
## 8 0.01 0.5 0.38250 0.06800735  
## 9 0.05 0.5 0.38250 0.06800735  
## 10 0.10 0.5 0.29000 0.06341004  
## 11 0.50 0.5 0.20375 0.03586723  
## 12 1.00 0.5 0.19750 0.02813657  
## 13 5.00 0.5 0.20625 0.05311479  
## 14 10.00 0.5 0.21250 0.05773503  
## 15 0.01 1.0 0.38250 0.06800735  
## 16 0.05 1.0 0.38250 0.06800735  
## 17 0.10 1.0 0.35125 0.04980866  
## 18 0.50 1.0 0.21375 0.03557562  
## 19 1.00 1.0 0.21750 0.03545341  
## 20 5.00 1.0 0.22000 0.04937104  
## 21 10.00 1.0 0.23250 0.05006940  
## 22 0.01 2.0 0.38250 0.06800735  
## 23 0.05 2.0 0.38250 0.06800735  
## 24 0.10 2.0 0.37000 0.06619626  
## 25 0.50 2.0 0.23250 0.02958040  
## 26 1.00 2.0 0.22750 0.03425801  
## 27 5.00 2.0 0.24250 0.04338138  
## 28 10.00 2.0 0.24125 0.04084609  
## 29 0.01 4.0 0.38250 0.06800735  
## 30 0.05 4.0 0.38250 0.06800735  
## 31 0.10 4.0 0.37875 0.06719840  
## 32 0.50 4.0 0.25750 0.03545341  
## 33 1.00 4.0 0.24000 0.03322900  
## 34 5.00 4.0 0.24625 0.04715886  
## 35 10.00 4.0 0.24625 0.04489571

best.model.2 <- svmfit.radial.tune$best.model  
  
predict.training.radial.2 <- predict(best.model.2, newdata = OJ.train)  
  
accuracy\_rate.7 <- round(mean(predict.training.radial.2 == OJ.train$Purchase),  
 digits = 4) \* 100  
  
predict.test.radial.2 <- predict(best.model.2, newdata = OJ.test)  
  
accuracy\_rate.8 <- round(mean(predict.test.radial.2 == OJ.test$Purchase),  
 digits = 4) \* 100

radial.table <- data.frame(Set = c("Train", "Test"),   
 Model\_origin = c(accuracy\_rate.5, accuracy\_rate.6),  
 Model\_tuned = c(accuracy\_rate.7, accuracy\_rate.8))  
  
pander(radial.table)

|  |  |  |
| --- | --- | --- |
| Set | Model\_origin | Model\_tuned |
| Train | 61.75 | 85.88 |
| Test | 58.89 | 82.96 |

#### (g) polymonial kernel

The same results as happened to radial kernal happened in this section as well. And as the cost is increased, the results are more meaningful.

svmfit.polynomial <- svm(Purchase ~ .,  
 data = OJ,  
 subset = train,  
 cost = 0.01,  
 kernel = 'polynomial',  
 degree = 2)  
  
summary(svmfit.polynomial)

##   
## Call:  
## svm(formula = Purchase ~ ., data = OJ, cost = 0.01, kernel = "polynomial",   
## degree = 2, subset = train)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: polynomial   
## cost: 0.01   
## degree: 2   
## gamma: 0.05263158   
## coef.0: 0   
##   
## Number of Support Vectors: 615  
##   
## ( 306 309 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## CH MM

predict.training.polynomial <- predict(svmfit.polynomial, OJ.train)  
  
table(predict = predict.training.polynomial,  
 original = OJ.train$Purchase)

## original  
## predict CH MM  
## CH 494 306  
## MM 0 0

accuracy\_rate.9 <- round(mean(predict.training.polynomial == OJ.train$Purchase),  
 digits = 4) \* 100  
  
predict.test.polynomial <- predict(svmfit.polynomial, OJ.test)  
  
table(predict = predict.test.polynomial,  
 original = OJ.test$Purchase)

## original  
## predict CH MM  
## CH 159 111  
## MM 0 0

accuracy\_rate.10 <- round(mean(predict.test.polynomial == OJ.test$Purchase),  
 digits = 4) \* 100

According to the result of the model tuning, the most optimal choice is cost = 5 and gamma = 0.1. And then the optimal model is applied to both sets. Summarized below are the results of this section.

svmfit.polynomial.tune <- tune(svm,   
 Purchase ~ .,  
 data = OJ.train,  
 kernel = "polynomial",  
 degree = 2,  
 ranges = list(cost = c(0.01,  
 0.05,  
 0.1,  
 0.5,  
 1,  
 5,  
 10),  
 gamma = c(0.1,  
 0.5,  
 1,  
 2,  
 4)))  
  
summary(svmfit.polynomial.tune)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost gamma  
## 0.05 2  
##   
## - best performance: 0.1775   
##   
## - Detailed performance results:  
## cost gamma error dispersion  
## 1 0.01 0.1 0.36375 0.04980866  
## 2 0.05 0.1 0.32625 0.03928617  
## 3 0.10 0.1 0.27500 0.04124790  
## 4 0.50 0.1 0.19625 0.03821086  
## 5 1.00 0.1 0.18625 0.04387878  
## 6 5.00 0.1 0.18250 0.04216370  
## 7 10.00 0.1 0.18125 0.03875224  
## 8 0.01 0.5 0.23125 0.02960973  
## 9 0.05 0.5 0.18750 0.04249183  
## 10 0.10 0.5 0.18000 0.04048319  
## 11 0.50 0.5 0.18125 0.04218428  
## 12 1.00 0.5 0.18125 0.04649149  
## 13 5.00 0.5 0.18500 0.04281744  
## 14 10.00 0.5 0.19250 0.04133199  
## 15 0.01 1.0 0.18625 0.04387878  
## 16 0.05 1.0 0.18250 0.04216370  
## 17 0.10 1.0 0.18125 0.03875224  
## 18 0.50 1.0 0.18375 0.04966904  
## 19 1.00 1.0 0.18625 0.05050096  
## 20 5.00 1.0 0.18750 0.04639804  
## 21 10.00 1.0 0.18875 0.04466309  
## 22 0.01 2.0 0.18125 0.04050463  
## 23 0.05 2.0 0.17750 0.04281744  
## 24 0.10 2.0 0.18500 0.05062114  
## 25 0.50 2.0 0.18875 0.04581439  
## 26 1.00 2.0 0.19125 0.04332131  
## 27 5.00 2.0 0.19125 0.04168749  
## 28 10.00 2.0 0.19500 0.04048319  
## 29 0.01 4.0 0.18000 0.03917553  
## 30 0.05 4.0 0.18500 0.05552777  
## 31 0.10 4.0 0.18625 0.04505013  
## 32 0.50 4.0 0.18875 0.04693746  
## 33 1.00 4.0 0.18625 0.04016027  
## 34 5.00 4.0 0.19625 0.04411554  
## 35 10.00 4.0 0.19125 0.04641674

best.model.3 <- svmfit.polynomial.tune$best.model  
  
predict.training.polynomial.2 <- predict(best.model.3, newdata = OJ.train)  
  
accuracy\_rate.11 <- round(mean(predict.training.polynomial.2 == OJ.train$Purchase),  
 digits = 4) \* 100  
  
predict.test.polynomial.2 <- predict(best.model.3, newdata = OJ.test)  
  
accuracy\_rate.12 <- round(mean(predict.test.polynomial.2 == OJ.test$Purchase),  
 digits = 4) \* 100

radial.table <- data.frame(Set = c("Train", "Test"),   
 Model\_origin = c(accuracy\_rate.9, accuracy\_rate.10),  
 Model\_tuned = c(accuracy\_rate.11, accuracy\_rate.12))  
  
pander(radial.table)

|  |  |  |
| --- | --- | --- |
| Set | Model\_origin | Model\_tuned |
| Train | 61.75 | 86 |
| Test | 58.89 | 81.48 |

#### Conclusion

Based on the overall accuracy rate, it can be concluded that radial model is the best model for this dataset, followed by polynomial model.