Support Vector Machines in Fixed Income

Ryan Finegan

11/11/2021

```r
### One of the best "Out of Box" Classifiers
# Use SVCs to predict the direction of the ten year yield deltas week to week
library(e1071)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
setwd("/Users/ryanfinegan/Documents")                # my working directory
df<-read.csv("10yrforecasting_r.csv")                # file for 10 year prediction
dates <- as.POSIXct(df$Dates, format = "%m/%d/%Y")   # converting to get just the year
df$Dates<-format(dates, format="%Y")                 # getting the year in dates
df.new = subset(df,select = c(direction,thirtyderivativelag1,thirtyderivativelag5,
                              movelag1,tenderivativelag1,movederivativelag1,dxyderivativelag1,
                              tenderivativelag5, dxyderivativelag5, movederivativelag5))
df.new <- df.new %>%
      mutate(direction = ifelse(direction == "Down",0,1))
```

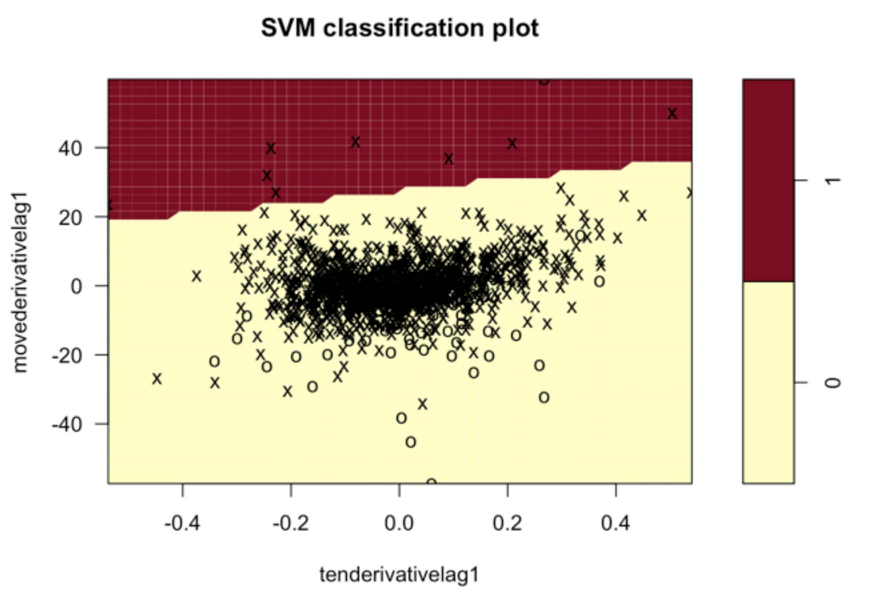
```r
# reproducible results
set.seed(3)
train=df.new[1:1200,]            # training split
test=df.new[1201:nrow(df.new),] # testing split
svc = svm(direction ~ .,data = train,cost=1,
          type = 'C-classification',kernel = 'linear',scale=FALSE) # I didn't standardize the data
y.pred = predict(svc, newdata = test[-1])  # predicting on everything but the target direction
(cm=table(test[, 1], y.pred))              # confusion matrix
```

```
##    y.pred
##       0   1
##   0 227  18
##   1 202   7
```

```r
correct=(cm[1,][1]+cm[2,][2])              # correct predictions
wrong=(cm[1,][2]+cm[2,][1])                # incorrect predictions
(acc=correct/(correct+wrong))              # accuracy
```

```
##         0
## 0.5154185
```

```r
plot(svc,train,movederivativelag1~tenderivativelag1)  # hyperplane plot given move derivative and ten year deriva
tive
```




```r
# move derivative very high is usually subject to weekly increase in ten year
```

```r
# finding the particular support vectors
vectors=svc$index
summary(svc)
```

```
##
## Call:
## svm(formula = direction ~ ., data = train, cost = 1, type = "C-classification",
##     kernel = "linear", scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  1120
##
##  ( 559 561 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

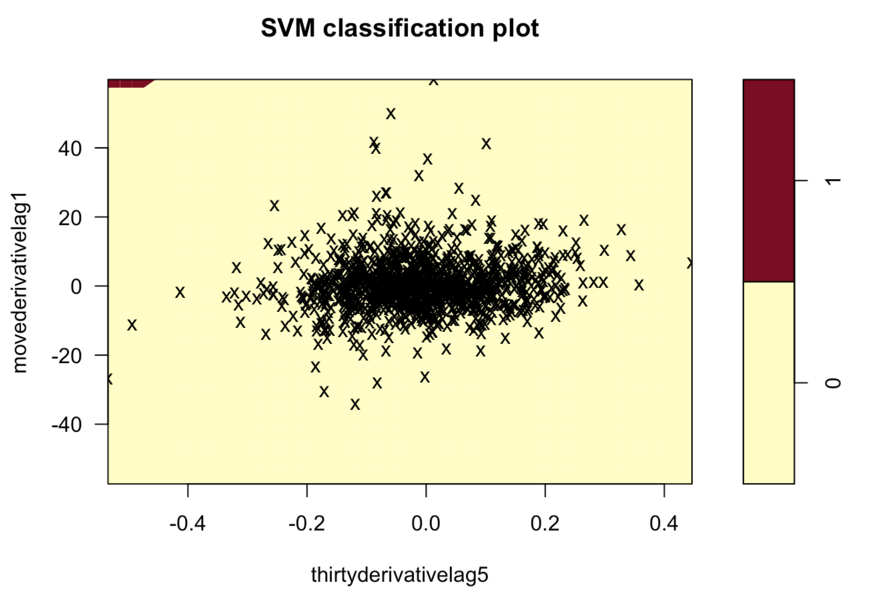
```r
# changing the cost to see a difference
set.seed(3)
svc = svm(direction ~ .,data = train,cost=0.1,
          type = 'C-classification',kernel = 'linear',scale=FALSE) # I didn't standardize the data
y.pred = predict(svc, newdata = test[-1])  # predicting on everything but the target direction
(cm=table(test[, 1], y.pred))              # confusion matrix
```

```
##    y.pred
##       0   1
##   0 245   0
##   1 209   0
```

```r
correct=(cm[1,][1]+cm[2,][2])              # correct predictions
wrong=(cm[1,][2]+cm[2,][1])                # incorrect predictions
(acc=correct/(correct+wrong))              # accuracy
```

```
##         0
## 0.5396476
```

```r
plot(svc,train,movederivativelag1~thirtyderivativelag5)  # hyperplane plot given move derivative and ten year der
ivative
```




```r
### cost function being small will mean many or more support vectors on or violating the margin
### cost function being large will mean few or fewer support vectors on or violating the margin
# model just predicted all decreases (non-linear could be better)
```

```r
### using random sampling and specific features for this part
### splitting the data set
set.seed(3)
df = subset(df,select = c(direction,thirtyderivativelag1,thirtyderivativelag2,
                          movelag1,movelag2,tenderivativelag1,
                          movederivativelag1,dxyderivativelag1,
                          tenderivativelag2, dxyderivativelag2,
                          movederivativelag2))
train_index <- sample(1:nrow(df), 800)  # trying random sample since it's classification
train <- df[train_index, ]              # training data split
test <- df[-train_index, ]              # testing data split
svm.linear <- svm(direction ~ .,        # direction dependent variable
                  data = train,         # training data
                  kernel = "linear",    # linear kernel
                  scale = T,            # scaling for True
                  cost = 0.01)          # small cost meaning many sv's on margin/violating margin
summary(svm.linear)                     # summary of the linear svc model
```

```
##
## Call:
## svm(formula = direction ~ ., data = train, kernel = "linear", cost = 0.01,
##     scale = T)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  749
##
##  ( 374 375 )
##
##
## Number of Classes:  2
##
## Levels:
##  Down Up
```

```r
table(train$direction[svm.linear$index])# showing the splits of up and down in the training data
```

```
##
## Down   Up
##  375  374
```

```r
# training and testing rates
data.frame(train_error = mean(predict(svm.linear, train) != train$direction),
           test_error = mean(predict(svm.linear, test) != test$direction))
```

```
##   train_error test_error
## 1     0.46125  0.4648712
```

```r
library(dplyr)
set.seed(3)
p.range <- seq(-2, 1, .2)
c.range <- 10^p.range
total <- 5
iter <- 3
cv.mat <- matrix(nrow = length(c.range), ncol = iter)
for (i in 1:iter) {
  svm.lin.tune <- tune(svm, direction ~ .,
                       data = train, kernel = "linear", scale = T,
                       ranges = list(cost = c.range),
                       tunecontrol = tune.control(sampling = "cross", cross = total))
  cv.mat[ ,i] <- svm.lin.tune$performances$error
}
(svm.linear.df <- data.frame(cost = c.range, CV_error = rowMeans(cv.mat)) %>%
  mutate(min_CV_error = as.numeric(CV_error == min(CV_error))))
```

```
##           cost  CV_error min_CV_error
## 1   0.01000000 0.4675000            0
## 2   0.01584893 0.4691667            0
## 3   0.02511886 0.4708333            0
## 4   0.03981072 0.4716667            0
## 5   0.06309573 0.4725000            0
## 6   0.10000000 0.4733333            0
## 7   0.15848932 0.4691667            0
## 8   0.25118864 0.4654167            0
## 9   0.39810717 0.4645833            0
## 10  0.63095734 0.4691667            0
## 11  1.00000000 0.4675000            0
## 12  1.58489319 0.4650000            0
## 13  2.51188643 0.4650000            0
## 14  3.98107171 0.4645833            0
## 15  6.30957344 0.4629167            1
## 16 10.00000000 0.4629167            1
```

```r
svm.linear.df %>% filter(min_CV_error == 1) %>% select(-min_CV_error)
```

```
##        cost  CV_error
## 1  6.309573 0.4629167
## 2 10.000000 0.4629167
```

```r
# lower cost => better accuracy (for this data set)
svm.linear <- svm(direction ~ ., data = train, kernel = "linear", scale = T, cost = 10^-0.8)
data.frame(train_error = mean(predict(svm.linear, train) != train$direction),
           test_error = mean(predict(svm.linear, test) != test$direction))
```

```
##   train_error test_error
## 1     0.45625  0.4625293
```

```r
library(ggplot2)
### radial kernel
set.seed(3)
cp.range <- seq(-2, 1, 0.2)
c.range <- 10^cp.range
total <- 10
iter <- 3
cv.mat <- matrix(nrow = length(cp.range), ncol = iter)
for (i in 1:iter) {
  svm_radial_tune <- tune(svm, direction ~ ., data = train, kernel = "radial", scale = T, ranges = list(cost = c.
range), tunecontrol = tune.control(sampling = "cross", cross = total))
  cv.mat[ ,i] <- svm_radial_tune$performances$error
}
svm.rad.df <- data.frame(cost = svm_radial_tune$performances$cost, CV_error = rowMeans(cv.mat)) %>%
  mutate(min_CV_error = as.numeric(CV_error == min(CV_error)))
### minimum cost and 0.01 cost since that did well earlier
svm.rad.1 <- svm(direction ~ ., data = train, kernel = "radial", scale = T, cost = 0.01)
svm.rad.2 <- svm(direction ~ ., data = train, kernel = "radial", scale = T, cost = 1)
svm.rad.df %>% filter(min_CV_error == 1 | cost == 0.01) %>% select(-min_CV_error) %>%
  cbind(data.frame(train_error = c(mean(predict(svm.rad.1, train) != train$direction),
                                   mean(predict(svm.rad.2, train) != train$direction)),
                   test_error = c(mean(predict(svm.rad.1, test) != test$direction),
                                  mean(predict(svm.rad.2, test) != test$direction))))
```

```
##        cost CV_error train_error test_error
## 1 0.0100000  0.46750      0.4675  0.4625293
## 2 0.1584893  0.46125      0.3350  0.4742389
```

```r
# polynomial kernel
cp.range <- c(seq(-2, 1, 0.2))
c.range <- 10^cp.range
total <- 10
iter <- 3
cv.mat <- matrix(nrow = length(cp.range), ncol = iter)
set.seed(720)
for (i in 1:iter) {
  svm.poly.tuner <- tune(svm, direction ~ ., data = train,
                         kernel = "polynomial",
                         scale = T,
                         ranges = list(degree = 2, cost = c.range),
                         tunecontrol = tune.control(sampling = "cross", cross = total))
  cv.mat[ ,i] <- svm.poly.tuner$performances$error
}

svm.polynomial.df <- data.frame(cost = svm.poly.tuner$performances$cost,
                                CV_error = rowMeans(cv.mat)) %>%
  mutate(min_CV_error = as.numeric(CV_error == min(CV_error)))
svm.poly.1 <- svm(direction ~ ., data = train,
                  kernel = "polynomial", scale = T, cost = 0.01, degree = 2)
svm.poly.2 <- svm(direction ~ ., data = train,
                  kernel = "polynomial", scale = T, cost = 10, degree = 2)
svm.polynomial.df %>% filter(cost == 10 | cost == 0.01) %>% select(-min_CV_error) %>%
  cbind(data.frame(train_error = c(mean(predict(svm.poly.1, train) != train$direction),
                                   mean(predict(svm.poly.2, train) != train$direction)),
                   test_error = c(mean(predict(svm.poly.1, test) != test$direction),
                                  mean(predict(svm.poly.2, test) != test$direction))))
```

```
##    cost  CV_error train_error test_error
## 1  0.01 0.4675000     0.46625  0.4625293
## 2 10.00 0.4858333     0.43500  0.4824356
```

```r
### best model
data.frame(kernel = c("Linear SVM", "Radial SVM", "Polynomial SVM"),
           CV_error = c(min(svm.linear.df$CV_error),
                        min(svm.rad.df$CV_error),
                        min(svm.polynomial.df$CV_error)),
           test_error = c(mean(predict(svm.linear, test) != test$direction),
                          mean(predict(svm.rad.2, test) != test$direction),
                          mean(predict(svm.poly.2, test) != test$direction)))
```

```
##           kernel  CV_error test_error
## 1     Linear SVM 0.4629167  0.4625293
## 2     Radial SVM 0.4612500  0.4742389
## 3 Polynomial SVM 0.4675000  0.4824356
```