Support Vector Machine Extension Project

Chris Hayduk

May 26, 2021

1 Expanded Mathematical Background

In the book "Introduction to Statistical Learning", the mathematics behind support vector machines was mostly abstracted away. The authors tended to favor high-level overviews of the math, while focusing mostly on code and examples. In this section, I use the authors' second book, "The Elements of Statistical Learning", in order to provide a more stable foundation for the math behind SVMs.

Recall that support vector classifiers fit an optimal separating hyperplane to a dataset. This hyperplane relied solely on "support vectors", which are points that landed inside the margin of the hyperplane, to calculate the separation. This decision boundary is always linear, but we can extend this method by enlarging the feature space using basis expansions such as polynomials or splines. In this way, we can fit a linear separating hyperplane in the enlarged space, which will result in a nonlinear boundary in the original space.

Support vector machines (SVMs) build upon this idea. We allow the dimension of the enlarged space to get very large - even infinite in some cases. We start by selecting basis functions

$$h_m(x), m = 1, \ldots, M$$

Then we fit the classifier using the input features

$$h(x_i) = (h_1(x_i), h_2(x_i), \ldots, h_M(x_i))$$

for $i = 1, \ldots, N$. We can represent the optimization problem and its solutions such that it only involves the input features via inner products. For certain choices of h, this inner product can be computed very cheaply. Now, we define the Lagrange dual function as follows,

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \langle h(x_i), h(x_{i'}) \rangle \tag{1}$$

where $\alpha_i, \alpha_{i'}$ are positive constraints for every i. The solution $f(x)$ can be written,

$$\begin{aligned} f(x) &= h(x)^T \beta + \beta_0 \\ &= \sum_{i=1}^{N} \alpha_i y_i \langle h(x), h(x_i) \rangle + \beta_0 \end{aligned} \tag{2}$$

We have that α_i, β_0 can be determined by solving the equation $y_i f(x_i) = 1$ for all x_i for which $0 < \alpha_i < C$, where C is the cost parameter.

Note that equations (1) and (2) only use $h(x_i)$ for all i in the context of taking the inner product. Hence, we do not need to know the basis function explicitly. We simply need to know the inner product in the enlarged feature space. This is represented as the kernel function,

$$K(x, x') = \langle h(x), h(x') \rangle$$

K is a symmetric positive definite function. Some popular examples of K in the SVM literature are as follows:

1. dth-Degree polynomial: $K(x, x') = (1 + \langle x, x' \rangle)^d$
2. Radial basis: $K(x, x') = \exp(-\gamma ||x - x'||^2)$
3. Neural network: $K(x, x') = \tanh(\kappa_1 \langle x, x' \rangle + \kappa_2)$

Let us now consider an example where the feature space has two inputs, X_1 and X_2, and we have chosen a 2nd-degree polynomial kernel. Then our kernel is given by,

$$\begin{aligned} K(X, X') &= (1 + \langle X, X' \rangle)^2 \\ &= (1 + X_1 X_1' + X_2 X_2')^2 \\ &= 1 + 2X_1 X_1' + 2X_2 X_2' + (X_1 X_1')^2 + (X_2 X_2')^2 + 2X_1 X_1' X_2 X_2' \end{aligned}$$

So we have that $M = 6$. We can choose $h_1(X) = 1$, $h_2(X) = \sqrt{2} X_1$, $h_3(X) = \sqrt{2} X_2$, $h_4(X) = X_1^2$, $h_5(X) = X_2^2$, and $h_6(X) = \sqrt{2} X_1 X_2$, then $K(X, X') = \langle h(X), h(X') \rangle$. Thus, the solution function is given by,

$$\hat{f}(x) = \sum_{i=1}^{N} \hat{\alpha}_i y_i K(x, x_i) + \hat{\beta}_0$$

The role of the C parameter is clearer in the enlarged feature space because perfect separation in this space is often possible. A large value of C will lead to a wiggly overfit in the original feature space. A small C encourages a small values of $||\beta||$, which makes $f(x)$ and thus the decision boundary smoother.

2 Data Analysis

Now that we have a bit stronger of a mathematical grounding for SVMs, we can shift our attention to applying the algorithm to a new data set. We will begin by using the `Smarket` data set from the `ISLR` package. We will attempt to classify the direction of a stock (whether it went up or down) based on the predictor variables.

```
#Load Smarket data
attach(Smarket)
names(Smarket)

## [1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"      "Lag5"
## [7] "Volume"    "Today"     "Direction"

#Check dimensions
dim(Smarket)

## [1] 1250    9

#Show summary of data
summary(Smarket)

##       Year           Lag1                Lag2                Lag3
##  Min.   :2001   Min.   :-4.922000   Min.   :-4.922000   Min.   :-4.922000
##  1st Qu.:2002   1st Qu.:-0.639500   1st Qu.:-0.639500   1st Qu.:-0.640000
##  Median :2003   Median : 0.039000   Median : 0.039000   Median : 0.038500
##  Mean   :2003   Mean   : 0.003834   Mean   : 0.003919   Mean   : 0.001716
##  3rd Qu.:2004   3rd Qu.: 0.596750   3rd Qu.: 0.596750   3rd Qu.: 0.596750
##  Max.   :2005   Max.   : 5.733000   Max.   : 5.733000   Max.   : 5.733000
##       Lag4                Lag5              Volume           Today
##  Min.   :-4.922000   Min.   :-4.92200   Min.   :0.3561   Min.   :-4.922000
##  1st Qu.:-0.640000   1st Qu.:-0.64000   1st Qu.:1.2574   1st Qu.:-0.639500
##  Median : 0.038500   Median : 0.03850   Median :1.4229   Median : 0.038500
##  Mean   : 0.001636   Mean   : 0.00561   Mean   :1.4783   Mean   : 0.003138
##  3rd Qu.: 0.596750   3rd Qu.: 0.59700   3rd Qu.:1.6417   3rd Qu.: 0.596750
##  Max.   : 5.733000   Max.   : 5.73300   Max.   :3.1525   Max.   : 5.733000
##  Direction
##  Down:602
##  Up  :648
##
##
##
##
```

This data set consists of percentage returns for the S& P500 stock index over 1250 days, from the beginning of 2001 until the end of 2005. For each date, the percentage returns for each of the five previous trading days are recorded, `Lag1` through `Lag5`. The following are also recorded:

- `Volume` - the number of shares traded on the previous day, in billions
- `Today` - the percentage return on the date in question
- `Direction` - whether the stock went up or down

We will compare the performance of an SVM to that of logistic regression on this data set, so we will begin by fitting logistic regression. We will begin by creating a training set of all years prior to 2005. We will then test the algorithms on a test set of the 2005 stock data:

```
#Create vector of True and False values
#Will use to subset training data
train = (Year<2005)

#Subset data for years before 2005
training_data = Smarket[train,]

#Get data for Year == 2005 for test data
test_data = Smarket[!train,]

#Display dimension of test data
dim(test_data)

## [1] 252   9

#Get response variables for test data
Direction.2005 = Direction[!train]
```

We will now train our logistic regression classifier:

```
#Fit GLM model to training data
glm.fits = glm(Direction~Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
               data = training_data, family = binomial)

#Predict responses for test data
glm.probs = predict(glm.fits, test_data, type="response")
```

Finally, let us compute the predictions for 2005 and compare them to the actual movements of the market over that time period:

```
#Create vector to contain GLM predictions
glm.pred = rep("Down", 252)
```

```r
#Switch any entry where prob > 0.5 to classify as "Up"
glm.pred[glm.probs>0.5] = "Up"

#Display table of predictions vs actual values
table(glm.pred, Direction.2005)

##         Direction.2005
## glm.pred Down Up
##     Down   77 97
##     Up     34 44

#Percent correctly classified
mean(glm.pred == Direction.2005)

## [1] 0.4801587

#Test error rate
mean(glm.pred != Direction.2005)

## [1] 0.5198413
```

We have that the test error rate is 52%, which is worse than random guessing! We will now examine whether an SVM is able to do a better job. We will use the `tune()` function in order to select the best choice of the γ and `cost` parameters for the SVM with radial kernel. We will use the radial kernel in order to capture any non-linearities that may arise in the stock data.

```r
set.seed(1)

#Optimize gamma and cost parameters for SVM
tune.out = tune(svm, Direction ~ Lag1+Lag2+Lag3+Lag4+Lag5+Volume,
                data=training_data, kernel = "radial",
                ranges = list(cost=c(0.1,1,10,100,1000),
                              gamma=c(0.5,1,2,3,4)))

#Summary of optimization
summary(tune.out)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
```

```
##    10     3
##
## - best performance: 0.4919192
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-01   0.5 0.5119495 0.03429289
## 2  1e+00   0.5 0.5231313 0.04953925
## 3  1e+01   0.5 0.5270505 0.04663609
## 4  1e+02   0.5 0.5120808 0.05663200
## 5  1e+03   0.5 0.5230707 0.04742496
## 6  1e-01   1.0 0.5089293 0.03608426
## 7  1e+00   1.0 0.5050505 0.04856195
## 8  1e+01   1.0 0.5050000 0.05708218
## 9  1e+02   1.0 0.5060101 0.05097908
## 10 1e+03   1.0 0.5090202 0.05243137
## 11 1e-01   2.0 0.5099293 0.03834069
## 12 1e+00   2.0 0.4950404 0.03184931
## 13 1e+01   2.0 0.4929899 0.04400072
## 14 1e+02   2.0 0.4959798 0.03901791
## 15 1e+03   2.0 0.4959798 0.03901791
## 16 1e-01   3.0 0.5099293 0.03834069
## 17 1e+00   3.0 0.5049394 0.03522618
## 18 1e+01   3.0 0.4919192 0.04043063
## 19 1e+02   3.0 0.4919192 0.04043063
## 20 1e+03   3.0 0.4919192 0.04043063
## 21 1e-01   4.0 0.5099293 0.03834069
## 22 1e+00   4.0 0.5019798 0.02629745
## 23 1e+01   4.0 0.5039798 0.03639452
## 24 1e+02   4.0 0.5039798 0.03639452
## 25 1e+03   4.0 0.5039798 0.03639452
```

```r
#Choose the best model from the optimization
svmfit = tune.out$best.model

#Predict response variables from test_data
svmpred= predict(svmfit, test_data)

#Display predicted response vs actual response
table(svmfit = svmpred, Direction.2005)
```

```
##       Direction.2005
## svmfit Down Up
##   Down   37 45
##   Up     74 96
```

```r
#Compute the test error rate
mean(svmpred != Direction.2005)

## [1] 0.4722222
```

We see here that our SVM with a radial kernel has a test error of 47.2%! While still a rather high figure, we have significantly improved over logistic regression with the same set of predictor variables, and now are able to perform better than random guessing. Hence, the SVM with a radial kernel must be capturing some inherent non-linearities in the stock data, which logistic regression was unable to model. However, let us examine our logistic regression model once again:

```r
summary(glm.fits)

##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial, data = training_data)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.302  -1.190   1.079   1.160   1.350
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.191213   0.333690   0.573    0.567
## Lag1        -0.054178   0.051785  -1.046    0.295
## Lag2        -0.045805   0.051797  -0.884    0.377
## Lag3         0.007200   0.051644   0.139    0.889
## Lag4         0.006441   0.051706   0.125    0.901
## Lag5        -0.004223   0.051138  -0.083    0.934
## Volume      -0.116257   0.239618  -0.485    0.628
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1383.3  on 997  degrees of freedom
## Residual deviance: 1381.1  on 991  degrees of freedom
## AIC: 1395.1
##
## Number of Fisher Scoring iterations: 3
```

We see that the predictors are all non-significant, meaning we should likely remove some of them in order to improve the fit. `Lag1` and `Lag2` both have by far the lowest p-values of the predictors (though still far from significant), so let us try fitting the logistic regression and SVM with `Lag1` and `Lag2` as our only two predictors.

```r
#Fit GLM model to training data with only 2 predictors
glm.fits = glm(Direction~Lag1 + Lag2,
               data = training_data, family = binomial)

glm.probs = predict(glm.fits, test_data, type="response")

glm.pred = rep("Down", 252)

glm.pred[glm.probs>0.5] = "Up"

table(glm.pred, Direction.2005)

##         Direction.2005
## glm.pred Down  Up
##     Down   35  35
##     Up     76 106

mean(glm.pred == Direction.2005)

## [1] 0.5595238

mean(glm.pred != Direction.2005)

## [1] 0.4404762
```

We see now that the test error rate for logistic regression has fallen to 44%! Let us perform the same analysis with the SVM:

```r
tune.out = tune(svm, Direction ~ Lag1+Lag2,
                data=training_data, kernel = "radial",
                ranges = list(cost=c(0.1,1,10,100,1000),
                              gamma=c(0.5,1,2,3,4)))

summary(tune.out)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   0.1   0.5
##
## - best performance: 0.4930808
```

```
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-01   0.5 0.4930808 0.04731145
## 2  1e+00   0.5 0.5080808 0.04404921
## 3  1e+01   0.5 0.5080909 0.03495052
## 4  1e+02   0.5 0.5051313 0.06235634
## 5  1e+03   0.5 0.4980909 0.06342086
## 6  1e-01   1.0 0.4980505 0.04489672
## 7  1e+00   1.0 0.5051010 0.04521609
## 8  1e+01   1.0 0.5101111 0.05171153
## 9  1e+02   1.0 0.5020707 0.04492083
## 10 1e+03   1.0 0.4990000 0.05243953
## 11 1e-01   2.0 0.5090606 0.04417970
## 12 1e+00   2.0 0.5100909 0.05458737
## 13 1e+01   2.0 0.5020202 0.04833033
## 14 1e+02   2.0 0.5020505 0.04154427
## 15 1e+03   2.0 0.5120303 0.03484585
## 16 1e-01   3.0 0.5000808 0.04844704
## 17 1e+00   3.0 0.5080707 0.04395064
## 18 1e+01   3.0 0.5020909 0.05662105
## 19 1e+02   3.0 0.5010101 0.02274754
## 20 1e+03   3.0 0.5200404 0.04790161
## 21 1e-01   4.0 0.4980909 0.05163961
## 22 1e+00   4.0 0.5060303 0.04500869
## 23 1e+01   4.0 0.5050101 0.03530226
## 24 1e+02   4.0 0.5230505 0.03296187
## 25 1e+03   4.0 0.5019798 0.03721501

svmfit = tune.out$best.model

svmpred= predict(svmfit, test_data)

table(svmfit = svmpred, Direction.2005)

##       Direction.2005
## svmfit Down  Up
##   Down   23  15
##   Up     88 126

mean(svmpred != Direction.2005)

## [1] 0.4087302
```

Again, we see that the SVM outperforms logistic regression: this time with a test error rate of 40.9%! We must also note that guessing "Up" for every data point in the test set would yield a test error rate of about 46%. Hence, in both the SVM and logistic regression previously, guessing "Up" every time would have fared better. However, after adjusting to only using `Lag1` and `Lag2`, both the SVM and logistic regression outperform this mark - a very important error rate to surpass.

Due to the consistently higher performance of our SVM with a radial kernel, it is quite likely that there are non-linearities in the `Smarket` data set. The SVM is able to model these non-linearities much more effectively than logistic regression, and so does a better job of classification.