Stat 542 HW3

Yizhou Zhang

Mar 14th, 2018

Problem 1a:Use ONLY the training data: Fit SVM with different tuning parameters

Use the `train` function in `caret` package to tune the parameters through CV.

```
q1a = subset(q1_tr, label == 4 | label ==9)
q1a_tst = subset(q1_tst, label == 4 | label ==9)
cv_5 = trainControl(method = "cv", number = 5)
set.seed(12345)
svmlinear = train(as.factor(label)~., data = q1a,method = 'svmLinear2',
                  trControl = cv_5, tuneLength = 5)
set.seed(12345)
svm_grid = expand.grid(C = 1:10,
                       scale =0.01,
                        degree = 1:5)
svmPoly = train(as.factor(label)~., data = q1a,method = 'svmPoly',
                trControl = cv_5, tuneLength = 5)
svmRadial = train(as.factor(label)~., data = q1a,method = 'svmRadial',
                  trControl = cv_5, tuneLength = 5)
```

I used the `train` function from the `caret` package to perform the tuning process. By specifying `tuneLength = 5`, the function is automatically tuned over five values of `C`: 0.25, 0.50, 1.00, 2.00, 4.00, and automatically chooses other parameters too. By specifying `method` as `svmLinear2`, `svmPoly`, `svmRadial`, different kernels were chosen. The cross-validation is five-fold, and the tuning metric is RMSE.

The results show that a SVM model with parameters `kernel = Polynomial, degree =2, scale = 0.1, C = 0.25` has achieved the highest performance with accuracy at 99.53%.

Problem 1b:Apply your selected model to the testing data and report the performance.

```
calc_acc = function(actual, predicted) {
  mean(actual == predicted)
}
pred = predict(svmPoly, q1a_tst )
acc = calc_acc(q1a_tst$label, pred)
```

Applyiing the `svmPoly` model to the test set achieves an accuracy at 0.98. According to the data documentation in the book ElemStatLearning , a 2.5% rate, or 97.5% accuracy rate would be excellent. The chosen model outperforms this standard.

Problem 1c:Comment on the following: 1)What are the advantages and disadvantages of different kernels that you used. 2)Is cross-validation effective on this analysis? Is there any potential pitfall that you could think of?

Linear kernel is most suitable for a situation when the classes are well separated with a relatively big margin in between. It also have less parameters to choose from. Furthermore, it has smaller number of support vectors thus is more computationally efficient. The `svmlinear` model has 73 support vectors which is the smallest among all three kernels. However, it is not suitable for a case when the separating hyperplane is nonlinear.

Polynomial kernel performs the best out of the three kernels for this dataset. Its nonlinear hyperplane is more flexible than linear kernel and is suitable for more complex data generating process. However, it has the most number of tuning parameters compared to the other two kernels, thus has the largest model uncertainty and computational burden.

Radial kernel is most suitable for a situation when the classes display a "donut" pattern (in 2-dimensional space), or when the separating hyperplane forms up a closed-circle pattern. It uses more support vectors than linear and polynomial kernels, as drawing a circle hyperplane naturally touches more points thus is more computationally heavy. In this data, the radial kernel svm uses `1296` support vectors, or the whole sample. This indicates that the radial svm is not able to draw a circled hyperplane between the two classes, and it has the lowest CV-accuracy.

Cross-Validation seems effective for the `svmPoly` model as an accuracy rate greater than 97.5% has been achieved. However, CV in itself might suffer from a few pitfalls. The first is parameter selection : how to choose the parameter values to tune over. Potentially, there is an infinite number of choices. Ideally, the tuning process should exhibit an "U" curve over the tuning space and the lowest point of the curve would be chosen. Secondly, the held-out data is not entirely removed from the training set, and the training set for each fold highly overlaps with each other. Ideally, the modeler should use another independent "test" set that is not used as training data in any fold.

Problem 2a:write your own LDA code.

Recall the discriminant function :

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k \ - \frac{1}{2}\mu_k\Sigma^{-1}\mu_k \ + log(\pi_k)$$

To manually develop the algorithm, we need to first calculate μ_k for each class, the priors π_k, and the Σ matrix.

calculate the Σ matrix by the formula:

$$\widehat{\Sigma} = \frac{1}{n-K}\sum_{k=1}^{K}\sum_{y_i=k}(x-\mu_k)(x-\mu_k)^T$$

With the sigma matrix in hand, now get the coefficients of the linear form:

$$\delta_k(x) = x^T w_k \ + b_k, \ where \ \ w_k = \Sigma^{-1}\mu_k \ , b_k = -\frac{1}{2}\mu_k\Sigma^{-1}\mu_k \ + log(\pi_k)$$

Start with the coefficient vector w_k

```
for (i in 0:9){
  temp  = paste("w", i, sep = "_")
  temp2  = eval(parse(text =paste("mu", i, sep = "_")))
  assign(temp, sigma_inv %*% temp2)
  rm(temp)
  rm(temp2)
}
```

Then get b_k. Note that π_k is not playing a role since the prior of every class is $\frac{1}{10}$

```
for (i in 0:9){
  temp  = paste("b", i, sep = "_")
  temp2  = eval(parse(text =paste("mu", i, sep = "_")))
  assign(temp, -0.5*  t(temp2) %*%  sigma_inv %*% temp2)
  rm(temp)
  rm(temp2)
}
```

The next step is to calculate the distribution matrix by self-defining a function `lda_pred` which generates the lda prediction results when the model is fit to a dataset.

```
lda_pred = function(df){
for (i in 0:9){
temp  = paste("class", i, sep = "_")
temp2  = eval(parse(text =paste("w", i, sep = "_")))
temp3  = eval(parse(text =paste("b", i, sep = "_")))
assign(temp, as.matrix(subset(df, select = -label )) %*% temp2
       + as.matrix(rep(temp3, dim(df)[1])))
}

discriminant = cbind(class_0, class_1, class_2, class_3,class_4,
                     class_5,class_6,class_7,class_8,class_9)

pred = apply(discriminant, 1, which.max) -  rep(1,length(discriminant))

return(pred)
}
#Also define a function to calculate prediction accuracy.
calc_acc = function(actual, predicted) {
  mean(actual == predicted)
}
pred_tst =lda_pred(p2tst)
```

Apply the function on the training set. The accuracy is 0.8256

Problem 2b:write code for Regularized DA.

The discriminant function of QDA is

$$\delta_k(x) = -\frac{1}{2}log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + log(\pi_k)$$

The key is to fit Σ_k for each class, and then linearly combine it with the total covariance matrix. Here I will start with $\alpha = 0.5$.

```
#get the nine covariance matrices, sig_k

for (i in 0:9){
  temp = paste("sig", i, sep = "_")
  temp1 = eval(parse(text=paste("demeaned", i, sep = "_")))
  assign(temp, (t(as.matrix(temp1)) %*% as.matrix(temp1))/5990)

  rm(temp)
  rm(temp1)
}
#Augment the covariance matrices with the original across-class covariance matrix.
for (i in 0:9){
temp =  paste("sigma", i, sep = "_")
temp1 = eval(parse(text=paste("sig", i, sep = "_")))
assign(temp, 0.5*sum + 0.5*temp1)

  rm(temp)
  rm(temp1)
}
```

now fit a function for qda. Again, note that the $\log(\pi_k)$ term is left out due to equal densities of all classes.

The following code generates the term $-\frac{1}{2}log|\Sigma_k|$

```
for (i in 0:9){
temp  = paste("b_q2", i, sep = "_")
temp2  = eval(parse(text =paste("sigma", i, sep = "_")))
assign(temp, -0.5* sum(log(svd(temp2)$d)))
rm(temp)
rm(temp2)
}
```

The accuracy applied on the test set is 0.8256.

Problem 2c: Compare the three methods

LDA model has the lowest computational burden, as it assumes a linear separating hyperplane between the classes. Therefore it assumes that the classes are relatively well separated, and that the covariance structure is the same across different classes.

QDA relaxes the assumption of identifical covariance matrices, and allows for nonlinear separating hyperplanes. However, QDA is not solvable when the covariance matrix of any class is singular, and it is computationally heavy.

RDA circumvents the limitation of QDA by guaranteeing a invertible covariance matrix, meanwhile relaxing the assumption of identical covariance matrix. Its main limitation is the parameter alpha needs tuning.

Problem 3a:formulate the dual problem of the linear separable SVM optimization problem into a form that can be solved by `solve.QP`

First obtain the results through the `e1071` package's `svm` function, and compute the coefficients and intercept.

```
sv = e1071::svm(as.factor(V3)~., data = q3_dat,kernel="linear",
                scale=FALSE, type="C-classification")
```

```
# get the slope and intercept terms from e1071 sum:
W = rowSums(sapply(1:length(sv$coefs), function(i) sv$coefs[i]*sv$SV[i,]))
svmline = c(sv$rho/W[2], -W[1]/W[2])
```

The setup of the `solve.QP` function is

$$\underset{\alpha_i}{\text{argmin}} \frac{1}{2}\beta^T D\beta - d^T\beta \quad s.t \;\; A^T\beta \geq b_0$$

The corresponding dual form of SVM is:

$$\underset{\alpha_i}{\text{argmin}} \frac{1}{2}\sum_{i,j=1}^{N} \alpha_i \left(y_i x_i\right)^T (y_j x_j)\, \alpha_j - \sum_{i=1}^{N} \alpha_i \quad s.t \;\; \alpha_i \geq 0 \; for \; \forall i \; , \; \sum_{i=1}^{N} \alpha_i y_i = 0$$

Under this sepcification, the parameters are:

$$D = \left(y_i x_i\right)^T (y_j x_j)\, , d = \mathbf{1}_n * 1b_0 = \mathbf{0}_(n+1) * 1A = \begin{bmatrix} y_1 & y_2 & \dots & y_n \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Additionally specifying `meq = 1` guarantees that only the first constraint is an equality constraint, or $\sum_{i=1}^{N} \alpha_i y_i = 0$. Note that in the above specification, D is positive semidefinite because it is a symmetric matrix. Adding it with a diagonal matrix with 10e-5 as the diagonal terms would guarantee positive definiteness. Mathematically in an optimization problem, adding the epsilon terms to D is equivalent to adding the epsilon term to the betas. This means that we must specify $\alpha_i > 10e-5$ as the condition to select support vectors, instead of using $\alpha_i > 0$.

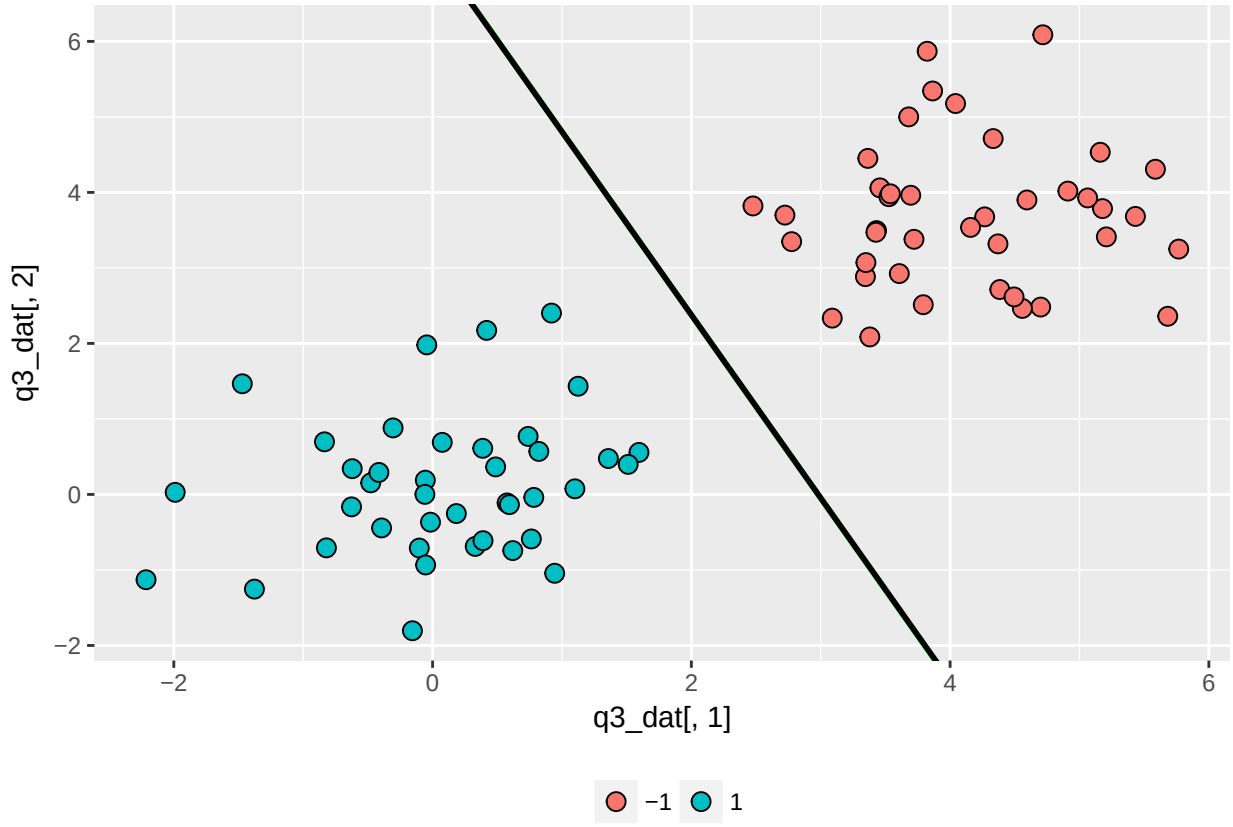
```
eps = 10e-5
Q = sapply(1:80, function(i) y[i]*t(x)[,i])
D = t(Q)%*%Q
d = matrix(1, nrow=2*n)
b0 = rbind( matrix(0, nrow=1, ncol=1) , matrix(0, nrow=2*n, ncol=1) )
A = t(rbind(matrix(y, nrow=1, ncol=2*n), diag(nrow=2*n)))
# call the QP solver:
sol = solve.QP(D +eps*diag(2*n), d, A, b0, meq=1, factorized=FALSE)
qpsol = matrix(sol$solution, nrow=2*n)
```

The solution above is the vector of α_i. To solve for the slope and intercept, we know that the dual form requires $\beta - \sum_{i=1}^{N} \alpha_i y_i x_i = 0$. According to page 420 and page 421 of the text book, for support vectors condition (12.16) must be exactly met, or $y_i(x_i^T\beta + \beta_0) = 1$. Since $y_i = 1 \; or \; -1$, we have $\beta_0 = y_i - x_i^T\beta$. The key is identifying the support vectors with $\alpha_i > 10e-5$

```
W2 = rowSums(sapply(1:80, function(i) qpsol[i]*y[i]*x[i,]))
findLine = function(a, y, X){
  nonzero =  abs(a) > 1e-5
  W = rowSums(sapply(which(nonzero), function(i) a[i]*y[i]*X[i,]))
  b = mean(sapply(which(nonzero), function(i) X[i,]%*%W- y[i]))
  slope = -W[1]/W[2]
  intercept = b/W[2]
  return(c(intercept,slope))
```

```
}
qpline = findLine( qpsol, y, x)
```

plot the separating hyperplane with the found separating hyperplane.



Problem 3b:extend the problem to nonseparable case.

Modifying Page 31 of the note yields the setup of the new optimizatio problem:

$$\underset{\alpha_i}{\operatorname{argmin}} \frac{1}{2} y_i y_j \alpha_i \alpha_j x_i^T x_j - \sum_{i=1}^{N} \alpha_i \quad s.t \ 0 \leq \alpha_i \leq C \ , \ \sum_{i=1}^{N} \alpha_i y_i = 0$$

There are n constraints with the form $\alpha_i \geq 0$, n constraints as $-\alpha \geq -C$, and one constraint as $\sum_{i=1}^{N} \alpha_i y_i = 0$ .In total there are $2n + 1$ constraints.

To make the problem be conformative with `solve.QP`, we dont have to change the specification of D and d. We just need to adjust b_0 and A for the additional n constraints $\alpha_i > -C$.

$$D = (y_i x_i)^T (y_j x_j),\ d = \mathbf{1}_n * 1, b_0 = [\mathbf{0}_n * 1\ , 0\ , -C * \mathbf{1}_n * 1], A = \begin{bmatrix} y_1 & y_2 & \dots & y_n \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ -1 & 0 & \dots & 0 \\ 0 & -1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -1 \end{bmatrix}$$

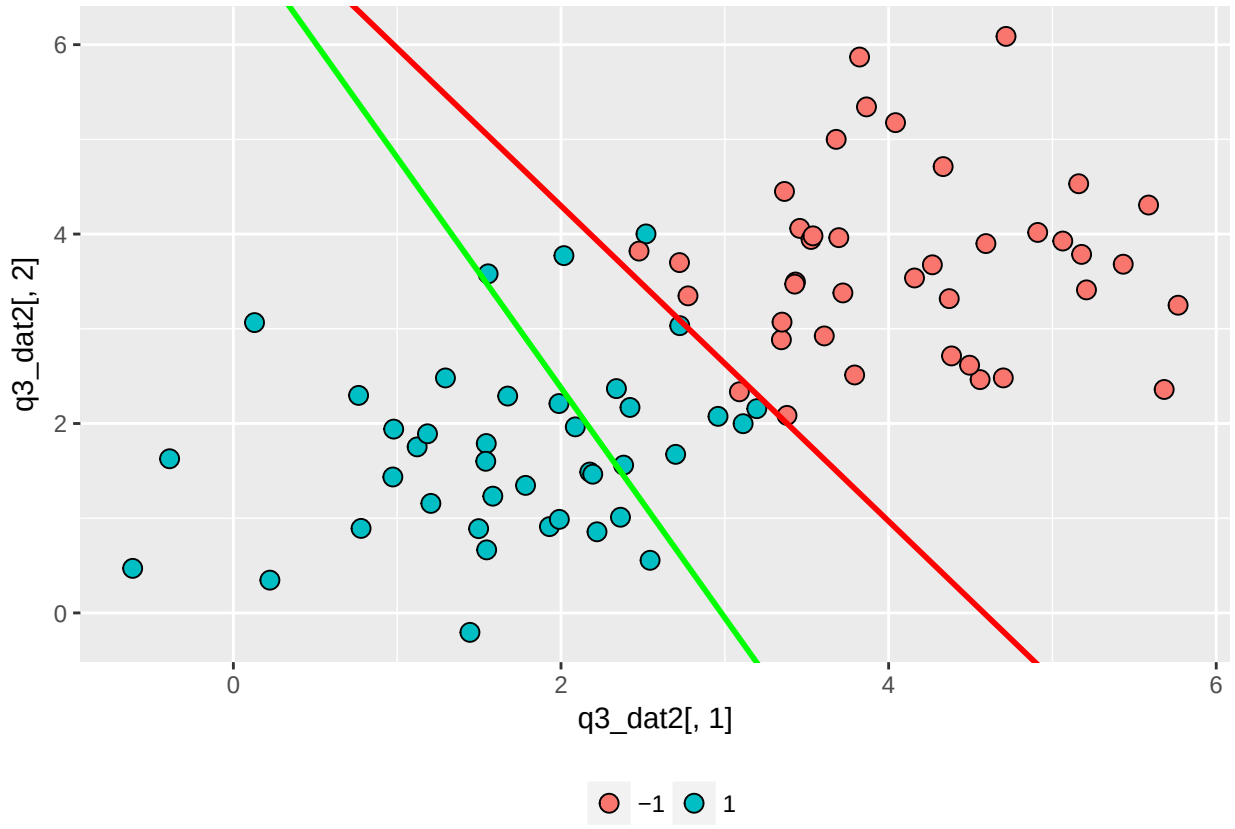
now solve the problem in `solve.QP`:

```
C= 0.25
Q = sapply(1:80, function(i) y[i]*t(x)[,i])
D = t(Q)%*%Q
d = matrix(1, nrow=2*n)
b0 = rbind( matrix(0, nrow=1, ncol=1) , matrix(0, nrow=2*n, ncol=1),
            matrix(-C, nrow=2*n, ncol=1) )
A = t(rbind(matrix(y, nrow=1, ncol=2*n), diag(nrow=2*n),-1*diag(nrow=2*n)))
# call the QP solver:
sol2 = solve.QP(D +eps*diag(2*n), d, A, b0, meq=1, factorized=FALSE)
qpsol2 = matrix(sol2$solution, nrow=2*n)
#Need to transform the solutions to coefficients of the separating hyperplane.
qpline2 = findLine( qpsol2, y, x)
```

solve the problem again using `e1071`

```
sv2 = e1071::svm(as.factor(V3)~., data = q3_dat2,kernel="linear",
                 scale=FALSE, type="C-classification", cost = 0.25)
# get the slope and intercept terms from e1071 svm:
W2 = rowSums(sapply(1:length(sv2$coefs), function(i) sv2$coefs[i]*sv2$SV[i,]))
svmline2 = c(sv$rho/W[2], -W[1]/W[2])
```

plot the separating hyperplane. The red line is the result from `solve.QP`, the green line results from `e1071::svm`



Problem 3c:derive the new decision function.

After basis expansion, if we follow every step in solving the dual form then we will reach $\beta = \sum_{i=1}^{N} \alpha_i y_i \Phi(x_i)$, where β is a $p^2 * 1$ vector. Plug this expression to the decision function $\Phi(X)^T\beta + \beta_0 = 0$ we have $\Phi(x)^T(\sum_{i=1}^{N} \alpha_i y_i \Phi(x_i)) + \beta_0 = 0$. The first term is equivalent to

$$\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j < \phi(x_i), \phi(x_j) > \ = \alpha^{\mathbf{T}}\mathbf{diag(y)K(x_i, x_j)diag(y)}\alpha$$

Therefore the new decision function is

$$\alpha^{\mathbf{T}}\mathbf{diag(y)K(x_i, x_j)diag(y)}\alpha + \beta_{\mathbf{0}} = \mathbf{0}$$

Problem 3d: Reformulate the dual form with kernel function.

$$\underset{\alpha_i}{\text{argmin}} \frac{1}{2} \sum_{i,j=1}^{N} \alpha^{\mathbf{T}}\mathbf{diag(y)K(x_i, x_j)diag(y)}\alpha - \sum_{\mathbf{i=1}}^{\mathbf{N}} \alpha_{\mathbf{i}} \quad \mathbf{s.t}\ \mathbf{0} \leq \alpha_{\mathbf{i}} \leq \mathbf{C}\ , \sum_{\mathbf{i=1}}^{\mathbf{N}} \alpha_{\mathbf{i}}\mathbf{y_i} = \mathbf{0}$$

There are n constraints with the form $\alpha_i \geq 0$, n constraints as $-\alpha \geq -C$, and one constraint as $\sum_{i=1}^{N} \alpha_i y_i = 0$ .In total there are $2n + 1$ constraints.

To make the problem be conformative with `solve.QP`, redefine the parameters as:

$$d = \mathbf{1}_n * 1\ ,\ \beta = \alpha, D = \alpha^{\mathbf{T}}\mathbf{diag(y)}\mathbf{K}(\mathbf{x_i}, \mathbf{x_j})\mathbf{diag(y)}\alpha$$

$$A = \begin{bmatrix} y_1 & y_2 & \dots & y_n \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ -1 & 0 & \dots & 0 \\ 0 & -1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -1 \end{bmatrix}$$

$$b_0 = [\mathbf{0}_n * 1\ , 0\ , -c * \mathbf{1}_n * 1]$$