STAT 542: Homework 9

Spring 2022, by Kamila Makhambetova (kamilam3)

Due: Thursday, April 14, 11:59 PM CT

Contents

Instruction

Students are encouraged to work together on homework. However, sharing, copying, or providing any part of a homework solution or code is an infraction of the University's rules on Academic Integrity. Any violation will be punished as severely as possible. Final submissions must be uploaded to compass2g. No email or hardcopy will be accepted. For **late submission policy and grading rubrics**, please refer to the course website.

- What is expected for the submission to **Gradescope**
  - You are required to submit one rendered **PDF** file `HWx_yourNetID.pdf`. For example, `HW01_rqzhu.pdf`. Please note that this must be a `.pdf` file generated by a `.Rmd` file. `.html` format cannot be accepted.
  - Please follow the instructions on Gradescope to select corresponding PDF pages for each question.
- Please note that your homework file is a **PDF** report instead of a messy collection of R codes. This report should **include**:
  - Your Name and NetID. (Replace `Ruoqing Zhu (rqzhu)` by your name and NetID if you are using this template).
  - Make all of your `R` code chunks visible for grading.
  - Relevant outputs from your `R` code chunks that support your answers.
  - Provide clear answers or conclusions for each question. For example, you could start with `Answer: I fit SVM with the following choice of tuning parameters ...`
  - Many assignments require your own implementation of algorithms. **Basic comments are strongly encouraged** to explain the logic to our graders. However, line-by-line code comments are unnecessary.
- Requirements regarding the `.Rmd` file.
  - You do **NOT** need to submit `Rmd` files. However, your PDF file should be rendered directly from it.
  - Make sure that you **set random seeds** for simulation or randomized algorithms so that the results are reproducible. If a specific seed number is not provided in the homework, you can consider using your NetID.
  - For some questions, there will be restrictions on what packages/functions you can use. Please read the requirements carefully. As long as the question does not specify such restrictions, you can use anything.

About HW9

In this HW, we use two examples to demonstrate how the RKHS and Representer Theorem are used in practice. In both questions, we will use the radial basis kernel, defined as

$$K(\mathbf{x}, \mathbf{z}) = e^{-\frac{\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}}.$$

You are not required to tune the parameter σ. The information will be given.

Question 1 [45 Points] Kernel Ridge Regression

Let's first generate a set of data using the following code (or some similar code in Python):

```
set.seed(1)
n = 500
p = 2
X = matrix(rnorm(n*p, 0, 1), n, p)
y = 2*sin(X[, 1]*2) + 2*atan(X[, 2]*4) + rnorm(n, 0, 0.5)

alldata = cbind(X, y)
train = alldata[1:200, ]
test = alldata[201:500, ]
```

[5 Points] As a comparison, we can first fit a ridge regression to this data. Use the `train` part to fit a ridge regression using the `glmnet()` package with 10-fold cross-validation. Use `lambda.min` as your tuning parameter to predict the testing data. What is the prediction error?

```
library(glmnet)

ridge.fit = cv.glmnet(x = train[, 1:2], y = train[, 3])

mse=mean((predict(ridge.fit, test[, 1:2], s = "lambda.min") - test[, 3])^2)
print(paste0("prediction error= ", mse))
```

```
## [1] "prediction error= 3.22660313017677"
```

To fit a kernel ridge regression, we use the following formulation:

$$\underset{\boldsymbol{\alpha}}{\text{minimize}} \quad \frac{1}{n}\|\mathbf{y} - \mathbf{K}\boldsymbol{\alpha}\|^2 + \lambda \boldsymbol{\alpha}^{\mathrm{T}}\mathbf{K}\alpha$$

It should be noted that you could add an α_0 term to this regression to rigorously add the intercept term. However, the theoretical mean of Y is zero. Hence, it is not a crucial issue, and not required here. Following our derivation in the class, perform the following to complete this kernel ridge regression:

- [10 Points] Compute the 200×200 kernel matrix $\mathbf{K}$. Use a Gaussian kernel and $\sigma = 1$.

```
#gaussian/ radial basis kernel function

gaus_kernel<-function(sigma, x_test, x_train){
  power=(t(x_test-x_train)%*%(x_test-x_train))/(2*(sigma^2))
  exp(-power)
}

#K matrix for train dataset
sigma=1
K=matrix(0,nrow(train), nrow(train))
```

```
for(i in 1:nrow(train)){
  for(j in 1:nrow(train)){
    K[i,j]=gaus_kernel(sigma, train[i, 1:2], train[j, 1:2])

}
}
```

- [10 Points] Based on the kernel ridge regression derivation, get the solution $\boldsymbol{\alpha}$. Use $\lambda = 0.01$

```
#Calculate alpha vector
#alpha=(K+n*lambda*I)^-1 *y

lambda = 0.01
alpha=solve(K+nrow(train)*lambda*diag(nrow(train)))%*%train[,3]
```

- [10 Points] The prediction of kernel ridge regression involves computing the kernel between a testing data and all the training data. To be specific, if we have a new testing data z, then the prediction $\widehat{f}(z)$ is

$$\widehat{f}(z) = \sum_{i=1}^{n} \alpha_i K(z, x_i)$$

  Since we have 300 testing data, this involves calculating a 300×200 testing data kernel matrix, denoted as K_{test}, and the vector of prediction is $K_{\text{test}}\boldsymbol{\alpha}$.

```
#function to calculate y_hat

preidct_funct<-function(x_test,x_train, alpha){
  K_test=matrix(0,nrow(x_test), nrow(x_train))

  for(i in 1:nrow(x_test)){
    for(j in 1:nrow(x_train)){
      K_test[i,j]=gaus_kernel(sigma, x_test[i,], x_train[j, ])
    }
  }

  y_pred=rep(0,nrow(x_test))

  for(i in 1:nrow(x_test)){
    alpha_K=alpha* K_test[i,]
    y_pred[i]=sum(alpha_K)
  }

  y_pred
}
```

- [10 Points] Predict the testing data using the kernel and $\widehat{\boldsymbol{\alpha}}$. What is the prediction error? Is it better than the ridge regression?

```
#calculate y_hat for test dataset

y_pred_test=preidct_funct(test[,1:2],train[,1:2], alpha)

#calculate test prediction error

mse2=mean( (y_pred_test- test[, 3])^2)
print(paste0("prediction error= ", mse2))
```

```
## [1] "prediction error= 0.799552391394581"
```

Comment

Kernel Ridge Regression is better than Ridge regression, as $MSE_{KRR} = 0.8$ and it is 4 times less than $MSE_{RR} = 3.2$.

RR is Ridge Regression, KRR is Kernel Ridge Regression.

Question 2 [55 Points] Non-linear SVM as Penalized Version

Recall that in HW8, we solved SVM using a penalized loss framework, with the logistic loss function:

$$L(y, f(x)) = \log(1 + e^{-yf(x)}).$$

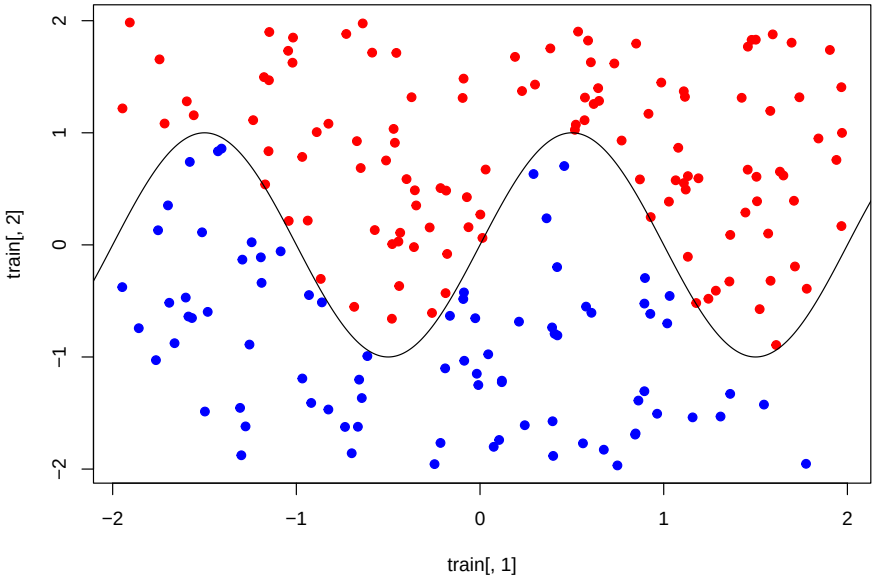
Again, we can specify the form of $f(\cdot)$ as in a RKHS. And the penalized objective function becomes

$$\frac{1}{n}\sum_{i=1}^{n} L(y_i, K_i^{\mathrm{T}}\boldsymbol{\alpha}) + \lambda\alpha^{\mathrm{T}}\mathbf{K}\alpha$$

where $\mathbf{K}$ is the same $n \times n$ kernel matrix as before and K_i is the ith column of $\mathbf{K}$. The data are generated by the following code.

```r
set.seed(1)
n = 500
p = 2
X = matrix(runif(n*p, -2, 2), n, p)
y = sign( X[, 2] - sin(pi*X[, 1]))
alldata = cbind(X, y)
train = alldata[1:200, ]
test = alldata[201:500, ]

# visualize the data
plot(train[, 1], train[, 2], col = ifelse(y > 0, "red", "blue"),
     pch = 19, )
lines(seq(-3, 3, 0.01), sin(pi*seq(-3, 3, 0.01)) )
```



Similar to the HW8 linear SVM penalized version, we perform the following steps to complete this model fitting:

- [15 Points] Write a penalized loss objective function `SVMfn(b, K, y, lambda)` corresponding to the form we specified. Make sure to use the $1/n$ scale in the loss function part.

```
#loss function

loss<-function(y,f){
  log(1+exp(-y*f))
}

# SVM/objective function

SVMfn<-function(alpha,K, y, lambda){

  sum=0

  for( i in 1:length(y)){
    sum=sum+loss(y[i],t(K[,i]%*%alpha))
  }

  obj_f=sum/length(y)+lambda*t(alpha)%*%K%*%alpha

  obj_f
}


#K matrix for train dataset

sigma=0.2
K2=matrix(0,nrow(train), nrow(train))

for(i in 1:nrow(train)){
  for(j in 1:nrow(train)){
    K2[i,j]=gaus_kernel(sigma, train[i, 1:2], train[j, 1:2])

}
}
```

- [20 Points] Derive the gradient of the loss function, typeset with LaTex. Then write a gradient function `SVMgr(b, K, y, lambda)` to implement it.

Derivation

$$L(y, f(x)) = \log(1 + e^{-yf(x)})$$

$$h(\alpha) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, K_i^{\mathrm{T}}\boldsymbol{\alpha}) + \lambda\alpha^{\mathrm{T}}\mathbf{K}\alpha$$

$$h(\alpha) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + e^{-y_i K_i^{\mathrm{T}}\alpha}) + \lambda\alpha^{\mathrm{T}}\mathbf{K}\alpha$$

Using chain rule for derivatives I calculated $\frac{\partial h}{\partial \alpha}$

$$\frac{\partial h}{\partial \alpha} = \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i K_i e^{-y_i K_i^T \alpha}}{1+e^{-y_i K_i^T \alpha}} + \lambda(K\alpha + K^T\alpha)$$

```
#gradient function

SVMgr<-function(alpha,K, y, lambda){
  sum=0

  for(i in 1:length(y)){
    y_K_a=-y[i]*t(K[,i])%*%alpha
    sum=sum+((-y[i]*K[,i])*exp(y_K_a))/(1+exp(y_K_a))
  }

  grad=sum/length(y)+lambda*(K%*%alpha+t(K)%*%alpha)
  grad

}
```

- [20 Points] Use the `optim()` function to solve this optimization problem by supplying the objective function and gradient function. Use $\sigma = 0.2$ and $\lambda = 0.01$. Initialize your coefficients as zeros. Report the following:
  - The optimal loss function
  - A confusion table of the training data
  - Mis-classification rate on training data.
- It could be difficult to obtain the decision line itself. However, its relatively easy to obtain the fitted label for the testing data. Hence, calculate the fitted label of the testing data and report the classification error. Plot the testing data using the fitted labels as colors. You should also add the true decision line (since you already know the true data generator). This would allow you to visualize (approximately) the decision line. You do not need to plot the original labels.

Note: 1) In a real problem, you can perform cross-validation to find the best tuning parameters. 2) You may also add a constant term α_0 to the problem to take care of intercept. These two are not required for the HW.

```
#find optimal alpha vector

alpha=rep(0,length(train[, 3]))
alpha_optimal=optim(alpha, SVMfn, SVMgr, method = "BFGS", K = K2, y = train[, 3] , lambda=lambda)$par


#calculate optimal loss

optimal_loss=SVMfn(alpha_optimal,K2, train[, 3], lambda) -lambda*t(alpha_optimal)%*%K2%*%alpha_optimal
print(paste0("optimal loss function= ", optimal_loss))
```

```
## [1] "optimal loss function= 0.532137284564035"
```

```
#calculate y_hat for classification problem using train dataset

y_hat_num=K2%*%alpha_optimal
y_hat_train=y_hat_num

for(i in 1:length(y_hat_num)){
  if(y_hat_num[i]>=0){
    y_hat_train[i]=1
  }else{
```

```
    y_hat_train[i]=-1
  }
}



#calculate Confusion matrix for train

y_train=train[,3]
conf_table<-table(y_train, y_hat_train)
print("Confusion matrix: ")
```

```
## [1] "Confusion matrix: "
```

```
conf_table
```

```
##        y_hat_train
## y_train  -1   1
##      -1  82   0
##      1    2 116
```

```
#calculate misclassification rate for train

misclass_rate=as.numeric(table(y_hat_train==train[,3])[1])/length(train[,3])
print(paste0("misclassification rate= ",misclass_rate))
```

```
## [1] "misclassification rate= 0.01"
```

```
#calculate y_hat_test for classification problem using test and train datasets
y_hat_num2=preidct_funct(test[,1:2],train[,1:2], alpha_optimal)
y_pred_test2=y_hat_num2


for(i in 1:length(y_hat_num2)){
  if(y_hat_num2[i]>=0){
    y_pred_test2[i]=1
  }else{
     y_pred_test2[i]=-1
  }
}



#calculate Confusion matrix

y_test=test[,3]
conf_table<-table(y_test, y_pred_test2)
print("Confusion matrix: ")
```

```
## [1] "Confusion matrix: "
```

```
conf_table
```

```
##       y_pred_test2
## y_test  -1   1
##     -1 132  21
##     1    5 142
```

```r
#calculate misclassification rate for test

misclass_rate2=as.numeric(table(y_pred_test2==test[,3])[1])/length(test[,3])
print(paste0("misclassification rate= ",misclass_rate2))
```

```
## [1] "misclassification rate= 0.0866666666666667"
```

```r
# visualize the data for test dataset
plot(test[, 1], test[, 2], col = ifelse(y_pred_test2 > 0, "red", "blue"),
       pch = 19, xlab = "x1", ylab = "x2" )
  lines(seq(-3, 3, 0.01), sin(pi*seq(-3, 3, 0.01)), col="black", lwd=2 )

#legend
op <- par(cex = 1.1)
    legend(-2.2,-1, c("Positive","Negative","decision line"), col=c("red", "blue", "black"),
         pch=c(19, 19, NA), lty=c(NA,NA,1),text.col=c("red", "blue", "black"))
```