Project 3

Filip Axelsson

2021-12-15

Task 1

(A)

Consider the two follwing equations:

$$\min_{\beta_0,\beta} \sum_{i=1}^{N} [1 - y_i f(x_i)]_+ + \frac{\lambda}{2} ||\beta||^2 \qquad (1)$$

where $f(x) = h(x)^T\beta + \beta_0$

$$\min_{\beta_0,\beta} \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N} \xi_i, \quad \text{subject to } \xi > 0, y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i \ \forall i \quad (2)$$

Note that (1) and (2) is the same when $\lambda = \frac{1}{C}$.
<u>Solution</u>:
Equation (1) can be rewritten as following when using $\lambda = \frac{1}{C}$

$$\min_{\beta_0,\beta} \sum_{i=1}^{N} [1 - y_i f(x_i)]_+ + \frac{1}{2C}||\beta||^2 \Leftrightarrow \min_{\beta_0,\beta} C\sum_{i=1}^{N} [1 - y_i f(x_i)]_+ + \frac{1}{2}||\beta||^2$$

We also know that $f(x) = h(x)^T\beta + \beta_0$ and now again rewrite the equation

$$\min_{\beta_0,\beta} C\sum_{i=1}^{N} [1 - y_i h(x)^T\beta + \beta_0]_+ + \frac{1}{2}||\beta||^2 \quad (3)$$

If we now focus on equation (2), we know that $y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i$ which aslo can be expressed as $\xi_{\leq} 1 - y_i(x_i^T\beta + \beta_0)$ and by putting in the result in the equation we get

$$\min_{\beta_0,\beta} \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N} [1 - y_i(x_i^T\beta + \beta_0)] \quad (4)$$

Since both $h(x)$ in equation (3) and x_i in equation 4 are function, leads to that both equation (1) and (2) are the same. ■

(B)

The binomial deviance has the following loss function $L[y, f(x)] = \log[1 + e^{-yf(x)}]$, we now want to derive the minimizing function which is given by $f(x) = \log(\frac{P(Y=-1|x)}{P(Y=1|x)})$. We start with setting up the problem, which is:

$$f^*(x) = \min_{f(x)} \mathbb{E}_{Y|x}[\log(1 + e^{-Yf(x)})]$$

If we now derivate this expression we get

$$\frac{\partial}{\partial f}\mathbb{E}_{Y|x}[\log(1+e^{-Yf(x)})] = \mathbb{E}_{Y|x}[\frac{-Y}{e^{Yf(x)}+1}] = 0$$

If we now evaluating this result when our targets are $Y = \pm 1$ gives us the following:

$$\frac{-(-1)}{e^{-f(x)}+1}P(Y=-1|x) - \frac{1}{e^{f(x)}+1}P(Y=+1|x) = 0 \Leftrightarrow \frac{e^{f(x)}+1}{e^{-f(x)}+1}P(Y=-1|x) - P(Y=+1|x) = 0$$

Which if we simplify and make $e^{f(x)}$ alone we get:

$$e^{f(x)} = \frac{P(Y=+1|x)}{P(Y=-1|x)}$$

If we now solve for $f(x)$ by taking the logarithm we get what we wanted to derive:

$$f(x) = \log(\frac{P(Y=-1|x)}{P(Y=+1|x)})$$

We have now derived the minimizing function for the binomial deviance. ■

Next problem is to derive the minimizing function for the SVM Hinge Loss, which is given by $f(x) = \text{sign}[P(Y=+1|x)+\frac{1}{2}]$. We start with setting up the problem, note the loss function is $[1-yf(x)]_+$.

$$f^*(x) = \min_{f(x)} \mathbb{E}_{Y|x}[[1-yf(x)]_+]$$

The derivative of this expression is then

Next problem is to derive the minimizing function for the SVM Hinge Loss, which is given by $f(x) = \text{sign}[P(Y=+1|x)+\frac{1}{2}]$. We start with setting up the problem, note the loss function is $[1-yf(x)]_+$.

$$f^*(x) = \min_{f(x)} \mathbb{E}_{Y|x}[[1-yf(x)]_+]$$

We now want to find $f^*(x)$ given x, that is:

$$\mathbb{E}_{Y|x}[[1-yf(x)]_+|x] = [1-f(x)]_+P(Y=+1|x) + [1+f(x)]_+(1-P(Y=+1|x))$$

Note that $f(x) \in [-1,1]$, which when $-1 \leq f(x) \leq 1$ gives:

$$\mathbb{E}_{Y|x}[[1-yf(x)]_+|x] = [1-f(x)]P(Y=+1|x) + [1+f(x)](1-P(Y=+1|x)) = f(x)(1-2P(Y=+1|x))+1$$

This leads to the following:

$$f(x) = \left\{ \begin{array}{cl} 1 & \text{if } P(Y=+1|x) \geq \frac{1}{2} \\ -1 & \text{if } P(Y=+1|x) < \frac{1}{2} \end{array} \right. = \text{sign}[P(Y=+1|x) - \frac{1}{2}]$$

Note that the hinge loss estimates the classifer $G(x)$ itself. ■

(C)

If we consider the Gaussian kernel $K(x,x') = exp(\gamma||x-x'||^2)$ we see that we a paramter γ, a natural question is how does this paramter affect the the bias-variance tradeoff and how should we choose it. Since γ defines how much influence a signel training example has, leads to that a larger γ will result in that the closer other examples must be to be affected. That is, with a high value of γ the decision boundry will only depend on the points that are the clossest to de decision boundry and ignoring the points that are farther away. That is the boundy will be be more jagged (flexed), which result to a lower bias but a higher variance. If we now consider a lower value of γ which leads to the decision boundries will take into consider the point that are farther will make the boundries less jagged, that is more linear. This leads to that a lower value of γ tends to give higher bias and lower variance. In order to choice the most optimal value of γ one should preforme cross-validation.

Task 2

(A)

In this project we are going to use the package “caret” for its function “train()”, the package “kernlab” is also needed since it will give the “train()”-function the SVM method. This functions gives the possibility to train a SVM, below in *Table 1* we can see the differnt parametrs in the function. Before we start fitting the data to the SVM, we need to decide which kernel the fucntion is going to use. We have a choice between polynomial kernel, which has the tuning parameters degree, scale and cost, and the other one is the radial basis function kernel which has the tuning parameters sigma and cost. Usually the polynomial kernels are less time consuming and provides less accuracy than the radial basis function kernel, however it’s easier to interpret. What I have read no one has shown that one kernel always will preforme better than another. What we can take into consider is the amount of tuning parameters, so therefore I will use the radial basis function kernel since it only have two tuning parameters, and has usually better preformence. The downside will be that it will probably be more time consuming than the polynomial kernel. Note that sigma are equivalent to gamma and most be greater than 0.

Table 1: List over parameters in the function train()

Parameter	Information
Method	Specifying which classification or regression model to use.
trControl	A list of values that define how this function acts
preProcess	A string vector that defines a pre-processing of the predictor data, eg “scale”, and “center”
tuneGrid	A data frame with possible tuning values.

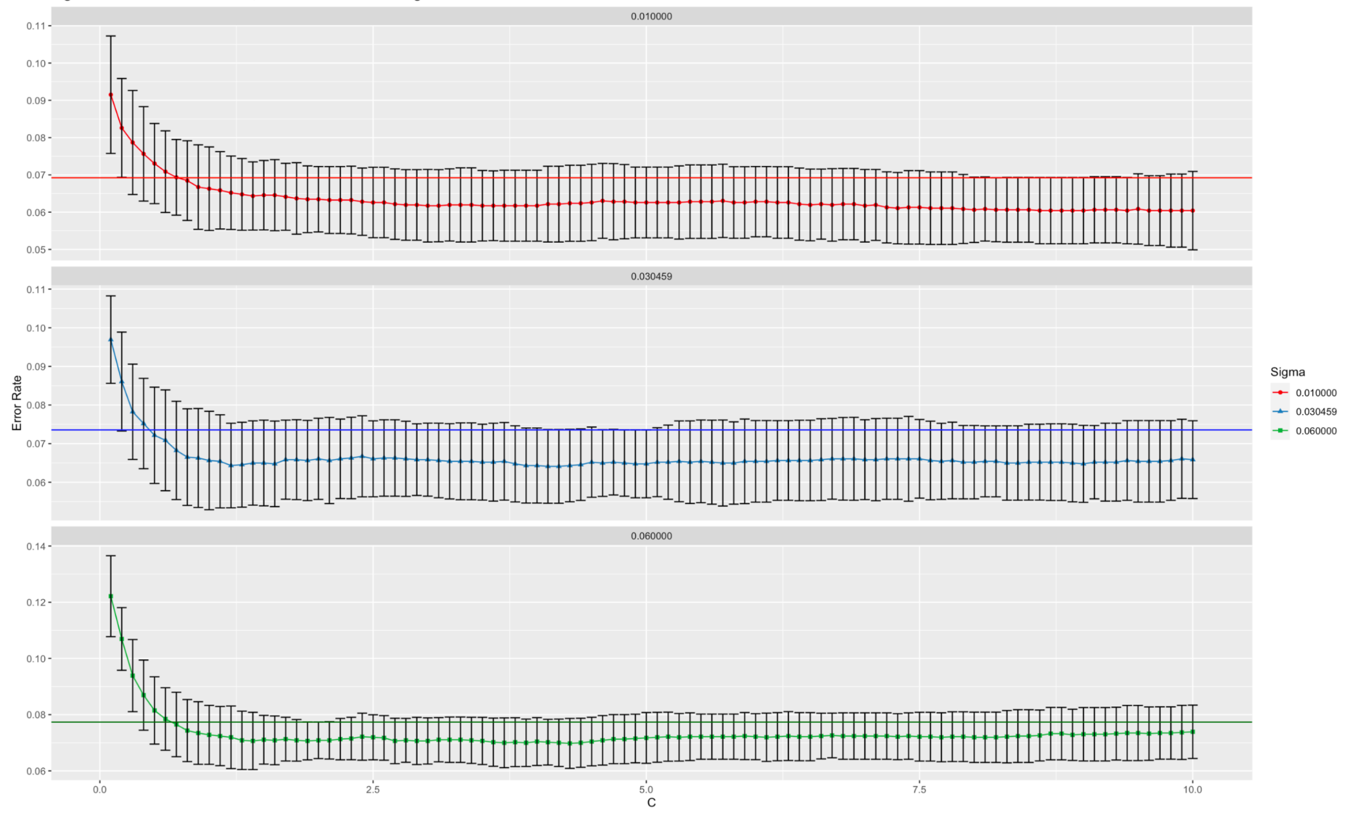
(B)

I have created the function “cv_svm(cv_folds,c,s)” where you can specify the number of folds in the cross-validation by the argument “cv_folds”. You can by the argument “s” specify the tuning parameter sigma and by the argument “c” you can specify the tuning parameter Cost. The function is using the radial basis kernel trick. The function can be viewed in the section “Code”.

(C)

In *Figure 1* we can see the result of the cross-valdition error against the cost C, for three different sigma/gamma 0.01, 0.030459 and 0.5. Note, i did preforme cross-valdation on the tuning parameter sigma, and got that the best result was 0.030459. However this took alot of time and therefore i only show 3 differnt values of sigma, one that is relativ small and one that is relative large compared with the optimal value.

Figure 1: CV-error versus Cost, for different values of sigma



(D)

I think, as many others that with the "right" kernel selection SVM can preforme very well, but it is worth to note that without a kernel it's just a linear model. A kernel, in this case we used the radial basis function, can work well for some problems and for some problem it doesn't. If we now look on tree based approaches such as gradient boosting, which clases as robust can work on a wide variety of problems. It can capture dependencies in ways that a linear model can't. Boosting often improve the preformance, and since we using a relative big dataset it can easily cause overfittning. Therefore this could be a reason why the gradient boosting trees preformed better than SVM with radis basis function as kernel. However it doesn't differ much between the both methods cv-error rates, at least looking from the one-standrard error perspective. One big difference I noticed was the time to train the models, the time it took to train a SVM was significantly longer than for gradient boosting trees. Conclusion, the gradient boosting trees preformed better than SVM.

Code

```r
# reading in the package
library(tidyverse)
library(caret)
library(kernlab)

# reading in the data
spam_data <- read_table2("../Statistisk inlärning/spam.data.txt", col_names = F)
```

```r
cv_svm <- function(cv_folds,c,s){
```

```r
# Decide how we should train the model, that is with cv.
train_control <- trainControl(method = "cv", number = cv_folds)

# Train the model
svm <- train(factor(X58)~.,  # specifying which model, eg which variable is the respone
             data = spam_data, # specifying the data
             method = "svmRadial",  # specifying the method, we are using the radial basis kernel
             trControl = train_control, # Applying cv
             preProcess = c("center","scale"),
             tuneGrid = expand.grid(C = c, sigma = s)) # Specifying how the tuning parameters
return(svm)
}

# Creating a cv with 10 folds, tuning parameter C fromg 0.1 to 10
# tuning for sigmas 0.01, 0.030459 (optimal) and 0.06
svm_sigma_c <- cv_svm(10,seq(0.1,10,0.1),c(0.01,0.030459,0.06))
```

```r
# Creating a function that can plot the cv with the standarderror bars

plot_cv_error <- function(cv_file){
  # convert the Accuracy to Error
  cv_file[["results"]][["Accuracy"]] <- (1-cv_file[["results"]][["Accuracy"]])
  cv_file %>%
    ggplot() +
    geom_errorbar(
    aes(ymin =cv_file[["results"]][["Accuracy"]]-cv_file[["results"]][["AccuracySD"]],
        ymax =cv_file[["results"]][["Accuracy"]]+cv_file[["results"]][["AccuracySD"]])) +
    scale_y_continuous("Error Rate") +
    scale_x_continuous("C") +
    facet_wrap(.~sigma, ncol = 1, scale = "free_y") # split the graphs to seprat sigmas.
}




# Taking out the value for the one-standard error

one_sd_sigma_0_01 =min(
  (1-svm_sigma_c[["results"]][["Accuracy"]][1:100])+svm_sigma_c[["results"]][["AccuracySD"]][1:100])
one_sd_sigma_0_03 =min(
  (1-svm_sigma_c[["results"]][["Accuracy"]][101:200])+svm_sigma_c[["results"]][["AccuracySD"]][101:200]
one_sd_sigma_0_06 =min(
  (1-svm_sigma_c[["results"]][["Accuracy"]][201:300])+svm_sigma_c[["results"]][["AccuracySD"]][201:300]

# making it into a tibble so we can plot them later
one_sd <- tibble(X=c(0.01,0.030459,0.06), Z= c(one_sd_sigma_0_01,one_sd_sigma_0_03,one_sd_sigma_0_06))


 # Finding a color that is transparent
 # a workaround since i coulndt find a good way to make
 # horizontal lines for each sigma when using face_wrap
 library(Rgb)
 # rgb(0, 0, 0, alpha = 0)
# Plottning the CV error vs Cost C
 plot_cv_error(svm_sigma_c) +
   scale_colour_brewer(palette ="Set1", name = "Sigma") + # rearrange the order of the colours
   labs(title = ("Figure 1: CV-error versus Cost, for different values of sigma")) +
   geom_hline(data = one_sd ,aes(yintercept = Z), col = c("red","#00000000","#00000000","#00000000",
                                                          "blue","#00000000",
                                                          "#00000000","#00000000","darkgreen"))
```