Exercise no 9

Corentin Lacroix 1812554

MTH 6312: Méthodes Statistiques D'Apprentissage

November 11, 2015

Exercise 1 Separation problem is a well-known convergence issue in logistic regression and the support vector machines. However, it is considered a curse in regression and a blessing in support vector machines (try to understand why?).

Suppose a binary regression with only one x. Consider 10 data with $y_i = 0$ and $x_i = 0$, and 10 data with $y_i = 1$ and $x_i = 1$, making a dataset with $n = 20$ sample size. Fit the logistic regression model with $\beta_0 = 0$ using the `glm` R function.

Draw the likelihood function for this data set, i.e. draw $\ell(\beta_1)$ (y-axis) versus β_1 (x-axis).

Solution 1 Given a particular data set, the fact that the logistic regression separation problem doesn't converge can indicate the optimization problem don't have existing, finite and unique solutions. Thus, parameters estimations given by the numerical methods can't be relevant and variances of parameters estimation will be high (result of the optimization problem will be strongly influenced by the data).

For SVM, my intuition is because of the lack of convergence in the optimization problem, initial values/parameters of numerical resolution algorithms (no link with the data) will strongly influence parameters estimations. Thus, as there is no real possible Cross Validation in SVM, one can average many boundary decisions constructed with the same algorithm (SVM) taking as inputs a large set of random values.

For this question, I created a R likelihood function corresponding to the log likelihood :

```
> likelihood <- function(X, Y, beta){
```

```
+   return(sum(Y * (beta * X) - log(1 + exp(beta * X))))
+}
> beta_rng <- seq(-20, 20, 0.1)
> context_likelihood <- function(beta){
+   return(likelihood(X, Y, beta))
+}
> plot(beta_rng, lapply(beta_rng, context_likelihood), type='l', xlab="Beta", ylab="
```

We can note that the log likelihood gets to its maximum value (0) when $\beta \to +\infty$ because the likelihood function is strictly increasing. Thus, the optimization linked to logistic regression doesn't admit a finite solution.

Exercise 2 Try to fit the logistic model by direct maximization of the log likelihood using the `optim` or the `nlm` function. Does it converge? Why?

Solution 2 By the following command lines, we can find the maximum of the log likelihood and its corresponding argument (having previously added a negative sign in front of the returned value of context-likelihood function) :

```
> optim(0, context_likelihood)
> optim(0, context_likelihood, lower = -10.0, upper = 40.0)
```

The optimization method is not the same when we specify the interval in which we are looking for a solution or not and it gives here different results. Moreover, in both cases, algorithm seems to converge (0 convergence flag returned).

The first method gives $\beta = 38.3$ and the second one $\beta = 20.7041$. Consequently, the second result is close to the estimation found direclty with the R glm routine (19.7). Moreover, in the second case, the message "CONVERGENCE: REL_REDUCTION_OF_F ¡= FACTR*EPSMCH" is returned by the optimizer. This message means that convergence has been reached by the algorithm because updates in objective functions got below the algorithm threshold for convergence.

Exercise 3 Modify your likelihood function to resolve this issue, and optimize your new function to estimate β_1.

Solution 3 I have modified the log likelihood function by adding a L1 norm constraint on β :

```
> reg_likelihood <- function(X, Y, beta, lambda){
+    return(sum(Y * (beta * X) - log(1 + exp(beta * X))) - lambda * abs(beta))
+}
> context_reg_likelihood <- function(beta){
+    return(-reg_likelihood(X, Y, beta, 4.5))
+}
```

The (log) likelihood function now gets to its maximum values for a unique and finite β. This value depends on the value of lambda. Now, both optimization methods give approximatively the same results. For values of lambda ≥ 5, the second optimization method returns a valid result but ends with an error. For λ close to 5, we get a β close to zero

whereas for λ close to 0 we get approximatively the same results than without constraints.

Here, this parametric optimization problem is the same than before (maximization of the likelihood) but with one more constraint on the absolute value of lambda. As the likelihood function was strictly increasing, the solution of this modified optimization problem is the specified boundary on beta in the other formulation (boundary dual with the lambda in our formulation).

Exercise 4 Remember $D(\beta_0, \boldsymbol{\beta})$ from ESL page 131.

Show that minimizing

$$D(\beta_0, \boldsymbol{\beta}) = -\sum_{i \in \mathcal{M}} y_i(\beta_0 + \mathbf{x}_i^\top \boldsymbol{\beta})$$

is equivalent to minimizing

$$S(\beta_0, \boldsymbol{\beta}) = \sum_{i=1}^{n} \{1 - y_i(\beta_0 + \mathbf{x}_i^\top \boldsymbol{\beta})\}_+.$$

Solution 4 Here, I think we have to suppose that data are separable (or your hint don't make any sense to me). If we suppose that, then we have

$$\exists(\hat{\boldsymbol{\beta}}, \beta_0) \; that \; satisfies \; : \; \forall i \in 1, \cdots, n \; y_i(\hat{\boldsymbol{\beta}} x_i + \beta_0) > 0$$

And the solution of the previous optimization problem must satisfy this constraint and thus, $D(\beta_0, \hat{\boldsymbol{\beta}}) = 0$ and $\mathcal{M} = \emptyset$.

Let $(\beta_0, \boldsymbol{\beta})$ be a solution of the previous optimization problem. Then the property I've just talked about is true. Consequently, we can affirm that $\underset{i \in 1, \cdots, n}{min} (y_i(\boldsymbol{\beta} x_i + \beta_0))$ exists and

is reached.

We have : $\forall \delta \in \left]0, \min_{i \in 1,\cdots,n} (y_i(\boldsymbol{\beta} x_i + \beta_0))\right[, \forall i \in 1, \cdots, n \ : \delta - y_i(\boldsymbol{\beta} x_i + \beta_0) < 0$ and consequently $\forall \delta \in \left]0, \min_{i \in 1,\cdots,n} (y_i(\boldsymbol{\beta} x_i + \beta_0))\right[, S_\delta(\beta_0, \beta) = \sum_{i=1}^{n} (\delta - y_i(\beta_0 + \beta x_i)_+) = 0$

But we have also $S_\delta \geq$ as a sum of positive terms. Consequently,

$$(\beta_0, \boldsymbol{\beta}) \textit{ is solution of the } S_\delta \textit{ optimization problem}$$

We could prove the inverse implication in a very similar way to prove the equivalence between these 2 optimization problems.

Here is where I am totally lost... Here we have in S a term δ whose range depend on β. We could "fix" δ to have a special relative place in its range. It gives us an infinity of optimization problems equivalent to our first optimization problem. But how can we force δ to be a specific value not depending on β ??

Secondly, I don't get the equivalence between the two optimizations problem.. In the first one, I clearly see why a linear boundary with normal vector β separates the separable data (signed distance of an input vector to the hyperplane * its label value must be positive for all data points, it is not SVM ?!). But in the second optimization problem, it seems that the solution is an optimal hyperplane that forces the margin between the hyperplane and the closest points being ≥ 1.

Exercise 5 For the zip code example, plot data on the first two linear discriminant axes, you may need to look at functions in `R>library("class")`

Solution 5 For this first plot, we need first to fit lda to our data. With R plot() function, we can direclty represent train data on the l first LDA directions found

```
#import zip dataset (first 500 first rows of zip test data)
> data(zip)
> zipdf <- data.frame(zip.test)[0:500,]


#fit lda with R lda function
> ldafit <- lda(X1 ~., data=zipdf)
> plot(ldafit, dimen=2)
```

Exercise 6 On the plot 5), if possible, show the linear discriminant bounaries (or regions).

Solution 6 I have spent nearly 12 hours on these questions. An it seems that all my attempts have failed... My goal in this question and the 2 followings was to predict labels for a very important number of points. This grid of points would cover the entire space spanned by the 2 most important Linear Discriminant algorithm LD1 and LD2.

Here is the code I used for this :

```
> data(zip)
> zipdf2 <- data.frame(zip.test)[0:500,]


X <- zipdf2[,-1] #predictors
Y <- zipdf2[,1] #labels
```

```
K <- length(unique(Y)) #nber of classes

N <- length(Y) #nber of data points

p <- ncol(X) #nber of features


#matrix containing class means points

mu.k <- do.call("rbind", lapply(1:K, function(k) colMeans(X[Y == k-1,])))

mu.bar <- colMeans(mu.k)


#prediction on train data

X.pred <- predict(ldafit, newdata = X)


#class centroids prediction

mu.k.pred <- predict(ldafit, newdata = as.data.frame(ldafit$means))


X.new <- X.pred$x #values of X in (LD1, LD2,..., LD9) space

mu.k.new <- mu.k.pred$x #idem for classes centroids


#constructionof the grid

x.lim <- range(X.new[, 1]) #limit of our future grid

y.lim <- range(X.new[, 2])

x.grid <- 100

y.grid <- 100

grid <- cbind(seq(x.lim[1], x.lim[2], length = x.grid),
```

```
rep(seq(y.lim[1], y.lim[2], length = y.grid), each = x.grid))


#we extend our grid to full (LD1, ..., LD9) coordinates

extendedGrid <- cbind2(grid, matrix(0,10000, 7))

#we transform the grid vectors into features space vectors

extendedGrid.basecoordinates <- extendedGrid %*% t(ldafit$scaling)


#make prediction for our grid points

grid.ldaClasses = predict(ldafit,

data.frame(extendedGrid.basecoordinates))$class


#plot data points prediction

plot(X.new[, 1], X.new[, 2], col = color[Y+1], type = "n",

main = "LDA classification regions", xlab = "LD1", ylab = "LD2")

points(grid[, 1], grid[, 2],

col = color[grid.ldaClasses + 1], pch = 19, cex = 0.3)

points(X.new[, 1], X.new[, 2], pch = Y + 1, cex = 0.8)

points(mu.k.new[, 1], mu.k.new[, 2], pch = 19, cex = 1.5)
```

At a point, when we have our 10000 * 2 grid, we need to extend this grid for its point to have full coordinates in (LD1, ... LD9) space. I thought that the sacling matrix could transform back grid points vectors in LDA space back to features space but it is wrong and I think it is the inverse.

I then tried not to use the R predict() funcion for predicting grid classes but directly by applying the rule $G = argmax(\delta_k(x))$ in the transformed space (as the common covariance matrix estimate is identity, discriminitive function expressions are really simple). Here is the following code :

```
mu.k <- do.call("rbind", lapply(1:K, function(k) colMeans(X[Y == k-1,])))
mu.bar <- colMeans(mu.k)

X.pred <- predict(ldafit, newdata = X)
mu.k.pred <- predict(ldafit, newdata = as.data.frame(ldafit$means))
X.new <- X.pred$x #values of X in (LD1, LD2,..., LD9) space
mu.k.new <- mu.k.pred$x #idem for classes centroids

x.lim <- range(X.new[, 1]) #limit of our future grid
y.lim <- range(X.new[, 2])
x.grid <- 100
y.grid <- 100
grid <- cbind(seq(x.lim[1], x.lim[2], length = x.grid),
 rep(seq(y.lim[1], y.lim[2], length = y.grid), each = x.grid))

#make prediction for our grid points
getLdaClass <- function(grid, mu){
```

```
    K <- nrow(mu)
    apply(grid, 1,
     function(x) which.max(-0.5 * dist(rbind(x, mu))[1:K] + log(ldafit$prior)[1:K]))
}
grid.LdaClasses <- getLdaClass(grid, mu.k.new[, 1:2])

#plot data points prediction
plot(X.new[, 1], X.new[, 2], col = color[Y+1], type = "n",
 main = "LDA classification regions",
 xlab = "LD1", ylab = "LD2")
points(grid[, 1], grid[, 2], col = color[grid.ldaClasses + 1], pch = 19, cex = 0.3)
points(X.new[, 1], X.new[, 2], pch = Y + 1, cex = 0.8)
points(mu.k.new[, 1], mu.k.new[, 2], pch = 19, cex = 1.5)
```

Exercise 7 On the plot 5), if possible, show the quadratic discriminant bounaries (or regions).

Solution 7 It seems that with these data fitting qda is impossible. When we take train data for training the model with...

```
> qdafit <- qda(as.factor(X1~., data=zipdf)
```

... for different train set sizes we always get an error. The error is : "not enough observations in one class" (when size ¡ 3900) or "Class 0 is not full rank" (when train set size $\geq$ 3900).

Could we avoid these errors by transforming data ? (to have centered + identity covariance matrix data).

Exercise 8 On the plot 5) show k-nearest neighbours regions, for k you judge appropriate visually, i.e. by seeing the regions on the first two linear discriminant axes.

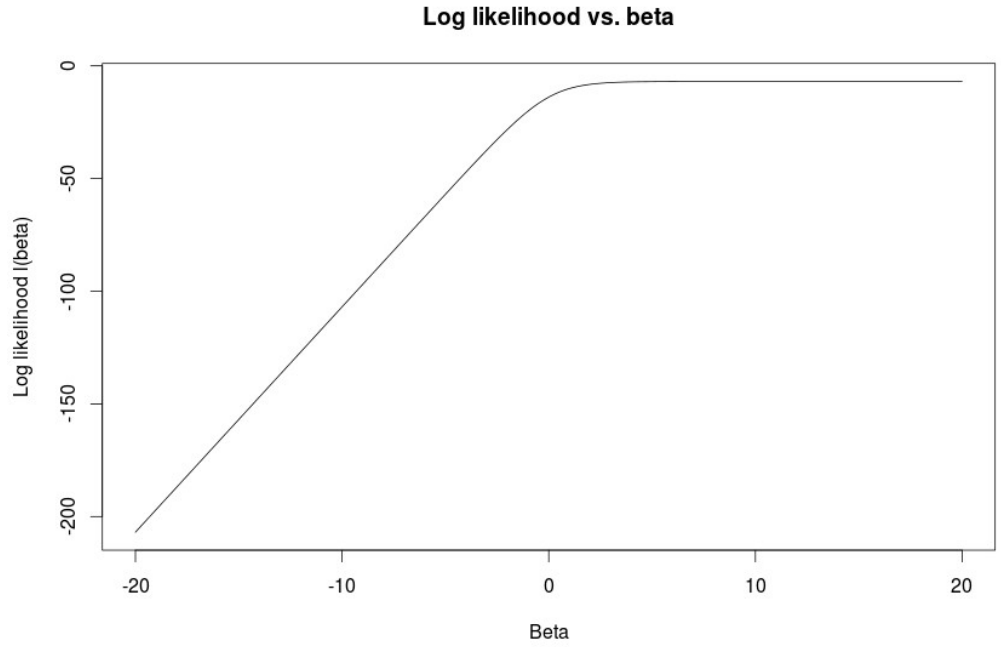
Solution 8 Here again, I am embarassed by the same problem than before (that I could avoid by knowing exact linear discriminant functions in the LDA space) : when we have a grid in the LDA space, how do we transform these vectors in the features space ? I have tried the matrix "Scaling" supposing that it was the LDA vectors matrix in the features space but it doesn't seem that it is correct (from what I have recently understood, it is the inverse).

MTH6312 Méthodes statistiques d’apprentissage

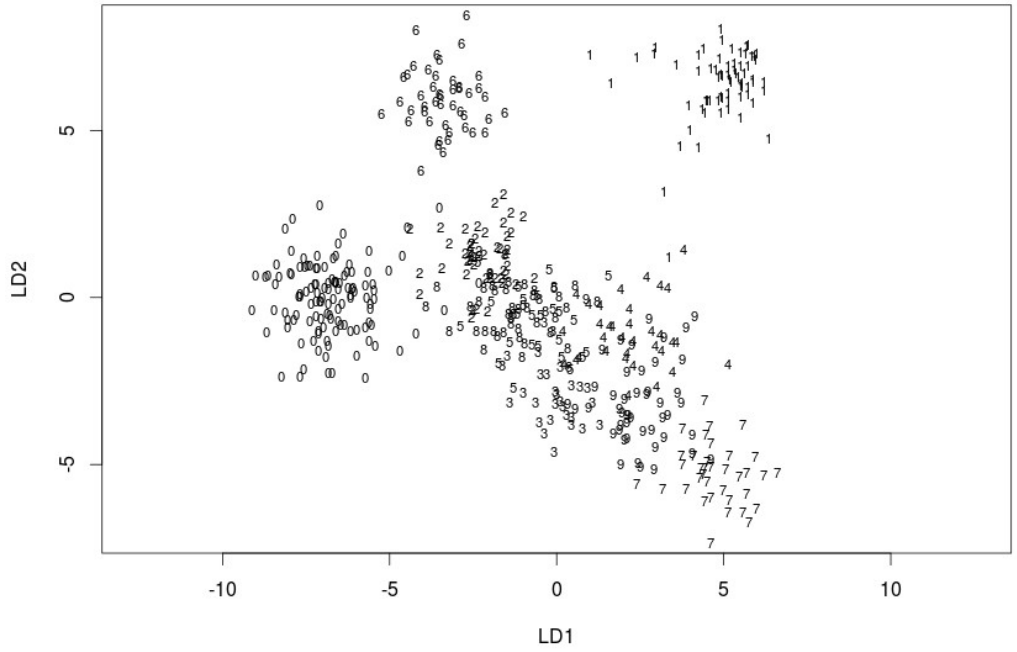
Exercice 9 – Annexe

Figures

Exercice 1



Exercice 5



Exercice 6