# Logistic Model to Classify the Handwritten Digits

We use the glmnet package , the description of the function are in the following:

fit a GLM with lasso or elasticnet regularization

**Description**

Fit a generalized linear model via penalized maximum likelihood. The regularization path is computed for the lasso or elasticnet penalty at a grid of values for the regularization parameter lambda. Can deal with all shapes of data, including very large sparse data matrices. Fits linear, logistic and multinomial, poisson, and Cox regression models.

**Usage**

glmnet(x, y,family=c("gaussian","binomial","poisson","multinomial","cox","mgaussian"),

weights, offset=NULL, alpha = 1, nlambda = 100,

lambda.min.ratio = ifelse(nobs<nvars,0.01,0.0001), lambda=NULL,

standardize = TRUE, intercept=TRUE, thresh = 1e-07, dfmax = nvars + 1,

pmax = min(dfmax \* 2+20, nvars), exclude, penalty.factor = rep(1, nvars),

lower.limits=-Inf, upper.limits=Inf, maxit=100000,

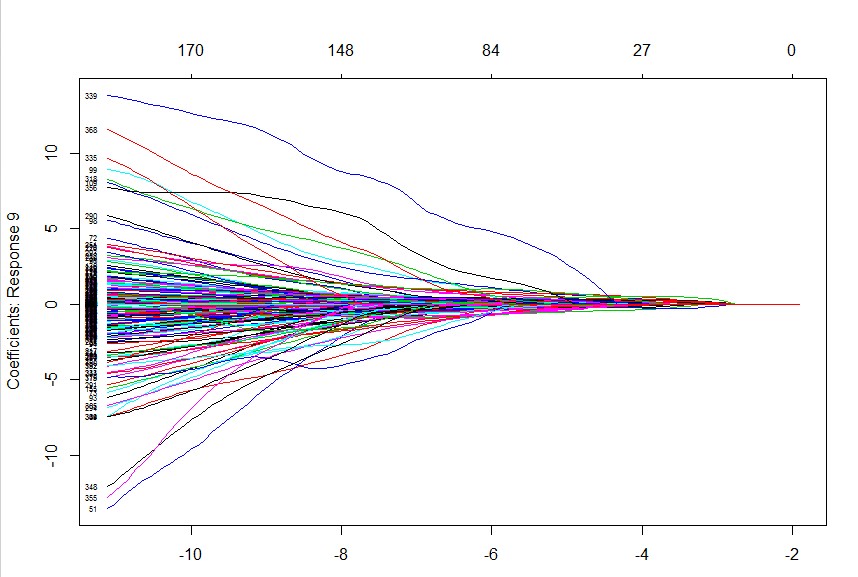
type.gaussian=ifelse(nvars<500,"covariance","naive"),

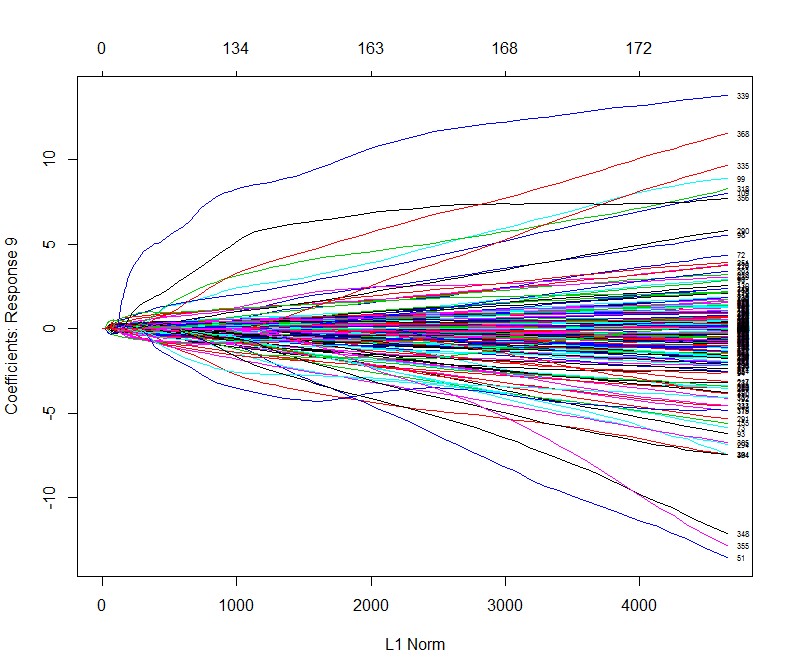
type.logistic=c("Newton","modified.Newton"),

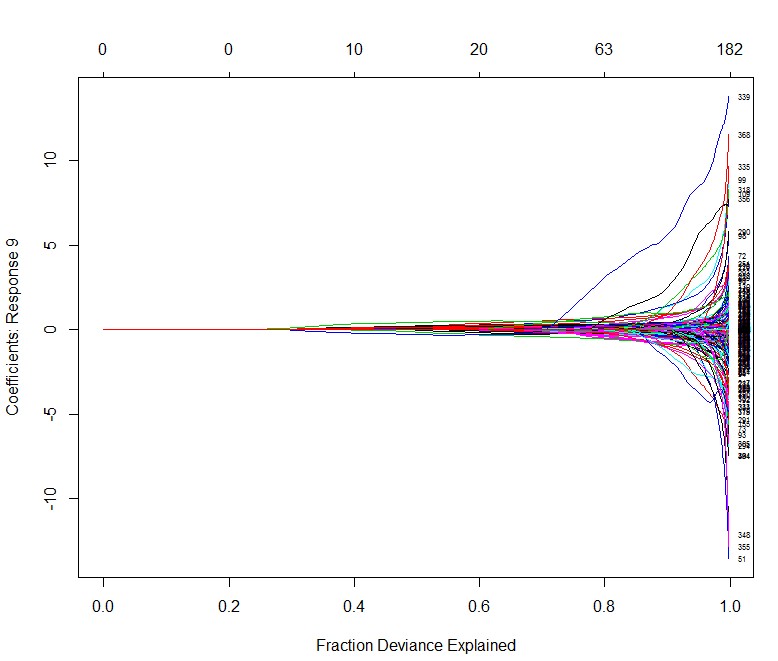
standardize.response=FALSE, type.multinomial=c("ungrouped","grouped"))

**Arguments**

|  |  |
| --- | --- |
| x | input matrix, of dimension nobs x nvars; each row is an observation vector. Can be in sparse matrix format (inherit from class "sparseMatrix" as in package Matrix; not yet available for family="cox") |
| y | response variable. Quantitative for family="gaussian", or family="poisson" (non-negative counts). For family="binomial" should be either a factor with two levels, or a two-column matrix of counts or proportions (the second column is treated as the target class; for a factor, the last level in alphabetical order is the target class). Forfamily="multinomial", can be a nc>=2 level factor, or a matrix with nc columns of counts or proportions. For either "binomial" or "multinomial", if y is presented as a vector, it will be coerced into a factor. For family="cox", y should be a two-column matrix with columns named 'time' and 'status'. The latter is a binary variable, with '1' indicating death, and '0' indicating right censored. The function Surv() in package **survival**produces such a matrix. For family="mgaussian", y is a matrix of quantitative responses. |
| family | Response type (see above) |
| weights | observation weights. Can be total counts if responses are proportion matrices. Default is 1 for each observation |
| offset | A vector of length nobs that is included in the linear predictor (a nobs x nc matrix for the"multinomial" family). Useful for the "poisson" family (e.g. log of exposure time), or for refining a model by starting at a current fit. Default is NULL. If supplied, then values must also be supplied to the predict function. |
| alpha | The elasticnet mixing parameter, with *0≤α≤ 1*. The penalty is defined as  *(1-α)/2||β||\_2^2+α||β||\_1.*  alpha=1 is the lasso penalty, and alpha=0 the ridge penalty. |
| nlambda | The number of lambda values - default is 100. |
| lambda.min.ratio | Smallest value for lambda, as a fraction of lambda.max, the (data derived) entry value (i.e. the smallest value for which all coefficients are zero). The default depends on the sample size nobs relative to the number of variables nvars. If nobs > nvars, the default is0.0001, close to zero. If nobs < nvars, the default is 0.01. A very small value oflambda.min.ratio will lead to a saturated fit in the nobs < nvars case. This is undefined for "binomial" and "multinomial" models, and glmnet will exit gracefully when the percentage deviance explained is almost 1. |
| lambda | A user supplied lambda sequence. Typical usage is to have the program compute its ownlambda sequence based on nlambda and lambda.min.ratio. Supplying a value oflambda overrides this. WARNING: use with care. Do not supply a single value for lambda(for predictions after CV use predict() instead). Supply instead a decreasing sequence of lambda values. glmnet relies on its warms starts for speed, and its often faster to fit a whole path than compute a single fit. |
| standardize | Logical flag for x variable standardization, prior to fitting the model sequence. The coefficients are always returned on the original scale. Default is standardize=TRUE. If variables are in the same units already, you might not wish to standardize. See details below for y standardization with family="gaussian". |
| intercept | Should intercept(s) be fitted (default=TRUE) or set to zero (FALSE) |
| thresh | Convergence threshold for coordinate descent. Each inner coordinate-descent loop continues until the maximum change in the objective after any coefficient update is less than thresh times the null deviance. Defaults value is 1E-7. |
| dfmax | Limit the maximum number of variables in the model. Useful for very large nvars, if a partial path is desired. |
| pmax | Limit the maximum number of variables ever to be nonzero |
| exclude | Indices of variables to be excluded from the model. Default is none. Equivalent to an infinite penalty factor (next item). |
| penalty.factor | Separate penalty factors can be applied to each coefficient. This is a number that multiplieslambda to allow differential shrinkage. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables (and implicitly infinity for variables listed in exclude). Note: the penalty factors are internally rescaled to sum to nvars, and the lambda sequence will reflect this change. |
| lower.limits | Vector of lower limits for each coefficient; default -Inf. Each of these must be non-positive. Can be presented as a single value (which will then be replicated), else a vector of lengthnvars |
| upper.limits | Vector of upper limits for each coefficient; default Inf. See lower.limits |
| maxit | Maximum number of passes over the data for all lambda values; default is 10^5. |
| type.gaussian | Two algorithm types are supported for (only) family="gaussian". The default whennvar<500 is type.gaussian="covariance", and saves all inner-products ever computed. This can be much faster than type.gaussian="naive", which loops throughnobs every time an inner-product is computed. The latter can be far more efficient for nvar >> nobs situations, or when nvar > 500. |
| type.logistic | If "Newton" then the exact hessian is used (default), while "modified.Newton" uses an upper-bound on the hessian, and can be faster. |
| standardize.response | This is for the family="mgaussian" family, and allows the user to standardize the response variables |
| type.multinomial | If "grouped" then a grouped lasso penalty is used on the multinomial coefficients for a variable. This ensures they are all in our out together. The default is "ungrouped" |

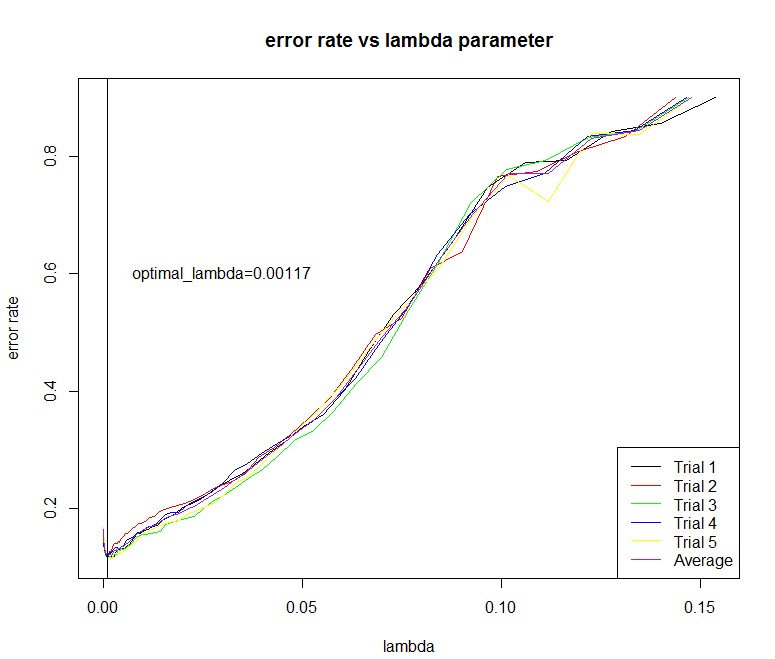






Here we use the method family=”multinormial” The error rate plots after running the algorithm 5 times with default 100 lambda are in the following plot.

From the below plot we can see that the optimal lambda is 0.0017.



#Then we do logistic regression on the whole training data with the optimal training data

model=glmnet(training.data,training.label,"multinomial")

#do test on test data

start\_time=proc.time()

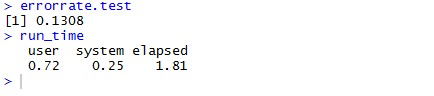
pre.test=predict(model,test.data,lambda.optimal,"class")

end\_time=proc.time()

run\_time=end\_time-start\_time

errorrate.test=sum((pre.test!=test.label))/nrow(test.data)

The results we get are in the following:



|  |  |
| --- | --- |
| Average predict time of one data stream(s) | The error rate of LDA method |
| 1.81e(-4) | 0.1308 |

**Code Appendix**

# use the logistic model to classify the handwritten digits

data=load("C:/Users/Christina/Desktop/digitsdata.RData")

image(t(1 - training.data[3,1,,])[,20:1],col=gray(seq(0, 1, length.out=256)),axes=FALSE, asp=1)

num.class <- dim(training.data)[1] # Number of classes

num.training <- dim(training.data)[2] # Number of training data per class

d <- prod(dim(training.data)[3:4]) # Dimension of each training image (rowsxcolumns)

num.test <- dim(test.data)[2] # Number of test data

dim(training.data) <- c(num.class \* num.training, d) # Reshape training data to 2-dim matrix

dim(test.data) <- c(num.class \* num.test, d) # Same for test.

training.label <- rep(0:9, num.training)# Labels of training data.

test.label <- rep(0:9, num.test) # Labels of test data

#install glmnet packages

library("glmnet")

rownames(training.data)=training.label

#divide training data into 2 parts:training and validation to compute optinal lambda

error\_rate=matrix(0,nrow=5,ncol=100)

lambda=matrix(0,nrow=5,ncol=100)

for(i in 1:5){

# this is because we want to run the whole data 5 times

# we sample rows aaccording to their labels and get the corresponding traing data and test data.

#class 0

class0=training.data[rownames(training.data)=="0",]

train\_0=sample((nrow(class0)),size=400,replace = FALSE, prob = NULL)

train0=class0[train\_0, ]

test0=class0[-train\_0, ]

#class 1

class1=training.data[rownames(training.data)=="1",]

train\_1=sample(nrow(class1),size=400,replace = FALSE, prob = NULL)

train1=class1[train\_1, ]

test1=class1[-train\_1, ]

#class 2

class2=training.data[rownames(training.data)=="2",]

train\_2=sample(nrow(class2), size=400,replace = FALSE, prob = NULL)

train2=class2[train\_2, ]

test2=class2[-train\_2, ]

# class 3

class3=training.data[rownames(training.data)=="3",]

train\_3=sample(nrow(class3), size=400,replace = FALSE, prob = NULL)

train3=class3[train\_3, ]

test3=class3[-train\_3, ]

#class 4

class4=training.data[rownames(training.data)=="4",]

train\_4=sample(nrow(class4), size=400,replace = FALSE, prob = NULL)

train4=class4[train\_4, ]

test4=class4[-train\_4, ]

#class 5

class5=training.data[rownames(training.data)=="5",]

train\_5=sample(nrow(class5), size=400,replace = FALSE, prob = NULL)

train5=class5[train\_5, ]

test5=class5[-train\_5, ]

#class 6

class6=training.data[rownames(training.data)=="6",]

train\_6=sample(nrow(class6), size=400,replace = FALSE, prob = NULL)

train6=class6[train\_6, ]

test6=class6[-train\_6, ]

#class 7

class7=training.data[rownames(training.data)=="7",]

train\_7=sample(nrow(class7), size=400,replace = FALSE, prob = NULL)

train7=class7[train\_7, ]

test7=class7[-train\_7, ]

# class 8

class8=training.data[rownames(training.data)=="8",]

train\_8=sample(nrow(class8), size=400,replace = FALSE, prob = NULL)

train8=class8[train\_8, ]

test8=class8[-train\_8, ]

#class 9

class9=training.data[rownames(training.data)=="9",]

train\_9=sample(nrow(class9), size=400,replace = FALSE, prob = NULL)

train9=class9[train\_9, ]

test9=class9[-train\_9, ]

train=rbind(train0,train1,train2,train3,train4,train5,train6,train7,train8,train9)

test=rbind(test0,test1,test2,test3,test4,test5,test6,test7,test8,test9)

#do logistic regression on the sampled training set

glmmodel=glmnet(train,rownames(train),"multinomial")

lambda[i,]=t(glmmodel$lambda)

for(j in 1:length(glmmodel$lambda)){

test\_pre=predict(glmmodel,test,glmmodel$lambda[j],"class")

error\_rate[i,j]=sum(test\_pre!=rownames(test))/nrow(test)

}

}

mean\_error\_rate=colMeans(error\_rate)

lambda.optimal=mean(lambda[,apply(t(mean\_error\_rate),1,which.min)])

m=as.matrix(colMeans(lambda))

matplot(cbind(t(lambda),m),cbind(t(error\_rate),mean\_error\_rate),type='l',col=c('black','red','green','blue','yellow','purple'),ylab='error rate',xlab='lambda',lty =1,cex=2)

title(main='error rate vs lambda parameter')

legend(legend=c('Trial 1','Trial 2','Trial 3','Trial 4','Trial 5','Average'),col=c('black','red','green','blue','yellow','purple'),'bottomright',lty=1)

abline(v=lambda.optimal)

text(x=0.03,y=0.6,label='optimal\_lambda=0.00117')

plot(glmmodel,xvar="lambda",label=TRUE)

plot(glmmodel,label=TRUE)

plot(glmmodel,xvar="dev",label=TRUE)

# we get lambda.optimal=0.00117

#Then we do logistic regression on the whole training data

model=glmnet(training.data,training.label,"multinomial")

#do test on test data

start\_time=proc.time()

pre.test=predict(model,test.data,lambda.optimal,"class")

end\_time=proc.time()

run\_time=end\_time-start\_time

errorrate.test=sum((pre.test!=test.label))/nrow(test.data)

> errorrate.test

[1] 0.1308

> run\_time

user system elapsed

0.72 0.25 1.81