L1 and L2 Penalized Regression Models

Jelle Goeman   Rosa Meijer   Nimisha Chaturvedi

Package version 0.9-52
Date: November 7, 2024

Contents

1 Citing *penalized*. **2**

2 Introduction **2**

3 Penalized likelihood estimation **3**
3.1 the nki70 data 3
3.2 the penalized function 3
3.3 choice of lambda 4
3.4 penfit objects 4
3.5 standardization 5
3.6 unpenalized covariates 6
3.7 factors 6
3.8 fitting in steps 7
3.9 a positivity constraint 7
3.10 fused lasso 10

4 Pretesting **11**

5 Cross-validation and optimization **12**
5.1 cross-validation 12
5.2 approximated cross-validation 13
5.3 breslow objects 14
5.4 profiling the cross-validated log likelihood 15
5.5 optimizing the cross-validated likelihood 16

6 A note on standard errors and confidence intervals **18**

1 Citing *penalized*.

To cite the package *penalized*, please cite Goeman, J. J., L1 penalized estimation in the Cox proportional hazards model. *Biometrical Journal* 52(1), 70–84.

2 Introduction

This short note explains the use of the *penalized* package. The package is designed for penalized estimation in generalized linear models. The lasso and elastic net algorithm that it implements is described in Goeman (2010).

The supported models at this moment are linear regression, logistic regression, poisson regression and the Cox proportional hazards model, but others are likely to be included in the future. As to penalties, the package allows an L1 absolute value ("lasso") penalty Tibshirani (1996, 1997), an L2 quadratic ("ridge") penalty (Hoerl and Kennard, 1970; Le Cessie and van Houwelingen, 1992; Verweij and Van Houwelingen, 1994), or a combination of the two (the "naive elastic net" of Zou and Hastie, 2005). It is also possible to have a *fused lasso* penalty with L1 absolute value ("lasso") penalty on the coefficients and their differences Tibshirani et al. (2005); Tibshirani and Wang (2007). The package also includes facilities for likelihood cross-validation and for optimization of the tuning parameter.

L1 and L2 penalized estimation methods shrink the estimates of the regression coefficients towards zero relative to the maximum likelihood estimates. The purpose of this shrinkage is to prevent overfit arising due to either collinearity of the covariates or high-dimensionality. Although both methods are shrinkage methods, the effects of L1 and L2 penalization are quite different in practice. Applying an L2 penalty tends to result in all small but non-zero regression coefficients, whereas applying an L1 penalty tends to result in many regression coefficients shrunk exactly to zero and a few other regression coefficients with comparatively little shrinkage. Combining L1 and L2 penalties tends to give a result in between, with fewer regression coefficients set to zero than in a pure L1 setting, and more shrinkage of the other coefficients. The *fused lasso* penalty, an extension of the lasso penalty, encourages sparsity of the coefficients and their differences by penalizing the L1-norm for both of them at the same time, thus producing sparse and piecewise constant stretches of non-zero coefficients. The amount of shrinkage is determined by tuning parameters λ_1 and λ_2. A value of zero always means no shrinkage (= maximum likelihood estimation) and a value of infinity means infinite shrinkage (= setting all regression coefficients to zero). For more details about the methods, please refer to the above-mentioned papers.

It is important to note that shrinkage methods are generally not invariant to the relative scaling of the covariates. Before fitting a model, it is prudent to consider if the covariates already have a natural scaling relative to each other or whether they should be standardized.

The main algorithm for L1 penalized estimation (lasso, elastic net) that used in this package is documented in Goeman (2010). It has been combined with ideas from Eilers et al. (2001) and Van Houwelingen et al. (2006) for efficient L2 penalized estimation. The algorithm used for fused lasso penalized estimation is described in Chaturvedi (2012)

3 Penalized likelihood estimation

The basic function of the package is the `penalized` function, which performs penalized estimation for fixed values of λ_1 and λ_2. Its syntax has been loosely modeled on that of the functions `glm` (package *stats*) and `coxph` (package *survival*), but it is slightly more flexible in some respects. Two main input types are allowed: one using *formula* objects, one using matrices.

3.1 the nki70 data

As example data we use the 70 gene signature of Van 't Veer et al. (2002) in the gene expression data set of Van de Vijver et al. (2002).

```
> library(penalized)
> library(survival)
> data(nki70)
```

This loads a *data.frame* with 144 breast cancer patients and 77 covariates. The first two covariates indicate the survival time and event status (time is in months), the next five are clinical covariates (diameter of the tumor, lymph node status, estrogen receptor status, grade of the tumor and age of the patients), and the other 70 are gene expression measurements of the 70 molecular markers. As we are interested in survival as an outcome, we also need the survival package.

```
> set.seed(1)
```

3.2 the penalized function

The `penalized` function can be used to fit a penalized prediction model for prediction of a response. For example, to predict the Estrogen Receptor status `ER` for the patients in the `nki70` data with the two markers "DIAPH3" and "NUSAP1" at $\lambda_1 = 0$ and $\lambda_2 = 1$, we can say (all are equivalent)

```
> fit <- penalized(ER, ~DIAPH3+NUSAP1, data=nki70, lambda2=1)
> fit <- penalized(ER, nki70[,10:11], data=nki70, lambda2=1)
> fit <- penalized(ER~DIAPH3+NUSAP1, data=nki70, lambda2=1)
```

The covariates may be specified in the second function argument (*penalized*) as a *formula* object with an open left hand side, as in the first line. Alternatively, they may be specified as a *matrix* or *data.frame*, as in the second line. If, as here, they are supplied as a *data.frame*, they are coerced to a matrix.

For consistency with `glm` and `coxph` the third option is also allowed, in which the covariates are included in the first function argument.

The `penalized` function tries to determine the appropriate generalized linear model from the *response* variable. This automatic choice may not always be appropriate. In such cases the model may be specified explicitly using the *model* argument.

For the examples in the rest of this vignette we use the Cox proportional hazards model, using the survival time (`Surv(time,event)`) as the response to be predicted. This is a *Surv* object.

```
> fit <- penalized(Surv(time,event)~DIAPH3+NUSAP1, data=nki70, lambda2=1)
```

We use `attach` to avoid specifying the *data* argument every time.

```
> attach(nki70)
```

3.3 choice of lambda

It is difficult to say in advance which value of *lambda1* or *lambda2* to use. The *penalized* package offers ways of finding optimal values using cross-validation. This is explained in Section 5

Note that for small values of *lambda1* or *lambda2* the algorithm be very slow, may fail to converge or may run into numerical problems, especially in high-dimensional data. When this happens, increase the value of *lambda1* or *lambda2*.

It is possible to specify both *lambda1* or *lambda2*. In this case both types of penalties apply, and a so-called *elastic net*.

```
> fit <- penalized(Surv(time,event)~DIAPH3+NUSAP1, data=nki70, lambda1=1, lambda2=1)
```

Sometimes it can be useful to have different values of *lambda1* or *lambda2* for different covariates. This can be done in *penalized* by specifying *lambda1* or *lambda2* as a vector.

```
> fit <- penalized(Surv(time,event)~DIAPH3+NUSAP1, data=nki70, lambda2=c(1,2))
```

3.4 penfit objects

The penalized function returns a *penfit* object, from which useful information can be extracted. For example, to extract regression coefficients, (martingale) residuals, individual relative risks and baseline survival curve, write

```
> residuals(fit)[1:10]
```

```
       125        127        128        129        130        132        134
-0.1554545  0.7119251 -0.3710430 -0.2335851 -0.4101212 -0.3424704  0.7235891
       135        136        137
-0.6391287  0.7468025 -0.4893899
```

```
> fitted(fit)[1:10]
```

```
      125       127       128       129       130       132       134       135
0.4810103 1.0518078 0.9170982 0.7227633 1.2690045 1.1124577 0.8552757 1.4158589
      136       137
1.1712807 0.6481456
```

```
> basesurv(fit)
```

```
A "breslow" object with 1 survival curve and 50 time points.
```

See `help(penfit)` for more information on *penfit* objects and Section 5.3 on *breslow* objects.

The `coefficients` function extracts the named vector of regression coefficients. It has an extra second argument *which* that can be used to specify which coefficients are of interest. Possible values of *which* are `nonzero` (the default) for extracting all non-zero coefficients, `all` for all coefficients, and `penalized` and *unpenalized* for only the penalized or unpenalized ones.

```
> coefficients(fit, "all")
```

To extract the loglikelihood of the fit and the evaluated penalty function, use

```
> loglik(fit)

[1] -258.5714

> penalty(fit)

      L1       L2
0.000000 1.409874
```

The `loglik` function gives the loglikelihood without the penalty, and the `penalty` function gives the fitted penalty, i.e. for L1 `lambda1` times the sum of the absolute values of the fitted penalized coefficients, and for L2 it is 0.5 times `lambda1` times the sum of squared penalized coefficients.

The *penfit* object can also be used to generate predictions for new data using the `predict` function. Pretending that the first three subjects in the `nki70` data are new subjects, we can find their predicted survival curves with either of

```
> predict(fit, ~DIAPH3+NUSAP1, data=nki70[1:3,])

A "breslow" object with 3 survival curves and 50 time points.

> predict(fit, nki70[1:3,c("DIAPH3","NUSAP1")])

A "breslow" object with 3 survival curves and 50 time points.
```

See Section 5.3 for more on breslow objects. We can get five year survival predictions by saying

```
> pred <- predict(fit, nki70[1:3,c("DIAPH3","NUSAP1")])
> survival(pred, time=5)

      125       127       128
0.8723044 0.7417559 0.7706856
```

3.5 standardization

If the covariates are not naturally on the same scale, it is advisable to standardize them. The function argument *standardize* (default: `FALSE`) standardizes the covariates to unit second central moment before applying penalization. This standardization makes sure that each covariate is affected more or less equally by the penalization.

The fitted regression coefficients that the function returns have been scaled back and correspond to the original scale of the covariates. To extract the regression coefficients of the standardized covariates, use the `coefficients` function with `standardize = TRUE`. This option is also available if the model was not fitted with standardized covariates, as the covariates are always standardized internally for numerical stability. To find the weights used by the function, use `weights(fit)`.

```
> coefficients(fit)

  DIAPH3   NUSAP1
0.223606 1.176807

> coefficients(fit, standardize = TRUE)

    DIAPH3     NUSAP1
0.05243638 0.31416138

> weights(fit)

   DIAPH3    NUSAP1
0.2345035 0.2669609
```

3.6 unpenalized covariates

In some situations it is desirable that not all covariates are subject to a penalty. Any additional covariates that should be included in the model without being penalized can be specified separately. using the third function argument (*unpenalized*). For example (the two commands below are equivalent)

```
> fit <- penalized(Surv(time,event), nki70[,8:77], ~ER, lambda2=1)
> fit <- penalized(Surv(time,event)~ER, nki70[,8:77], lambda2=1)
```

This adds estrogen receptor status as an unpenalized covariate. Note in the second line that right hand side of the *formula* object in the *response* argument is automatically taken to be the *unpenalized* argument because the *penalized* argument was given by the user.

In linear and logistic regression the intercept is by default never penalized. The use of an intercept can be suppressed with `penalized = ~0`. The intercept is always removed from the penalized model matrix, unless the penalized model consists of only an intercept.

It is possible to include an offset term in the model. Use the `offset` function in the *unpenalized* argument, which must then be of *formula* type. The Cox model implementation allows `strata` terms.

```
> fit <- penalized(Surv(time,event)~strata(ER), nki70[,8:77], lambda2=1)
```

3.7 factors

If some of the factors included in the *formula* object *penalized* are of type *factor*, these are automatically made into dummy variables, as in `glm` and `coxph`, but in a special way that is more appropriate for penalized regression.

Unordered factors are turned into as many dummy variables as the factor has levels. This ensures a symmetric treatment of all levels and guarantees that the fit does not depend on the ordering of the levels. See `help(contr.none)` for details.

Ordered factors are turned into dummy variables that code for the difference between successive levels (one dummy less than the number of levels). L2 penalization on such factors therefore leads to small successive differences;

L1 penalization leads to ranges of successive levels with identical effects. See `help(contr.diff)` for details.

When fitting a model with factors with more than two levels with an L1 penalty, it is advisable to add a small L2 penalty as well in order to speed up convergence. By varying the L2 penalty it can be checked that the L2 penalty is not so large that it influences the estimates.

To override the automatic choice of contrasts, use `C` (package *stats*).

The *response* argument may also be also be specified as a *factor* in a logistic regression model. In that case, the value `levels(response)[1]` is treated as a failure (0), and all other values as a success (1).

3.8 fitting in steps

In some cases it may be interesting to visualize the effect of changing the tuning parameter *lambda1* or *lambda2* on the values of the fitted regression coefficients. This can be done using the function argument *steps* in combination with the `plotpath` function. At this moment, this functionality is only available for visualizing the effect of *lambda1* (not for*fused lasso* estimation).

When using the *steps* argument, the function starts fitting the model at the maximal value of λ_1, that is the smallest value that shrinks all regression coefficients to zero. From that value it continues fitting the model for *steps* successively decreasing values of λ_1 until the specified value of *lambda1* is reached.

If the argument *steps* is supplied to `penalized`, the function returns a *list* of *penfit* objects. These can be accessed individually or their coefficients can be plotted using `plotpath`.

```
> fit <- penalized(Surv(time,event), nki70[,8:77], lambda1=1,
      steps=50, trace = FALSE)
> plotpath(fit, log="x")
```

Following Park and Hastie (2007) it is possible to choose the values of λ_1 in such a way that these are the change-points at which the active set changes. This can be done by setting *steps* = *"Park"*.

```
> fit <- penalized(Surv(time,event), nki70[,8:77], lambda1=1,
      steps="Park", trace = FALSE)
```

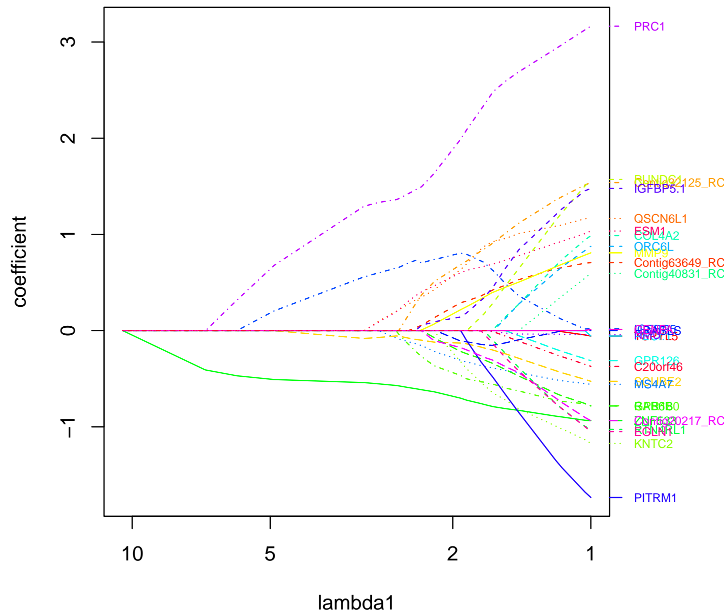
Note that `plotpath` plots the unstandardized coefficients by default. Standardized coefficients can be plotted (even when the model was not fitted with standardized coefficients) with the *standardize* argument.

3.9 a positivity constraint

In some applications it is natural to restrict all estimated regression coefficients to be non-negative. Such a positivity constraint is an alternative type of constrained estimation that is easily combined with L1 and L2 penalization in the algorithm implemented in the *penalized* package.

To add a positivity restriction to the regression coefficients of all penalized covariates, set the function argument *positive* to `TRUE` (the default is `FALSE`). Note that it is not strictly necessary to also include an L1 or L2 penalty; the model can also be fitted with only a positivity constraint.

```
> plotpath(fit, log="x")
```

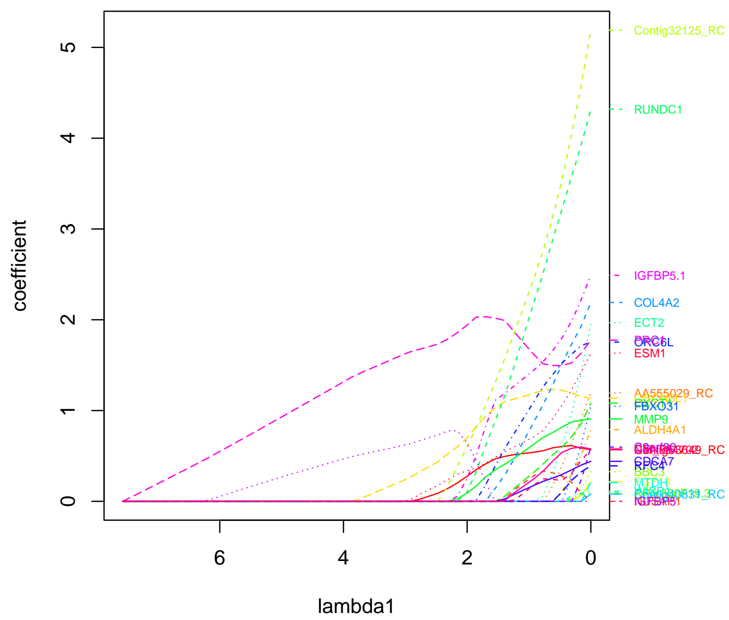



```
> fit <- penalized(Surv(time,event), nki70[,8:77], positive=TRUE)

> coefficients(fit)

Contig63649_RC     AA555029_RC        ALDH4A1        QSCN6L1           FGF18
    0.56883350      1.19479968     0.78893478     1.13549561      0.21511238
Contig32125_RC            BBC3    RP5.860F19.3          OXCT1            MMP9
    5.19060693      0.32578805     0.09279051     1.08253778      0.90786251
        RUNDC1            ECT2          WISP1           MTDH  Contig40831_RC
    4.31995520      1.96799708     0.11182254     0.20738967      0.08058631
        COL4A2          FBXO31          ORC6L           RFC4           CDCA7
    2.19059076      1.04830672     1.75330155     0.39074008      0.44076424
        C9orf30        IGFBP5.1           PRC1          CENPA        NM_004702
    0.59695825      2.48990214     1.77488012     0.57635421      0.56779362
          ESM1
    1.63222843
```

It is also possible to constrain only part of the regression coefficients to be non-negative by giving the *positive* argument as a logical vector.

```
> coef(penalized(Surv(time,event), nki70[,8:16], positive=c(F,rep(T,8))))
```

```
> fit0 <- penalized(Surv(time,event), nki70[,8:77], positive=TRUE,
      steps=50)
> plotpath(fit0)
```



3.10 fused lasso

For problems involving features that can be ordered in some meaningful way, it might be useful to take into account the information about their spatial structure while estimating the coefficients. For example, copy number data exhibit spatial correlation along the genome. This suggests that feature selection should take genomic location into account for producing more interpretable results for copy number based classifiers. Fused lasso takes the genomic location into account by putting a L1 penalty on the coefficients as well as on their differences. It, thus produces sparse results with local constancy of the coefficient profile. Fot estimating using *fused lasso* one can use the basic *penalized* function of the package with the function argument *fusedl* set to `TRUE`. The argument *fusedl* can take values in two form: logical or a vector. For example in case of copy number data if the information about the genomic location is available then *fusedl* can be given as an input, a vector of these locations. If the function argument *fusedl* is set to `TRUE` or it is a vector then the *penalized* function performs *fused lasso* penalized estimation for a fixed value of λ_1 and λ_2. Note that for *fused lasso* estimation, the value for λ_2 given in the function is used for putting L1 penalty on the differences of the coefficients. We demonstrate the fused lasso feature of the penalized function by applying it on a simulated dataset with binomial response. We generate a data set with 100 samples and 70 probes, with mean equal to 0 and variance equal to 0.5.

```
>  X <- matrix(0,70,100)
>   for (i in 1:100){
       X[1:70,i] <- rnorm(70,mean=0,sd=0.5)
    }
>  colnames(X) = as.character (1:ncol(X))
>   rownames(X) = as.character (1:nrow(X))
```

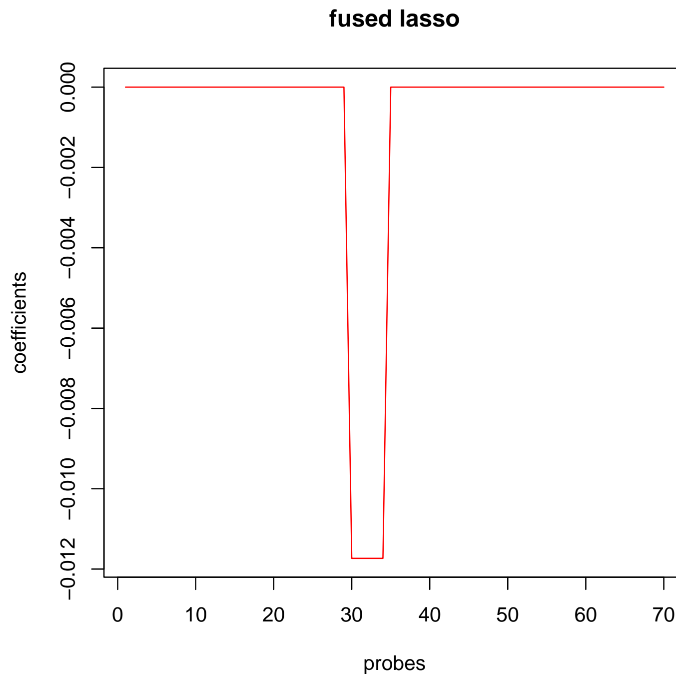
Out of these 100 samples, 50 are selected randomly for getting aberrations in the region 30:40. The mean for the aberrated region is taken to be -0.7 and variance 0.5.

```
>  a <- sample(1:ncol(X),50,prob=rep(0.5,length(1:ncol(X))))
>  for (i in 1:50){
        X[30:40,a[i]]<-rnorm(length(30:40),mean = -0.7 ,sd=0.5)
     }
```

We generate the probabilities of the samples being 1 or 0, by using beta equal to -1 in logistic model.

```
>      Xbeta <- rnorm(100, mean = 0, sd = 0.5)
>      Xbeta[a] <- rnorm (length(a) , mean = -0.7 , sd = 0.5)
>      beta <- numeric(100)
>      beta [1:length(beta)] <- -1
>    responsep <- numeric(100)
>  for(i in 1:100){
       coeff <- -beta[i] * Xbeta[i]
       responsep[i] <- 1/(1+exp(coeff))
    }
```

```
> fit <- penalized(response, X, lambda1 = 2, lambda2=3,fusedl=TRUE)
> plot(coefficients(fit,"all")[-1],main = "fused lasso", col="red",xlab = "probes",ylab =
```




```
>   X <- t(X)
>     response=responsep
>       for(i in 1:100){
        response[i] <- sample(0:1 , size = 1  , prob = c((1-responsep[i]),responsep[i]))
    }
```

For estimating the coefficients using fused lasso, the *flasso* argument in the *penalized* function can take two forms. If it is logical then the genomic location (for example chromosome number in copy number data) information becomes 1 for every probe. Otherwise if the information is available then it can be given as a vector to the *flasso* argument.

```
> fit <- penalized(response, X, lambda1 = 2, lambda2=2,fusedl=TRUE)
```

```
> chr = c(rep(1,30),rep(2,20),rep(3,10),rep(4,10))
> fit <- penalized(response, X, lambda1 = 2, lambda2=2,fusedl=chr)
```

4 Pretesting

Before fitting a penalized regression model it can be worthwhile to test the global null hypothesis of no association between any of the predictor variables and the

response. A package that can do this is, and which ties very closely to the *penalized* package is the *globaltest* package, available from `www.bioconductor.org`. The package can be installed using the bioconductor install script

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("globaltest")
```

The interface of *globaltest* is very similar to the interface of *penalized*. To test for any evidence of association in the `nki70` data, say

```
> gt(Surv(time,event), nki70[,8:77])
```

The resulting p-value can be interpreted as a global indicator of predictive ability. Data sets that have a significant test result almost always have a an optimal lambda value smaller than infinity.

See the vignette of the *globaltest* package for details.

5 Cross-validation and optimization

Cross-validation can be used to assess the predictive quality of the penalized prediction model or to compare the predictive ability of different values of the tuning parameter.

The *penalized* package uses likelihood cross-validation for all models. Likelihood cross-validation has some advantages over other optimization criteria: it tends to be a continuous function of the tuning parameter; it can be defined in a general way for almost any model, and it does not require calculation the effective dimension of a model, which is problematic in L1 penalized models. For the Cox proportional hazards model, the package uses cross-validated log partial likelihood (Verweij and Van Houwelingen, 1993), which is a natural extension of the cross-validated log likelihood to the Cox model.

Five functions are available for calculating the cross-validated log likelihood and for optimizing the cross-validated log likelihood with respect to the tuning parameters. They have largely the same arguments. See `help(cvl)` for an overview.

5.1 cross-validation

The function `cvl` calculates the cross-validated log likelihood for fixed values of λ_1 and λ_2.

It accepts the same arguments as `penalized` (except *steps*: see `profL1` below) as well as the *fold* argument. This will usually be a single number k to indicate k-fold cross-validation. In that case, the allocation of the subjects to the folds is random. Alternatively, the precise allocation of the subjects into the folds can be specified by giving *fold* as a vector of the length of the number of subjects with values form 1 to k, each indicating the fold allocation of the corresponding subject. The default is to do leave-one-out cross-validation. For having cross-validated fused lasso estimates, one should set the argument *fusedl* to `TRUE`.

In addition there is the argument *approximate* (default value is `FALSE`). If its value is set to `TRUE`, instead of true cross-validation an approximation method

is used that is much faster. This method is explained in more detail in the next subsection. When a linear model is fitted, the approximation method is no longer approximative but results in exact answers. For this reason, the package will automatically set the value of *approximate* to `TRUE` in case of a linear model. This argument does not works for fused lasso cross-validation.

The function `cvl` returns a names *list* with four elements:

`cvl` the cross-validated log likelihood.

`fold` the fold allocation used; this may serve as input to a next call to `cvl` to ensure comparability.

`predictions` the predictions made on each left-out subject. The format depends on the model used. In logistic regression this is just a vector of probabilities. In the Cox model this is a collection of predicted survival curves (a *breslow* object). In the linear model this is a collection of predicted means and predicted standard deviations (the latter are the maximum penalized likelihood estimates of σ^2).

`fullfit` the fit on the full data (a *penfit* object)

```
> fit <- cvl(Surv(time,event), nki70[,8:77], lambda1=1, fold=10)

> fit$cvl

[1] -255.2703

> fit$fullfit

Penalized cox regression object
70 regression coefficients of which 28 are non-zero

Loglikelihood =         -214.92
L1 penalty =         24.29771         at lambda1 =  1

> fit <- cvl(Surv(time,event), nki70[,8:77], lambda1=2, fold=fit$fold)
```

5.2 approximated cross-validation

To save time, one can choose to set the argument *approximate* in the function `cvl` to `TRUE`. For now, this option is only available for ridge models, so models where the lasso penalty equals 0.

In that case the cross-validated likelihood will not be calculated by leaving out one or a set of observation each time and refitting the model, but will be based on approximations. These approximations are based on a Taylor expansion around the estimate of the full model. Since refitting the model is no longer necessary, a lot of time can be saved in this way.

The results are in most cases quite accurate, but the method tends to be a little too optimistic which results in slightly too high values of the corresponding *cvl*. The method works best for large data sets and a resampling scheme with many folds.

The same option can be chosen in `optL2` and `profL2` (see below). Here one must again be aware of the tendency to be a little too optimistic which will

result in optimal penalty values that are a little smaller than the ones found by real cross-validation.

In the following example, the cross-validated log-likelihood is calculated twice. First by using leave-one-out cross-validation, then by using the approximation method.

```
> fit1 <- cvl(Surv(time,event), nki70[,8:77], lambda2=10)
> fit1$cvl
[1] -251.0248
> fit2 <- cvl(Surv(time,event), nki70[,8:77], lambda2=10, approximate=TRUE)
> fit2$cvl
[1] -250.9277
```

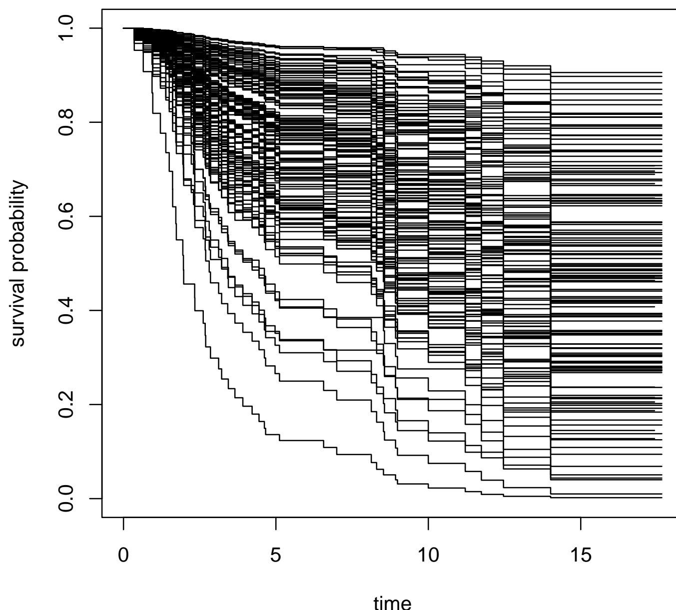
As we can see, the answers are very similar.

5.3 breslow objects

The *breslow* class is defined in the *penalized* package to store estimated survival curves. They are used for the predictions in cross-validation and for the baseline survival estimated in the `penalized` function. See `help(breslow)` for details.

```
> fit$predictions
A "breslow" object with 144 survival curves and 51 time points.
> time(fit$predictions)
 [1]  0.0000000  0.3531828  0.6488706  0.9363276  0.9609856  1.2101300
 [7]  1.3880903  1.5003422  1.6098563  1.6125941  1.7166324  1.7330595
[13]  1.9466119  1.9657769  1.9739904  2.2231348  2.2970568  2.3353867
[19]  2.3408624  2.6146475  2.6803559  2.6967830  2.8117728  2.8528405
[25]  3.1211499  3.2197125  3.4195756  3.4387406  3.6550308  3.9151266
[31]  4.2190281  4.4462697  4.6214921  4.6625599  4.9719370  5.1170431
[37]  6.5653662  6.9952088  8.1286790  8.3039014  8.5284052  8.5612594
[43]  8.9253936  8.9883641  9.9986311 11.2114990 11.7399042 12.4654346
[49] 14.0123203 17.4209446 17.6591376
> as.data.frame(basesurv(fit$fullfit))[1:10,]
    survival      time
1  1.0000000 0.0000000
2  0.9947948 0.3531828
3  0.9895787 0.6488706
4  0.9842225 0.9363276
5  0.9788527 0.9609856
6  0.9733970 1.2101300
7  0.9678684 1.3880903
8  0.9622905 1.5003422
9  0.9566938 1.6098563
10 0.9509489 1.6125941
```

```
> plot(fit$predictions)
```




```
> plot(fit$predictions)
```

We can easily extract the 5 year cross-validated survival probabilities

```
> survival(fit$predictions, 5)[1:10]

        1         2         3         4         5         6         7         8
0.9352079 0.6817205 0.7814439 0.8895766 0.6752050 0.6943773 0.8033875 0.4208009
        9        10
0.6467053 0.9106909
```

5.4 profiling the cross-validated log likelihood

The functions `profL1` and `profL2` can be used to examine the effect of the parameters λ_1 and λ_2 on the cross-validated log likelihood. The `profL1` function can be used to vary λ_1 while keeping λ_2 fixed, vice versa for `profL2`.

The minimum and maximum values between which the cross-validated log likelihood is to be profiled can be given as *minlambda1* and *maxlambda1* or *minlambda2* and *maxlambda2*, respectively. The default value of *minlambda1* and *minlambda2* is at zero. The default value of *maxlambda1* is at the maximal value of λ_1, that is the smallest value that shrinks all regression coefficients to zero. There is no default for *maxlambda2*.

The number of steps between the minimal and maximal values can be given in the *steps* argument (default 100). These steps are equally spaced if the

argument *log* is FALSE or equally spaced on the log scale if the argument *log* is TRUE. Note that the default value of *log* differs between profL1 (FALSE) and profL2 (TRUE). If *log* is TRUE, *minlambda1* or *minlambda2* must be given by the user as the default value is not usable.

By default, the profiling is stopped prematurely when the cross-validated log likelihood drops below the cross-validated log likelihood of the null model with all penalized regression coefficients equal to zero. This is done because it avoids lengthy calculations at small values of λ when the models are most likely not interesting. The automatic stopping can be controlled using the option *minsteps* (default *steps*/2). The algorithm only considers early stopping after it has done at least *minsteps* steps. Setting minsteps equal to *steps* cancels the automatic stopping.

The functions profL1 and profL2 return a named list with the same elements as returned by cvl, but each of cvl, predictions, fullfit is now a *vector* or a *list* (as appropriate) as multiple cross-validated likelihoods were calculated. An additional vector lambda is returned which lists the values of λ_1 or λ_2 at which the cross-validated likelihood was calculated.

The allocation of the subjects into cross-validation folds is done only once, so that all cross-validated likelihoods are calculated using the same allocation. This makes the cross-validated log likelihoods more comparable. As in cvl the allocation is returned in fold.

It is also possible in these functions to set fold = 1. This will cause no cross-validation to be performed, but will let only the full data fits be calculated. This can be used in a similar way to the use of the penalized function with its *steps* argument, only with more flexibility.

The profiles can be plotted using the output of profL1 and profL2 or directly using the *plot* arguments of these functions.

```
> fit1 <- profL1(Surv(time,event), nki70[,50:70],fold=10, plot=TRUE)
> fit2 <- profL2(Surv(time,event), nki70[,50:70],fold=fit1$fold,
      minl = 0.01, maxl = 1000)
> plot(fit2$lambda, fit2$cvl, type="l", log="x")
```

The plotpath function can again be used to visualize the effect of the tuning parameter on the regression coefficients.

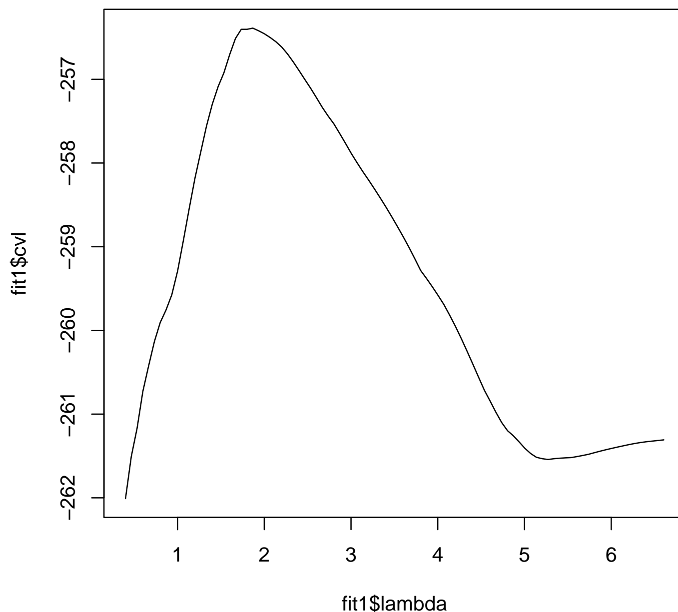
```
> plotpath(fit2$fullfit, log="x")
```

5.5 optimizing the cross-validated likelihood

Often we are not interested in the whole profile of the cross-validated likelihood, but only in the optimum. The functions optL1 and optL2 can be used to find the optimal value of λ_1 or λ_2.

The algorithm used for the optimization is the Brent algorithm for minimization without derivatives (Brent, 1973, see also help(optimize)). When using this algorithm, it is important to realize that this algorithm is guaranteed to work only for unimodal functions and that it may converge to a local maximum. This is especially relevant for L1 optimization, as the cross-validated likelihood as a function of λ_1 very often has several local maxima. It is recommended only to use optL1 in combination with profL1 to prevent convergence to the wrong

```
> plot(fit1$lambda, fit1$cvl, type="l")
```



optimum. The cross-validated likelihood as a function of λ_2, on the other hand, is far better behaved and practically never has local maxima. The function `optL2` can safely be used even without combining it with `profL2`.

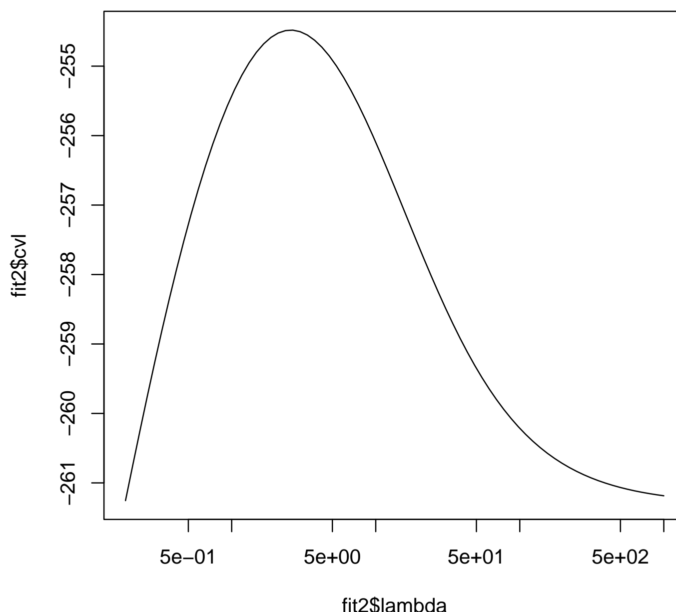
The functions `optL1` and `optL2` take the same arguments as `cvl`, and some additional ones.

The arguments *minlambda1* and *maxlambda1*, and *minlambda2* and *maxlambda2* can be used to specify the range between which the cross-validated log likelihood is to be optimized. Both arguments can be left out in both functions, but supplying them can improve convergence speed. In `optL1`, the parameter range can be use to ensure that the function converges to the right maximum. In `optL2` the user can also supply only one of *minlambda2* and *maxlambda2* to give the algorithm advance information of the order of magnitude of λ_2. In this case, the algorithm will search for an optimum around *minlambda2* or *maxlambda2*.

The functions `optL1` and `optL2` return a named list just as `cvl`, with an additional element `lambda` which returns the optimum found. The returned `cvl`, `predictions`, `fullfit` all relate to the optimal λ found.

```
> opt1 <- optL1(Surv(time,event), nki70[,50:70], fold=fit1$fold)

> opt1$lambda

[1] 1.865688

> opt1$cvl
```

```
> plot(fit2$lambda, fit2$cvl, type="l", log="x")
```




```
[1] -256.3876
```

```
> opt2 <- optL2(Surv(time,event), nki70[,50:70], fold=fit2$fold)
```

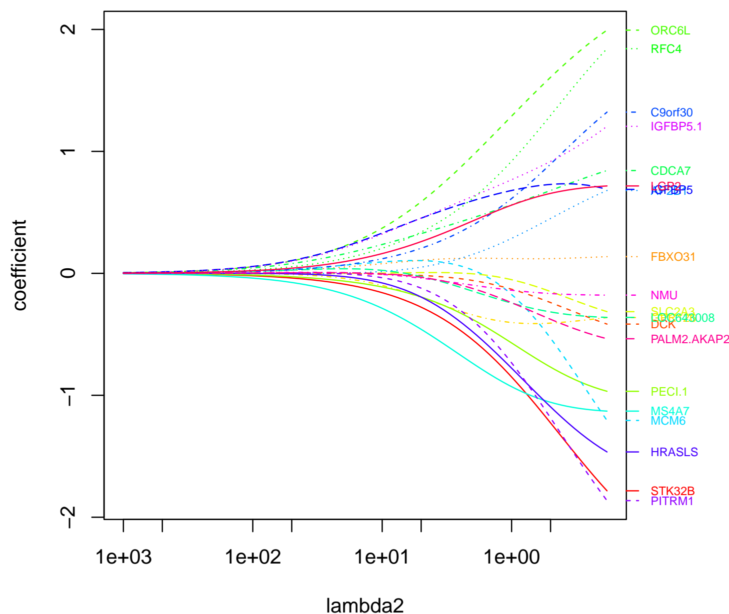
6 A note on standard errors and confidence intervals

It is a very natural question to ask for standard errors of regression coefficients or other estimated quantities. In principle such standard errors can easily be calculated, e.g. using the bootstrap.

Still, this package deliberately does not provide them. The reason for this is that standard errors are not very meaningful for strongly biased estimates such as arise from penalized estimation methods. Penalized estimation is a procedure that reduces the variance of estimators by introducing substantial bias. The bias of each estimator is therefore a major component of its mean squared error, whereas its variance may contribute only a small part.

Unfortunately, in most applications of penalized regression it is impossible to obtain a sufficiently precise estimate of the bias. Any bootstrap-based calculations can only give an assessment of the variance of the estimates. Reliable estimates of the bias are only available if reliable unbiased estimates are available, which is typically not the case in situations in which penalized estimates are used.

```
> plotpath(fit2$fullfit, log="x")
```



Reporting a standard error of a penalized estimate therefore tells only part of the story. It can give a mistaken impression of great precision, completely ignoring the inaccuracy caused by the bias. It is certainly a mistake to make confidence statements that are only based on an assessment of the variance of the estimates, such as bootstrap-based confidence intervals do.

Reliable confidence intervals around the penalized estimates can be obtained in the case of low dimensional models using the standard generalized linear model theory as implemented in `lm`, `glm` and `coxph`. Methods for constructing reliable confidence intervals in the high-dimensional situation are, to my knowledge, not available.

References

Brent, R. P. (1973). *Algorithms for Minimization without Derivatives.* Englewood Cliffs: Prentice-Hall.

Eilers, P., J. Boer, G. van Ommen, and J. C. van Houwelingen (2001). Classification of microarray data with penalized logistic regression. In M. L. Bittner, Y. Chen, A. N. Dorsel, and E. R. Dougherty (Eds.), *Proceedings of SPIE*, Volume 4266, pp. 187–198.

Goeman, J. J. (2010). L1 penalized estimation in the cox proportional hazards model. *Biometrical Journal 52*(1), 70–84.

Hoerl, A. E. and R. W. Kennard (1970). Ridge regression: biased estimation for nonorthogonal problems. *Technometrics 12*(1), 55–67.

Le Cessie, S. and J. C. van Houwelingen (1992). Ridge estimators in logistic regression. *Applied Statistics 41*(1), 191–201.

Park, M. Y. and T. Hastie (2007). l_1-regularized path algorithm for generailized linear models. *Journal of the Royal Statistical Society, Series B 69*(4), 659–677.

Tibshirani, R. (1996). Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society Series B-Methodological 58*(1), 267–288.

Tibshirani, R. (1997). The LASSO method for variable selection in the Cox model. *Statistics in Medicine 16*(4), 385–395.

Tibshirani, R., S. Rosset, et al. (2005). Sparsity and smoothness via the fused lasso. *Journal of the Roral Statistical Society 67*, 91–108.

Tibshirani, R. and P. Wang (2007). Spatial smoothing and hot spot detection for cgh data using the fused lasso. *Biostatistics Journal 9*, 18–29.

Van de Vijver, M. J., Y. D. He, L. J. van 't Veer, H. Dai, A. A. M. Hart, D. W. Voskuil, G. J. Schreiber, J. L. Peterse, C. Roberts, M. J. Marton, M. Parrish, D. Atsma, A. Witteveen, A. Glas, L. Delahaye, T. van der Velde, H. Bartelink, S. Rodenhuis, E. T. Rutgers, S. H. Friend, and R. Bernards (2002). A gene-expression signature as a predictor of survival in breast cancer. *New England Journal of Medicine 347*(25), 1999–2009.

Van Houwelingen, J. C., T. Bruinsma, A. A. M. Hart, L. J. van 't Veer, and L. F. A. Wessels (2006). Cross-validated Cox regression on microarray gene expression data. *Statistics in Medicine 25*(18), 3201–3216.

Van 't Veer, L. J., H. Y. Dai, M. J. van de Vijver, Y. D. D. He, A. A. M. Hart, M. Mao, H. L. Peterse, K. van der Kooy, M. J. Marton, A. T. Witteveen, G. J. Schreiber, R. M. Kerkhoven, C. Roberts, P. S. Linsley, R. Bernards, and S. H. Friend (2002). Gene expression profiling predicts clinical outcome of breast cancer. *Nature 415*(6871), 530–536.

Verweij, P. J. M. and H. C. Van Houwelingen (1993). Cross-validation in survival analysis. *Statistics in Medicine 12*(24), 2305–2314.

Verweij, P. J. M. and H. C. Van Houwelingen (1994). Penalized likelihood in cox regression. *Statistics in Medicine 13*(23-24), 2427–2436.

Zou, H. and T. Hastie (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B-Statistical Methodology 67*, 301–320.