L1 and L2 Penalized Regression Models

Jelle Goeman

November 16, 2007

Contents

1 Introduction

This short note explains the use of the *penalized* package. The package is designed for penalized estimation in generalized linear models.

The supported models at this moment are linear regression, logistic 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). 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 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 that used in this package will be documented in a forthcoming paper. It has been combined with ideas from Eilers et al. (2001) and Van Houwelingen et al. (2006) for efficient L2 penalized estimation.

2 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. Two main input types are allowed: one using *formula* objects, one using matrices.

2.1 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)
> 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.

2.2 the penalized function

To fit a model to predict survival (`Surv(time,event)`) with the two markers "DIAPH3" and "NUSAP1" at $\lambda_1 = 0$ and $\lambda_2 = 1$, we can say (all are equivalent)

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

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

```
> attach(nki70)
```

2.3 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 3

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*.

2.4 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
-0.1299336  0.7104811 -0.3517060 -0.2083512 -0.4264021 -0.3621108  0.7464918
       135        136        137
-0.6172103  0.7367359 -0.4470460
```

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

      125       127       128       129       130       132       134       135
0.4023261 1.0605204 0.8671254 0.6451380 1.3203100 1.1783128 0.7849620 1.3615191
      136       137
1.2242175 0.5909803

> basesurv(fit)

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

See `help(penfit)` for more information on *penfit* objects and Section 3.2 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")

      DIAPH3       NUSAP1
-0.003347245  1.610876235
```

2.5 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 = FALSE`. 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
-0.003347245  1.610876235

> coefficients(fit, standardize = TRUE)

       DIAPH3        NUSAP1
-0.0007849407  0.4300408953

> weights(fit)

   DIAPH3    NUSAP1
0.2345035 0.2669609
```

2.6 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.

In rare cases each covariate may have to penalized in a different way, or some covariates have to be given an L2 penalty and others an L1 penalty. In those cases, the arguments *lambda1* and *lambda2* may be supplied as vectors of the same length as the number of covariates in the function argument *penalized*.

2.7 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.

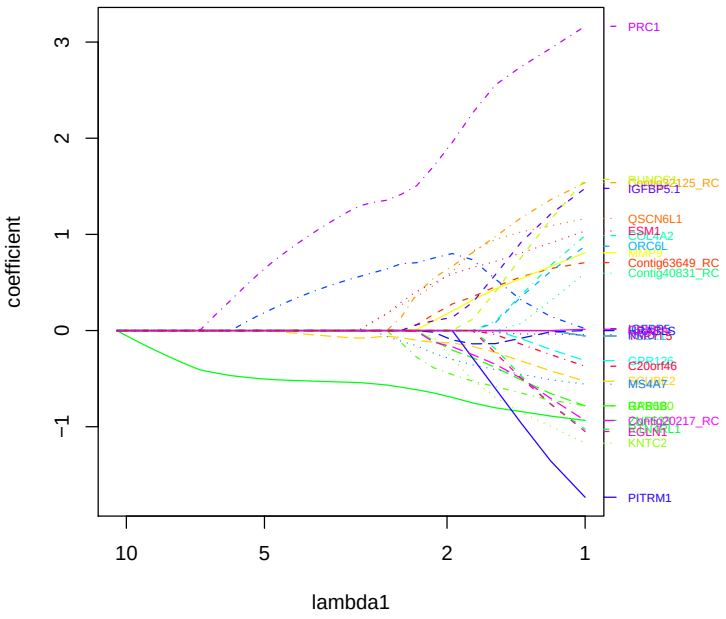
To override the automatic choice of contrasts, use `C` (package *stats*).

2.8 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*.

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.

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



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")
```

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 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.

3.1 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.

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] -258.5888

> 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)
```

3.2 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)
```

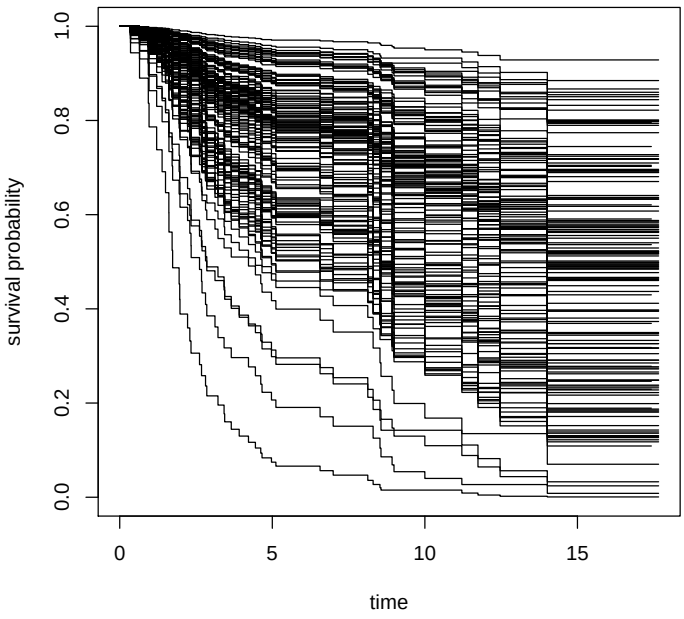
3.3 profiling the cross-validated log likelihood

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

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



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.

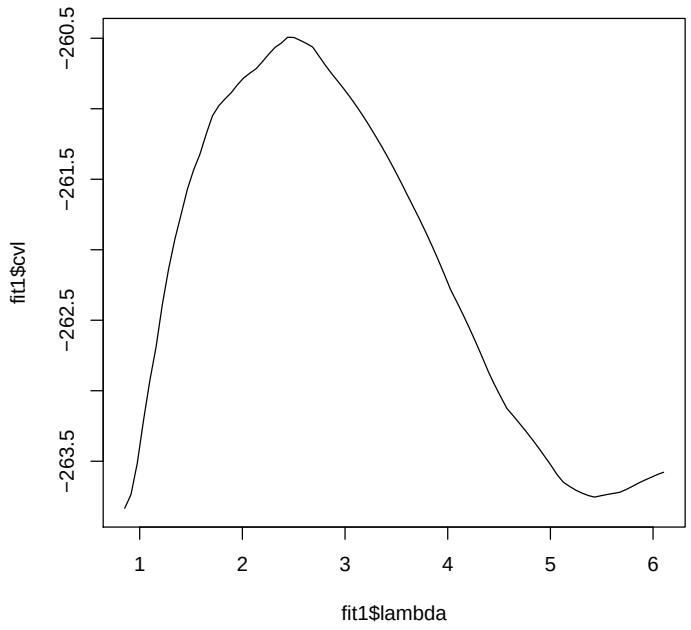
By default, the profiling is stopped prematurely when the cross-validated log likelihood drops below the cross-validated log likelihood of the null model with all penalized regression coefficients equal to zero. This is done because it avoids lengthy calculations at small values of λ when the models are most likely not interesting. The automatic stopping can be controlled using the option *minsteps* (default *steps*/4). The algorithm only considers stopping prematurely after it has done at least *minsteps* steps. Setting `minsteps=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

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



can be used in a similar way to the use of the `penalized` function with its *steps* argument, only with more flexibility.

```
> fit1 <- profL1(Surv(time, event), nki70[, 50:70], fold = 10)
> plot(fit1$lambda, fit1$cvl, type = "l")
> 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.

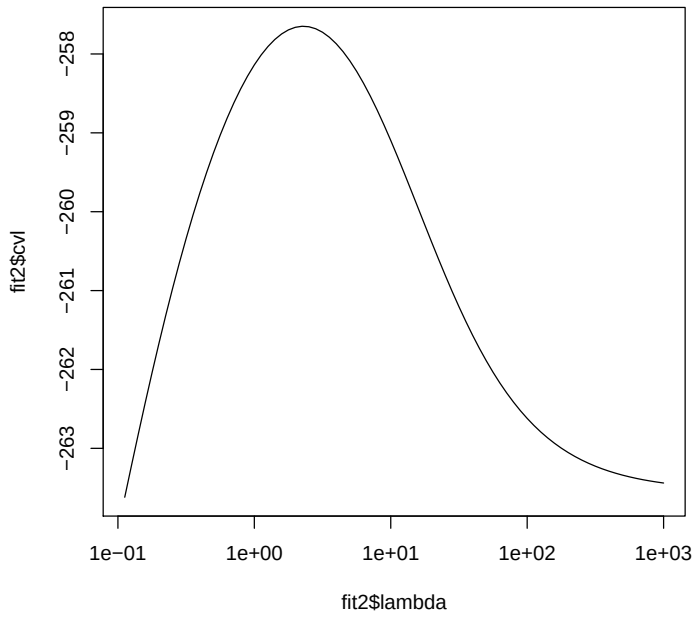
```
> plotpath(fit2$fullfit, log = "x")
```

3.4 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.

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



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 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.

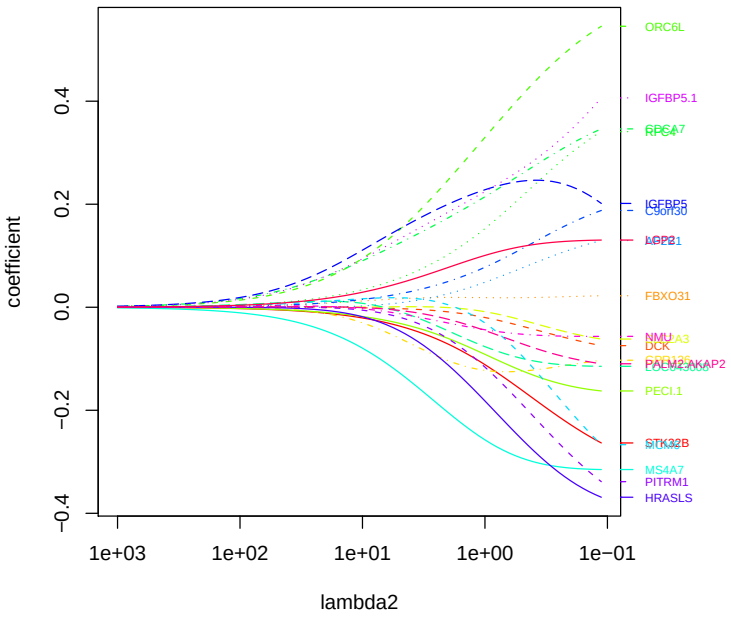
The arguments *minlambda1* and *maxlambda1*, and *minlambda2* and *maxlambda2* can be used to specify the range between which the cross-validated log likelihood is to be optimized. Both arguments can be left out in both functions, but supplying them can improve convergence speed. In `optL1`, the parameter 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
```

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




```
[1] 2.463987
```

```
> opt1$cvl
```

```
[1] -260.4834
```

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

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.

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.

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.

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.