Assignment 6: Smoothing and regression splines

Ian Wallgren, Wanchang Zhang, Lavinia Hriscu, Victor Jimenez

In this assignment we will use splines to obtain a smooth interpolator between some variables of the `bikes.Washington` dataset, which contains information on the bike-sharing rental service in Washington.

```r
load("bikes.Washington.RData")
attach(bikes)
names(bikes)
```

```
## [1] "instant"    "yr"         "dayyr"      "weekday"    "workingday"
## [6] "temp"       "hum"        "windspeed"  "cnt"
```

Spline smoothing is another way of obtaining a non-parametric estimator of the regression function $m(x)$, in this case by minimizing the square error and adding a term penalizing the lack of smoothness of the estimator:

$$\min_{\tilde{m}\in\mathcal{M}} \left\{ \sum_{i=1}^{n} (y_i - \tilde{m}(x_i))^2 + \phi(\tilde{m}) \right\}$$

where $\mathcal{M} \subseteq \mathcal{C}^p$ is a subclass of functions having p continuous derivatives and $\phi(\tilde{m}) : \mathcal{C}^p \longrightarrow \mathbb{R}$ is a functional penalizing the lack of smoothness of $\tilde{m}$. If data are in the interval $[a, b] \subseteq \mathbb{R}$, we choose $\mathcal{M} = W_2^2[a, b]$ the Sobolev space in the interval, whose functions are square-integrable and with square-integrable second derivative in the interval. Then the penalty term in the optimization problem then becomes:

$$\phi(\tilde{m}) = \lambda \int_a^b (\tilde{m}''(x))^2 dx$$

This is intuitive, as the second derivative is associated with the smoothness of the function, and high absolute values indicate extreme concavity or convexity in the interval. By adding this penalization term, we obtain a regularizer for our problem that we can tune with the penalty parameter λ, and that allow us to control the solution, which is unique, and corresponds to a cubic spline with knots $x_1, \dots, x_n$. If $\lambda \longrightarrow 0$, we will have an interpolator, whereas if $\lambda \longrightarrow \infty$ we will obtain a straight line, as the second derivative of $\tilde{m}$ will be brought down to zero.

It can be proven that the solution of the optimization problem is a natural cubic spline $s(x)$ (natural for having $s^{(l+j)}(a) = s^{(l+j)}(b) = 0$ for $j = 0, 1, \dots, l-1$), and thus that the problem can be instead solved in the space of natural splines of order p, which is finite dimensional:

- Base of splines $\mathcal{S}_p = S[p, a = t_0, \dots, t_{k+1} = b]$ has dimension $p + k + 1$
- Base of natural splines $\mathcal{N}_p = N[p, a = t_0, \dots, t_{k+1} = b]$ has dimension k

where $k + 1$ is the number of intervals for the knots where the spline is defined. Thus, in the space of natural splines, the optimal spline can be expressed in $\mathcal{N}_p$ as $s(x) = \sum_{j=1}^{n} \alpha_j N_j(x) = N(x)^T \alpha$. Then:

$$\sum_{i=1}^{n} (y_i - s(x_i))^2 = \sum_{i=1}^{n} (y_i - N(x_i)^T \alpha)^2 = (Y - N_x\alpha)^T (Y - N_x\alpha)$$

$$s''(x) = \alpha^T N''(x) \Longrightarrow \int_a^b (s''(x))^2 = \alpha^T A \alpha$$

where $A = \int_a^b N''(x)(N''(x))^T$. Then the optimization results:

$$\min_{\alpha \in \mathbb{R}^n} \Psi(\alpha) = (Y - N_x\alpha)^T(Y - N_x\alpha) + \lambda\alpha^T A\alpha$$

which can be solved analytically so that the vector of fitted values can be expressed as:

$$\hat{Y} = N_x\hat{\alpha} = N_x(N_x^T N_x + \lambda A)^{-1}N_x^T Y = SY$$

where S is the smooth function. As we can see, the spline estimator of $m(x)$ is a linear smoother, and thus we can tune λ by CV (also efficient LOO-CV, as it is consistent) and estimate the residual variance.

In order to make the computation more efficient, a different basis for $\mathcal{S}_p$ is considered. Since $s(x) \in \mathcal{N}_p$ and $\mathcal{N}_p \subset \mathcal{S}p$, then solving the problem in $\mathcal{S}p$ with this new basis will yield the same result. The B-splines, as they are called the elements of this new basis, can be obtained by considering $2M$ auxiliary knots τ_i arbitrarily chosen, that must lay at both extremes of the current knots ($\tau_{1:M} \leq t_0$ and $t_{k+1} \geq \tau_{k+M+1:k+2M}$) and iteratively computing:

$$B_{j,m} = \frac{x - \tau_j}{\tau_{j+m-1} - \tau_j} B_{j,m-1} + \frac{\tau_{j+m} - x}{\tau_{j+m} - \tau_{j+1}} B_{j+1,m-1} \quad j = 1, ..., k + 2M - m$$

with $B_{j,1} = I_{[\tau_j, \tau_{j+1}]}$ when $m = 1$. For $m = M = 4$ the functions $B_{j,4}$ are a basis for the vector space of cubic splines with knots $t_1, \ldots, t_k$ defined in $[a, b]$, also called basis of cubic B-splines. The improvement with respect to the previous computation of the solution comes from the following property:

$$B_{j,4}(x) = 0 \iff x \notin [\tau_j, \tau_{j+4}]$$

That is, the elements of the basis that are far apart yield zero in the integral of the penalization term and shall not be considered for the computation in that interval, thus sparing a lot of computation.

In this new basis the optimization problem has the following form:

$$\min_{\beta \in \mathbb{R}^{n+4}} \Psi(\beta) = (Y - B_x\beta)^T(Y - B_x\beta) + \lambda\beta^T D\beta$$

where $D = \int_a^b B_i''(x)B_j''(x)dx$ and $B_x \in \mathbb{R}^{n\times(n+4)}$. Now the solution is:

$$\hat{Y} = B_x\hat{\beta} = B_x(B_x^T B_x + \lambda D)^{-1}B_x^T Y = S_B Y$$

Even if the expression is the same as before, now $B_x^T B_x$ and D are band matrices, with elements (i, j) equal to zero if $|i - j| > 4$, meaning that the required matrix inversion can be easily computed by means of Cholesky decomposition.

In order to perform regression with splines, both k (the number of knots) and λ (the penalization parameter) can be tuned. The usual practice is to fix k to be large, so that the estimators have high flexibility, and consider λ as the only smoothing parameter to be adjusted via CV.

1. Cubic splines and GCV

In this first part of the assignment we will consider the non-parametric regression of `cnt` (the count of total rental bikes) as a function of `instant` (the row index). As it follows from the previous theoretical development, cubic regression splines will be used and the smoothing parameter λ will be chosen by GCV.

```r
x = bikes$instant
y = bikes$cnt
```

The function `smooth.spline` of the package `stats` will be used. The most important arguments of this function are the following:

- `x` (predictor variable) and `y` (target variable)
- `df`: desired equivalent number of degrees of freedom $\nu(\lambda)$
- `lambda`: smoothing parameter
- `cv`: if no smoothing parameter (or equivalent: `df` or `spar`) is provided, the most suitable one is computed via cross validation: `TRUE` for LOO-CV and `FALSE` for GCV.

a) Finding λ by GCV

First, the optimal value of the smoothing parameter λ will be determined by Generalized Cross-Validation. The function `smooth.spline` already provides a method for choosing λ_{GCV} if we do not provide a smoothing penalty:

```r
spline.fit = smooth.spline(x, y, cv = FALSE)
lambda.gcv = spline.fit$lambda
```

```
## [1] 1.005038e-07
```

b) Equivalent number of degrees of freedom

The equivalent number of degrees of freedom of the fit is the most intuitive smoothing parameter. For any linear smoother it can be computed as:

$$\nu(\lambda) = \text{Trace}(S(\lambda))$$

In this case, we can obtain it directly from the `smooth.spline` result:

```r
df.gcv = spline.fit$df
```

```
## [1] 93.34091
```

As we can see, it is a high value.

c) Number of knots used

As mentioned earlier, when adjusting for λ the number of knots is usually set to a high value, so to grant flexibility to the fit. We can obtain it from the element `knots`:

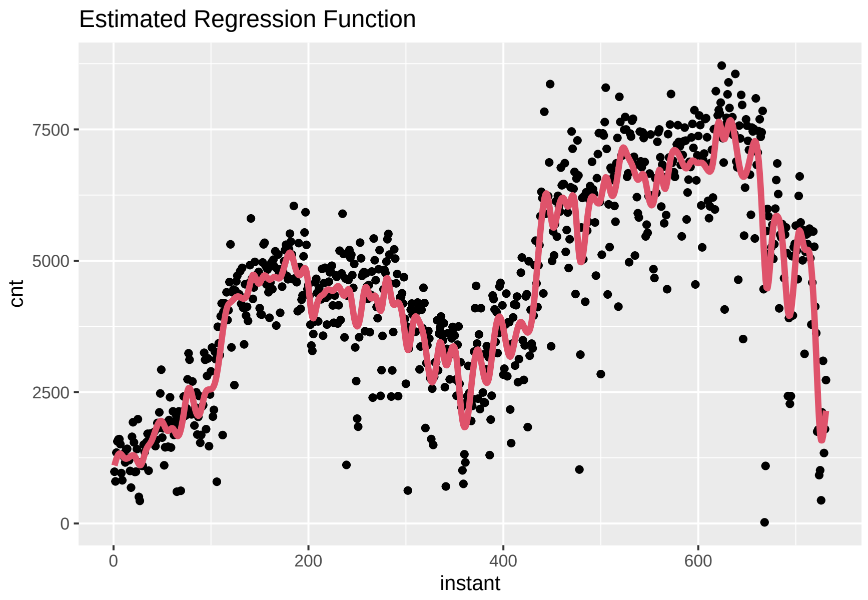
```r
k.knots = length(spline.fit$fit$knot)
```

```
## [1] 140
```

d) Scatterplot and $\tilde{m}(x)$

The predicted values for `cnt` according to the regression function can be obtained by means of the function `predict`. We will plot the results obtained.

```r
y.pred = predict(spline.fit, x)$y
```



e) Estimating with unpenalized regression splines

An alternative to solving the penalized optimization problem consists of fitting a generalized linear model using a cubic B-spline basis matrix as the regressors. Now k, the number of knots, is the smoothing parameter.

We will use the function `lm` and `bs` of the package `splines` with `n.knots = df.gcv - 4`. The function `bs` generates the basis matrix of cubic spline functions in the defined knots and the function `lm` will fit the generalized linear model with the splines as regressors.

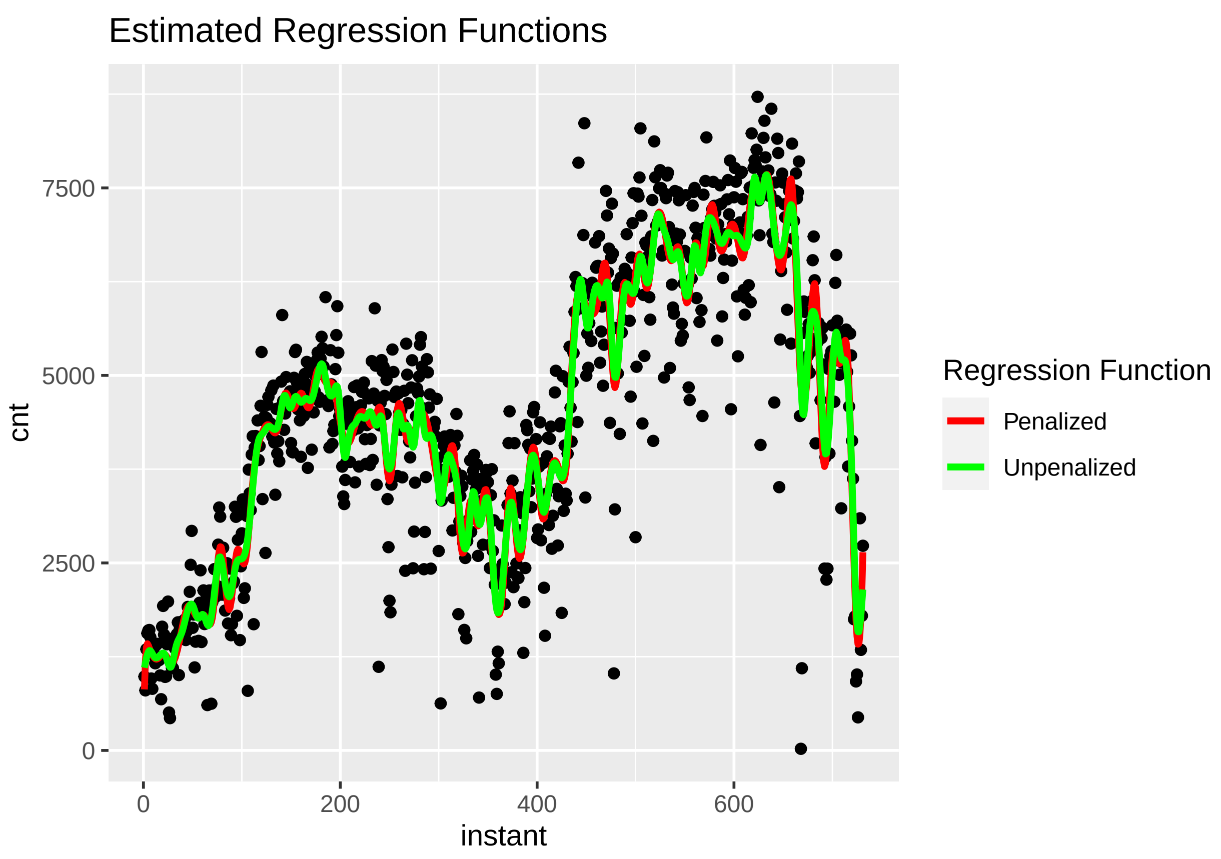
```
n.knots = df.gcv - 4
my.knots = quantile(x,((1:n.knots)-.5)/n.knots)
bspline.matrix = bs(x, knots = my.knots, degree = 3)
lm.fit = lm(y ~ bspline.matrix)
```

As we can see, a high number of effective degrees of freedom allow the regression line to adjust more accurately to the local variations of the data. In this case, however, a much lower value of degrees of freedom would probably have been better to interpret the variations of `cnt` as function of `instant` in a more qualitative way.

f) Compare the two regression estimations

We can compare the results obtained in the prediction of `cnt` with both methods by plotting them in the same graph.

```
y.pred.nop = predict(lm.fit, newdata = data.frame(x))
```



2. Logistic regression with P-IRWLS

Splines regression can also be used to perform generalized nonparametric regression. As in the previous assignment, the response variable Y has a conditional distribution:

$$(Y|X = x) \sim f(y; m(x), \psi) = f(y; g^{-1}(\theta(x)), \psi); \quad \psi \in \mathbb{R}^p$$

where $m(x) = E(Y|X = x) \in C^2$ for which an invertible link function $g(\cdot)$ exists, such that $\theta(x) \in C^2$ is free of constraints. Now, we estimate $\theta(x)$ locally by maximizing the expected log-likelihood function and incorporating a lack-of-smoothness penalization term, so that:

$$\max_{\theta \in W_2^2[a,b]} \left\{ \sum_{i=1}^{n} \log(f(y_i|\theta(x_i)) + \lambda \int_a^b (\theta''(x))^2 dx \right\}$$

whose optimal solution $\hat{\theta}(x)$ is a natural cubic spline. The solution to this problem is not analytic and a possible numerical optimization algorithm to achieve it is P-IRWLS (penalized IRWLS). The function `logistic.IRWLS.splines` that performs P-IRWLS has been provided by the professor. The most important arguments of this function are:

- `x` (explanatory variable) and `y` (target variable, which now is binary)
- `x.new`: the new values of explanatory variable where we want to predict $\mathbb{P}(y = 1|x = x.new)$
- `df`: equivalent number of parameters

In this case we are interested in modelling `cnt.5000`, a binary aggregation of `cnt` that takes value 1 when `cnt` > 5000, as a function of `temp`. We will use `df=6` in this section.

```
cnt.5000 = as.numeric(cnt > 5000)
model = logistic.IRWLS.splines(temp, cnt.5000, df = 6)
```

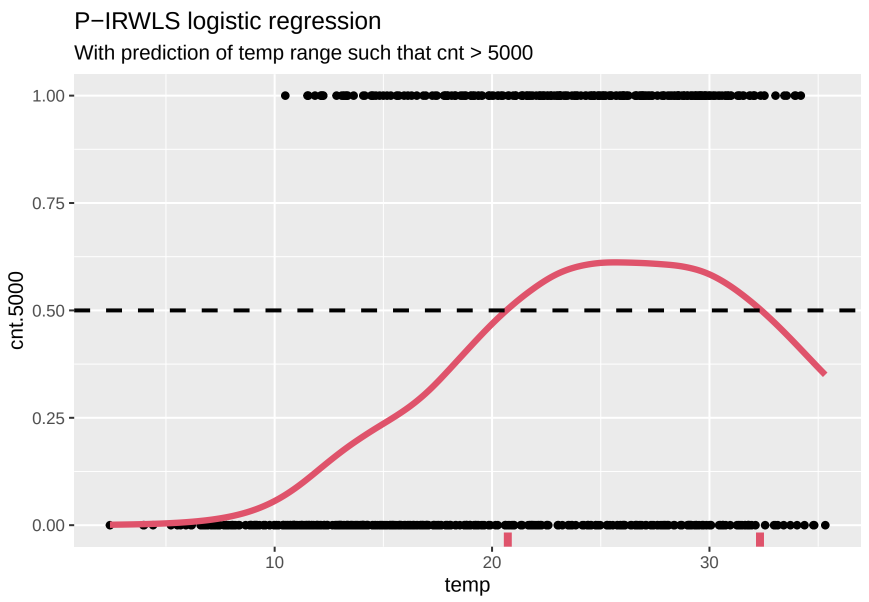
We now want to find the range of `temp` for which $\mathbb{P}(y=1|x=x.new)$. We will use the argument `x.new` to pass a a thinner grid of points and thus obtain a better prediction:

```
temp.grid = seq(range(temp)[1],range(temp)[2],.1)
model.thin = logistic.IRWLS.splines(temp, cnt.5000, x.new=temp.grid, df = 6)
```

The range of temperatures that we are seeking will correspond to the range of positions of the fitted regression with value larger than 0,5.

```
temp.range = range(temp.grid[model.thin$predicted.values > 0.5])
```

```
## [1] 20.72435 32.32435
```



b) Choosing `df` by k-fold log-likelihood CV

Finally, we will repeat the regression choosing the optimal `df` by log-likelihood cross validation; that is, the value that provides the fitting that maximizes the likelihood of observing the true target values. The expression of the likelihood corresponds to the logistic model and was already used in the previous assignment.

$$df_{k-CV} = \arg\max_{df} \frac{1}{k} \sum_{i=1}^{k} l^{(-i)} = \arg\max_{df} \frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{n_k} y_j^{(k)} \log(p_j^{(k)}) + (1-y_j^{(k)}) \log(1-p_j^{(k)})$$

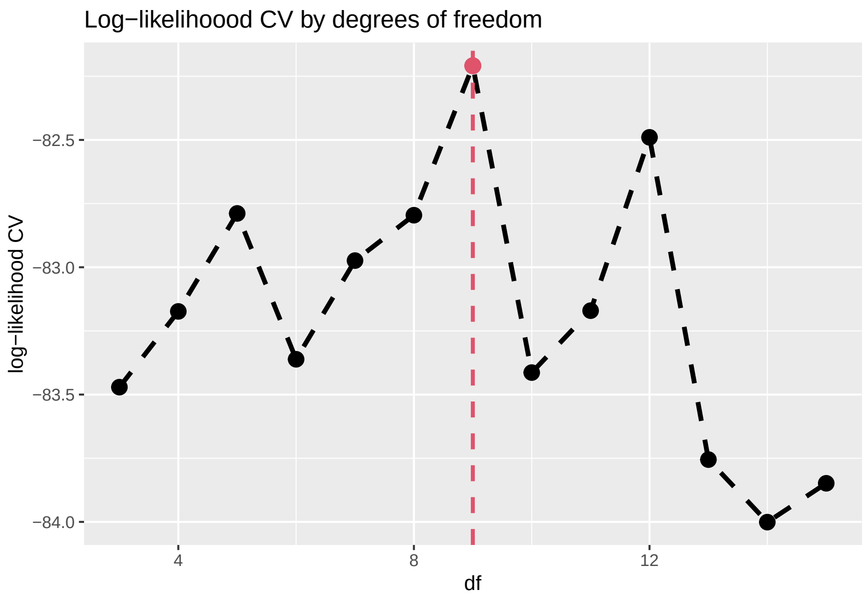
The function implementing the previous equation is the following:

```
log.likelihood.kfoldCV = function(x,y,k,df.vals){
  log.likelihood.cv = numeric(length(df.vals))
  for (i in 1:length(df.vals)){
    log.likelihood = numeric(k)
    kfolds = caret::createFolds(y, k = k)
    for (j in 1:k){
```

```r
      train.ind = unlist(kfolds[-j])
      test.ind = kfolds[[j]]

      x.train = x[train.ind]
      y.train = y[train.ind]
      x.test = x[test.ind]
      y.test = y[test.ind]

      model.fold = logistic.IRWLS.splines(x.train, y.train, x.new = x.test, df = df.vals[i])
      p.pred = model.fold$predicted.values
      log.likelihood[j] = sum(y.test*log(p.pred) + (1-y.test)*log(1-p.pred))
    }
    log.likelihood.cv[i] = mean(log.likelihood)
  }

  return(log.likelihood.cv)
}
```

We will perform the k-fold CV for `k=5` and considering the possible values `df=seq(3,15)`.

```r
df.vals = seq(3,15)
log.likelihood.cv = log.likelihood.kfoldCV(temp, cnt.5000,5,df.vals)
df.kcv = df.vals[which.max(log.likelihood.cv)]
```

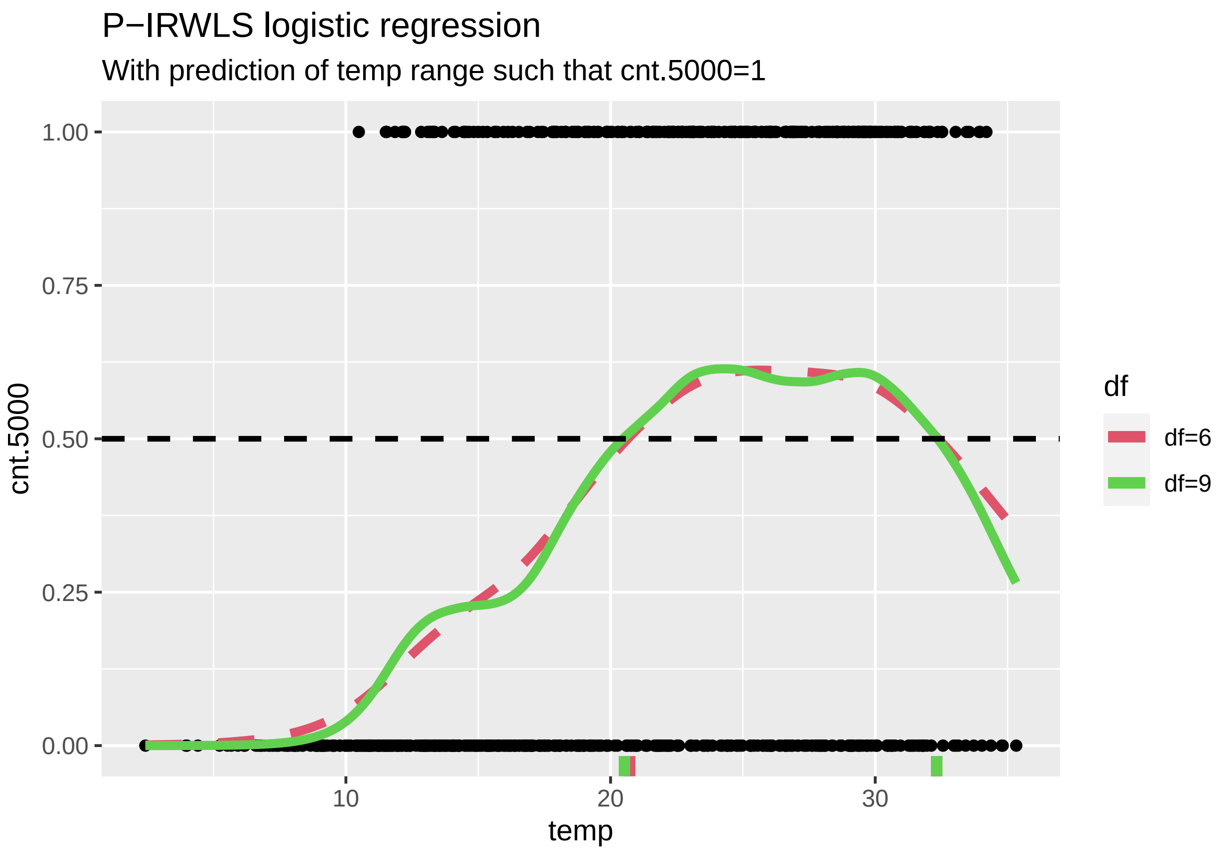


The number of effective degrees of freedom for this model maximizing the likelihood by k-fold CV is:

```
## [1] 9
```

We will perform the regression again with this new value and compare the results to the ones obtained in the previous section:

```
model.thin.kcv = logistic.IRWLS.splines(temp, cnt.5000, x.new=temp.grid, df = df.kcv)
temp.range.kcv = range(temp.grid[model.thin.kcv$predicted.values > 0.5])
```



We see clearly how the degrees of freedom affect directly the convexity of the curve, as now we can identify visually a more bumped line. The temperature range expanded slightly towards negative temperatures:

```
## [1] 20.52435 32.32435
```