1 Loss Function and its Derivatives: Smoothing Splines

In this Section, we write down the math for estimation via splines. We will use the symbol $\bigoplus$ to denote the log-space summation operator: $\bigoplus_{i=1}^{N} \log(a_i) = \log\left(\sum_{i=1}^{N} a_i\right)$. Writing expressions using this notation will allow us to obtain numerically stable computational expressions.

Let $m \in \{1, \dots, M\}$ index component predictive models and $i \in \{1, \dots, N\}$ index prediction cases in the training data. For example, in a seasonal time series prediction context i may index times at which we make predictions for seasonal quantities, or combination of a time at which we make a prediction and the prediction horizon for predictions at individual weeks. Let $f_m(y_i|\mathbf{x}_i)$ denote the predictive density from model m for the value of the random variable Y_i. The $\mathbf{x}_i$ is a vector observed covariates which may be used by any of the component models as conditioning variables in forming the predictive distribution, and is also used in calculating the component model weights. Note that the component models and computation of the component model weights may use a proper subset of the variables in $\mathbf{x}_i$.

The combined predictive density for case i is

$$f(y_i|\mathbf{x}_i) = \sum_{m=1}^{M} \pi_m(\mathbf{x}_i) f_m(y_i|\mathbf{x}_i), \text{ where} \tag{1}$$

$$\pi_m(\mathbf{x}_i) = \frac{\exp\{\sum_{j=1}^{J} s_{mj}(x_{ij})\}}{\exp\{\sum_{m'=1}^{M}\sum_{j=1}^{J} s_{m'j}(x_{ij})\}} \tag{2}$$

$$= \frac{\exp(\sum_{j=1}^{J} B'_{ij}\boldsymbol{\theta}_{mj})}{\exp(\sum_{m'=1}^{M}\sum_{j=1}^{J} B'_{ij}\boldsymbol{\theta}_{m'j})} \tag{3}$$

In Equation (22) the $\pi_m(\mathbf{x}_i)$ are the model weights, which we regard as functions of $\mathbf{x}_i$. These weights must be non-negative and sum to 1 across m. We ensure that these constraints are met by parameterizing the $\pi_m(\mathbf{x}_i)$ in terms of the softmax transformation of a separate smoothing spline for each covariate in Equation (2). Equation (3) represents the spline function s_{mj} for the mth model and jth covariate evaluated at x_{ij} as the product of a vector B'_{ij} of B-spline basis functions evaluated at x_{ij} and a corresponding vector $\boldsymbol{\theta}_{mj}$ of coefficients.

As usual with smoothing splines, we proceed with estimation of the coefficient vectors $\boldsymbol{\theta}_{mj}$ under a penalty on the integrated second derivative of the spline functions. We use a penalty λ_j for each covariate j that is shared across all values of m, so that λ_j effectively controls the smoothness of the weighting function as the jth covariate varies.

Putting these pieces together, the penalized loss function used in estimation is

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log\{f(y_i|\mathbf{x}_i)\} - \frac{1}{2}\sum_j \lambda_j \sum_m \int \{s''(x)\}\,\mathrm{d}x \tag{4}$$

$$= \sum_{i=1}^{N} \log\left\{\sum_{m=1}^{M} \pi_{mi} f_m(y_i|\mathbf{x}_i)\right\} - \frac{1}{2}\sum_j \lambda_j \sum_m \boldsymbol{\theta}'_{mj}\Omega_j\boldsymbol{\theta}_{mj} \tag{5}$$

$$= \sum_{i=1}^{N} \log\left\{\sum_{m=1}^{M} \frac{\exp(\sum_{j=1}^{J} B'_{ij}\boldsymbol{\theta}_{mj})}{\sum_{m'=1}^{M}\exp(\sum_{j=1}^{J} B'_{ij}\boldsymbol{\theta}_{m'j})} f_m(y_i|\mathbf{x}_i)\right\} - \frac{1}{2}\sum_j \lambda_j \sum_m \boldsymbol{\theta}'_{mj}\Omega_j\boldsymbol{\theta}_{mj} \tag{6}$$

In order to estimate $\boldsymbol{\theta}$, we will use the backfitting algorithm that is standard for estimating generalized additive models. This can alternatively be conceptualized as optimizing the penalized loss via a block update algorithm, where we update the parameters $\boldsymbol{\theta}_{mj}$ for one spline at a time. To do this, we will use a Newton update in each step:

$$\boldsymbol{\theta}_{mj}^{\text{new}} = \boldsymbol{\theta}_{mj}^{\text{old}} - \left\{\frac{\partial^2 \ell(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}_{mj}\partial\boldsymbol{\theta}'_{mj}}\right\}^{-1} \frac{\partial \ell(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}_{mj}}, \tag{7}$$

where the partial derivatives are taken at $\boldsymbol{\theta}_{mj}^{\text{old}}$.

To proceed, we must calculate the gradient and Hessian of the likelihood function with respect to a particular set of spline parameters which we will denote by $\boldsymbol{\theta}_{m^*j^*}$.

It will be helpful to obtain the preliminary results that

$$\begin{aligned}
\frac{\partial \pi_{m^*i}}{\partial \boldsymbol{\theta}_{m^*j^*}} &= \frac{\partial}{\partial \boldsymbol{\theta}_{m^*j^*}} \frac{\exp(\sum_{j=1}^J B_{ij}' \boldsymbol{\theta}_{m^*j})}{\exp(\sum_{m'=1}^M \sum_{j=1}^J B_{ij}' \boldsymbol{\theta}_{m'j})} \\
&= \frac{\exp(\sum_{j=1}^J B_{ij}' \boldsymbol{\theta}_{m^*j}) \left\{ \sum_{m'=1}^M \exp(\sum_{j=1}^J B_{ij}' \boldsymbol{\theta}_{m'j}) \right\} B_{ij^*} - \left\{ \exp(\sum_{j=1}^J B_{ij}' \boldsymbol{\theta}_{m^*j}) \right\}^2 B_{ij^*}}{\left\{ \sum_{m'=1}^M \exp(\sum_{j=1}^J B_{ij}' \boldsymbol{\theta}_{m'j}) \right\}^2} \\
&= (\pi_{m^*i} - \pi_{m^*i}^2) B_{ij^*}, \qquad (8)
\end{aligned}$$

while for $m \neq m^*$

$$\begin{aligned}
\frac{\partial \pi_{mi}}{\partial \boldsymbol{\theta}_{m^*j^*}} &= \frac{\partial}{\partial \boldsymbol{\theta}_{m^*j^*}} \frac{\exp(\sum_{j=1}^J B_{ij}' \boldsymbol{\theta}_{mj})}{\exp(\sum_{m'=1}^M \sum_{j=1}^J B_{ij}' \boldsymbol{\theta}_{m'j})} \\
&= \frac{-\exp(\sum_{j=1}^J B_{ij}' \boldsymbol{\theta}_{mj}) \exp(\sum_{j=1}^J B_{ij}' \boldsymbol{\theta}_{m^*j}) B_{ij^*}}{\left\{ \sum_{m'=1}^M \exp(\sum_{j=1}^J B_{ij}' \boldsymbol{\theta}_{m'j}) \right\}^2} \\
&= (-\pi_{m^*i} \pi_{mi}) B_{ij^*}. \qquad (9)
\end{aligned}$$

We now calculate the gradient as

$$\begin{aligned}
\frac{\partial \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_{m^*j^*}} &= \frac{\partial}{\partial \boldsymbol{\theta}_{m^*j^*}} \left[\sum_{i=1}^N \log \left\{ \sum_{m=1}^M \pi_{mi} f_m(y_i|\mathbf{x}_i) \right\} - \frac{1}{2} \sum_j \lambda_j \sum_m \boldsymbol{\theta}_{mj}' \Omega_j \boldsymbol{\theta}_{mj} \right] \\
&= \sum_{i=1}^N \frac{1}{f(y_i|x_i)} \left\{ \pi_{m^*i} f_{m^*}(y_i|x_i) - \sum_{m=1}^M \pi_{mi} \pi_{m^*i} f_m(y_i|x_i) \right\} B_{ij^*} - \lambda_{j^*} \Omega_{j^*} \boldsymbol{\theta}_{m^*j^*} \\
&= \sum_{i=1}^N \pi_{m^*i} \left\{ \frac{f_{m^*}(y_i|x_i)}{f(y_i|x_i)} - 1 \right\} B_{ij^*} - \lambda_{j^*} \Omega_{j^*} \boldsymbol{\theta}_{m^*j^*} \\
&= B_{j^*}' \boldsymbol{g} - \lambda_{j^*} \Omega_{j^*} \boldsymbol{\theta}_{m^*j^*}. \qquad (10)
\end{aligned}$$

In Equation (10), B_{j^*} is an $N \times D_{j^*}$ matrix with the D_{j^*} spline basis functions for covariate j^* evaluated at each observation $i = 1, \dots, N$. $\boldsymbol{g}$ is a column vector of length N where entry i is $\pi_{m^*i} \left\{ \frac{f_{m^*}(y_i|x_i)}{f(y_i|x_i)} - 1 \right\}$. This quantity has the interpretation that the contribution to the gradient from observation i is positive if the conditional density for $Y_i|X_i$ from model m^* was larger at the observed outcome y_i than the combined conditional density after the previous iteration.

We can calculate the Hessian as

$$\frac{\partial^2 \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_{m^*j^*} \partial \boldsymbol{\theta}'_{m^*j^*}} = \frac{\partial \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}'_{m^*j^*}} \left[\sum_{i=1}^N B_{ij^*} \pi_{m^*i} \left\{ \frac{f_{m^*}(y_i|x_i)}{f(y_i|x_i)} - 1 \right\} - \lambda_{j^*} \Omega_{j^*} \boldsymbol{\theta}_{m^*j^*} \right] \tag{11}$$

$$= \frac{\partial \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}'_{m^*j^*}} \left[\sum_{i=1}^N B_{ij^*} \left\{ \frac{\pi_{m^*i} f_{m^*}(y_i|x_i)}{\sum_{m'=1}^M \pi_{m'i} f_{m'}(y_i|x_i)} - \pi_{m^*i} \right\} - \lambda_{j^*} \Omega_{j^*} \boldsymbol{\theta}_{m^*j^*} \right] \tag{12}$$

$$= \sum_{i=1}^N B_{ij^*} \left\{ \frac{(\pi_{m^*i} - \pi^2_{m^*i}) f_{m^*}(y_i|x_i) \left\{ \sum_{m'=1}^M \pi_{m'i} f_{m'}(y_i|x_i) \right\} B'_{ij^*}}{\left\{ \sum_{m'=1}^M \pi_{m'i} f_{m'}(y_i|x_i) \right\}^2} \right. \tag{13}$$

$$- \frac{\pi^2_{m^*i} f_{m^*}(y_i|x_i) \left\{ f_{m^*}(y_i|x_i) - \sum_{m'=1}^M \pi_{m'i} f_{m'}(y_i|x_i) \right\} B'_{ij^*}}{\left\{ \sum_{m'=1}^M \pi_{m'i} f_{m'}(y_i|x_i) \right\}^2} \tag{14}$$

$$\left. - (\pi_{m^*i} - \pi^2_{m^*i}) B'_{ij^*} \right\} - \lambda_{j^*} \Omega_{j^*} \tag{15}$$

$$= \sum_{i=1}^N B_{ij^*} \left[\pi_{m^*i} \left\{ \frac{f_{m*}(y_i|x_i)}{f(y_i|x_i)} - 1 \right\} - \pi^2_{m^*i} \left\{ \left(\frac{f_{m*}(y_i|x_i)}{f(y_i|x_i)} \right)^2 - 1 \right\} \right] B'_{ij^*} - \lambda_{j^*} \Omega_{j^*} \tag{16}$$

$$= -B'_{j^*} W B_{j^*} - \lambda_{j^*} \Omega_{j^*}. \tag{17}$$

Here, $f(y_i|x_i)$ is the combined predictive density from the ensemble using the model weights obtained after the previous iteration of the parameter updating algorithm. W is an $N \times N$ diagonal matrix where the ith entry on the diagonal is $\left[\pi^2_{m^*i} \left\{ \left(\frac{f_{m*}(y_i|x_i)}{f(y_i|x_i)} \right)^2 - 1 \right\} - \pi_{m^*i} \left\{ \frac{f_{m*}(y_i|x_i)}{f(y_i|x_i)} - 1 \right\} \right]$. In general these values may be negative, and the Hessian is not guaranteed to be positive definite (particularly if λ_{j^*} is close to 0). In that case, the Newton update of Equation (7) is not appropriate since the update may not move in an ascent direction (cite Nocedal and Wright). In such cases, it is standard procedure to modify the Hessian to ensure that it is positive definite. In this case, this can be achieved by truncating the diagonal entries of W below at $\delta > 0$; we have used $\delta = 1$. This choice for δ is conservative and may reduce the convergence rate of the estimation procedure; we have not explored other possible values for δ.

Now returning to Equation (7), the updated parameters in each iteration of the Newton-Raphson optimization procedure are obtained as

$$\boldsymbol{\theta}^{\text{new}}_{mj} = \boldsymbol{\theta}^{\text{old}}_{mj} - \left\{ \frac{\partial^2 \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_{mj} \partial \boldsymbol{\theta}'_{mj}} \right\}^{-1} \frac{\partial \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_{mj}} \tag{18}$$

$$= \boldsymbol{\theta}^{\text{old}}_{mj} - \left\{ -B'_{j^*} W B_{j^*} - \lambda_{j^*} \Omega_{j^*} \right\}^{-1} \left\{ B'_{j^*} \boldsymbol{g} - \lambda_{j^*} \Omega_{j^*} \boldsymbol{\theta}^{\text{old}}_{m^*j^*} \right\} \tag{19}$$

$$= \left\{ B'_{j^*} W B_{j^*} + \lambda_{j^*} \Omega_{j^*} \right\}^{-1} \left\{ B'_{j^*} \boldsymbol{g} + B'_{j^*} W B_{j^*} \boldsymbol{\theta}^{\text{old}}_{m^*j^*} \right\} \tag{20}$$

$$= \left\{ B'_{j^*} W B_{j^*} + \lambda_{j^*} \Omega_{j^*} \right\}^{-1} B'_{j^*} W \left\{ W^{-1} \boldsymbol{g} + B_{j^*} \boldsymbol{\theta}^{\text{old}}_{m^*j^*} \right\} \tag{21}$$

In Equation (20), we have used the fact that $\boldsymbol{\theta}^{\text{old}}_{m^*j^*} = \left\{ B'_{j^*} W B_{j^*} + \lambda_{j^*} \Omega_{j^*} \right\}^{-1} \left\{ B'_{j^*} W B_{j^*} + \lambda_{j^*} \Omega_{j^*} \right\} \boldsymbol{\theta}^{\text{old}}_{m^*j^*}$, so $\boldsymbol{\theta}^{\text{old}}_{m^*j^*} - \left\{ B'_{j^*} W B_{j^*} + \lambda_{j^*} \Omega_{j^*} \right\}^{-1} \lambda_{j^*} \Omega_{j^*} \boldsymbol{\theta}^{\text{old}}_{m^*j^*} = \left\{ B'_{j^*} W B_{j^*} + \lambda_{j^*} \Omega_{j^*} \right\}^{-1} B'_{j^*} W B_{j^*} \boldsymbol{\theta}^{\text{old}}_{m^*j^*}$. The updated estimates $\boldsymbol{\theta}^{\text{new}}_{mj}$ from Equation (21) have the form of parameter estimates from a smoothing spline where the observations have weights specified by the elements of W and the "working response" vector is given by $W^{-1} \boldsymbol{g} + B_{j^*} \boldsymbol{\theta}^{\text{old}}_{m^*j^*}$. It is informative to observe that the second term in this working response is the fitted spline value after the previous parameter update step, and the first term specifies that the parameters should be updated so as to move the spline fit in the direction of the vector $\boldsymbol{g}$ of contributions to the gradient from each observation, scaled by the inverse of the observation weights in W.

2 Loss Function and its Derivatives: Boosting

In this Section, we write down the math. We will use the symbol $\bigoplus$ to denote the log-space summation operator: $\bigoplus_{i=1}^N \log(a_i) = \log\left(\sum_{i=1}^N a_i\right)$. Writing expressions using this notation will allow us to obtain numerically stable computational expressions.

Let $m \in \{1, \ldots, M\}$ index component predictive models and $i \in \{1, \ldots, N\}$ index prediction cases in the training data. For example, in a seasonal time series prediction context i may index times at which we make predictions for seasonal quantities, or combination of a time at which we make a prediction and the prediction horizon for predictions at individual weeks. Let $f_m(y_i|\mathbf{x}_i)$ denote the predictive density from model m for the value of the random variable Y_i. The $\mathbf{x}_i$ is a vector observed covariates which may be used by any of the component models as conditioning variables in forming the predictive distribution, and is also used in calculating the component model weights. Note that the component models and computation of the component model weights may use a proper subset of the variables in $\mathbf{x}_i$.

The combined predictive density for case i is

$$f(y_i|\mathbf{x}_i) = \sum_{m=1}^M \pi_m(\mathbf{x}_i) f_m(y_i|\mathbf{x}_i), \text{ where} \tag{22}$$

$$\pi_m(\mathbf{x}_i) = \frac{\exp\{\rho_m(\mathbf{x}_i)\}}{\sum_{m'=1}^M \exp\{\rho_{m'}(\mathbf{x}_i)\}} \tag{23}$$

In Equation (22) the $\pi_m(\mathbf{x}_i)$ are the model weights, which we regard as functions of $\mathbf{x}_i$. These weights must be non-negative and sum to 1 across m. We ensure that these constraints are met by parameterizing the $\pi_m(\mathbf{x}_i)$ in terms of the softmax transformation of real-valued functions $\rho_m(\mathbf{x}_i)$ in Equation (23). For notational brevity, we will suppress the expression of these quantities as functions of $\mathbf{x}_i$ and write $\rho_m(\mathbf{x}_i) = \rho_{mi}$ with $\boldsymbol{\rho} = (\rho_{11}, \ldots, \rho_{MN})$, and $\pi_m(\mathbf{x}_i) = \pi_{mi}$ with $\boldsymbol{\pi} = (\pi_{11}, \ldots, \pi_{MN})$.

Our goal is to estimate the functions ρ_{mi}. To do this, we require as inputs cross-validated estimates of $\log\{f_m(y_i|\mathbf{x}_i)\}$ for each component model m and case i in the training data; we denote these values by $\log\{f_m^{cv}(y_i|\mathbf{x}_i)\}$. We will focus on optimization of the log-score of the combined predictive distribution for now; we may consider other loss functions in the future. Considered as a function of the vector of values ρ_{mi} for each combination of m and i, this loss function is given by

$$\begin{aligned}
L(\boldsymbol{\rho}) &= \sum_{i=1}^N \log\{f(y_i|\mathbf{x}_i)\} & (24)\\
&= \sum_{i=1}^N \log\left\{\sum_{m=1}^M \pi_{mi} f_m(y_i|\mathbf{x}_i)\right\} & (25)\\
&= \sum_{i=1}^N \log\left\{\sum_{m=1}^M \frac{\exp(\rho_{mi})}{\sum_{m'=1}^M \exp(\rho_{m'i})} f_m(y_i|\mathbf{x}_i)\right\} & (26)
\end{aligned}$$

We must find the first and second order partial derivatives of L with respect to each $\rho_{m^*i^*}$.

$$\begin{aligned}
\frac{\partial}{\partial \rho_{m^*i^*}} L(\boldsymbol{\rho}) &= \frac{\partial}{\partial \rho_{m^*i^*}} \sum_{i=1}^N \log\left\{\sum_{m=1}^M \frac{\exp(\rho_{mi})}{\sum_{m'=1}^M \exp(\rho_{m'i})} f_m(y_i|\mathbf{x}_i)\right\} & (27)\\
&= \left\{\frac{1}{\sum_{m=1}^M \frac{\exp(\rho_{mi^*})}{\sum_{m'=1}^M \exp(\rho_{m'i^*})} f_m(y_{i^*}|\mathbf{x}_{i^*})}\right\} \times \frac{\partial}{\partial \rho_{m^*i^*}} \sum_{m=1}^M \frac{\exp(\rho_{mi^*})}{\sum_{m'=1}^M \exp(\rho_{m'i^*})} f_m(y_{i^*}|\mathbf{x}_{i^*}) & (28)
\end{aligned}$$

Now note that for $m^* = m$,

$$\begin{aligned}
\frac{\partial}{\partial \rho_{m^* i^*}} \pi_{m i^*} &= \frac{\partial}{\partial \rho_{m^* i^*}} \frac{\exp(\rho_{m i^*})}{\sum_{m'=1}^{M} \exp(\rho_{m' i^*})} \\
&= \frac{\exp(\rho_{m i^*}) \left\{\sum_{m'=1}^{M} \exp(\rho_{m' i^*})\right\} - \exp(\rho_{m i^*})^2}{\left\{\sum_{m'=1}^{M} \exp(\rho_{m' i^*})\right\}^2} \\
&= \pi_{m i^*} - \pi_{m i^*}^2 .
\end{aligned}$$

For $m^* \neq m$,

$$\begin{aligned}
\frac{\partial}{\partial \rho_{m^* i^*}} \pi_{m i^*} &= \frac{\partial}{\partial \rho_{m^* i^*}} \frac{\exp(\rho_{m i^*})}{\sum_{m'=1}^{M} \exp(\rho_{m' i^*})} \\
&= \frac{-\exp(\rho_{m i^*}) \exp(\rho_{m^* i^*})}{\left\{\sum_{m'=1}^{M} \exp(\rho_{m' i^*})\right\}^2} \\
&= -\pi_{m i^*} \pi_{m^* i^*} .
\end{aligned}$$

Substituting these results into Equation (28), we obtain

$$\begin{aligned}
\frac{\partial}{\partial \rho_{m^* i^*}} L(\boldsymbol{\rho}) &= \left\{\frac{1}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})}\right\} \left\{\pi_{m^* i^*} \left(f_{m^*}(y_{i^*} | \mathbf{x}_{i^*}) - \sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})\right)\right\} \qquad (29) \\
&= \frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*})}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})} - \pi_{m^* i^*} \qquad (30)
\end{aligned}$$

Now, we calculate the second order derivative as

$$\begin{aligned}
\frac{\partial^2}{\partial \rho_{m^* i^*}^2} L(\boldsymbol{\rho}) &= \frac{\partial}{\partial \rho_{m^* i^*}} \left[\frac{\partial}{\partial \rho_{m^* i^*}} L(\boldsymbol{\rho})\right] \qquad (31) \\
&= \frac{\partial}{\partial \rho_{m^* i^*}} \left[\frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*})}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})} - \pi_{m^* i^*}\right] \qquad (32) \\
&= \frac{\left(\pi_{m^* i^*} - \pi_{m^* i^*}^2\right) f_{m^*}(y_{i^*} | \mathbf{x}_{i^*}) \sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})}{\left\{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})\right\}^2} \qquad (33) \\
&\quad - \frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*}) \left\{\pi_{m^* i^*} \left(f_{m^*}(y_{i^*} | \mathbf{x}_{i^*}) - \sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})\right)\right\}}{\left\{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})\right\}^2} \qquad (34) \\
&\quad - \left(\pi_{m^* i^*} - \pi_{m^* i^*}^2\right) \qquad (35) \\
&= \left(1 - \pi_{m^* i^*}\right) \frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*})}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})} \qquad (36) \\
&\quad - \left[\frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*})}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})}\right]^2 + \pi_{m^* i^*} \frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*})}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})} \qquad (37) \\
&\quad - \left(\pi_{m^* i^*} - \pi_{m^* i^*}^2\right) \qquad (38) \\
&= \frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*})}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})} - \left[\frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*})}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})}\right]^2 - \left(\pi_{m^* i^*} - \pi_{m^* i^*}^2\right) \qquad (39)
\end{aligned}$$

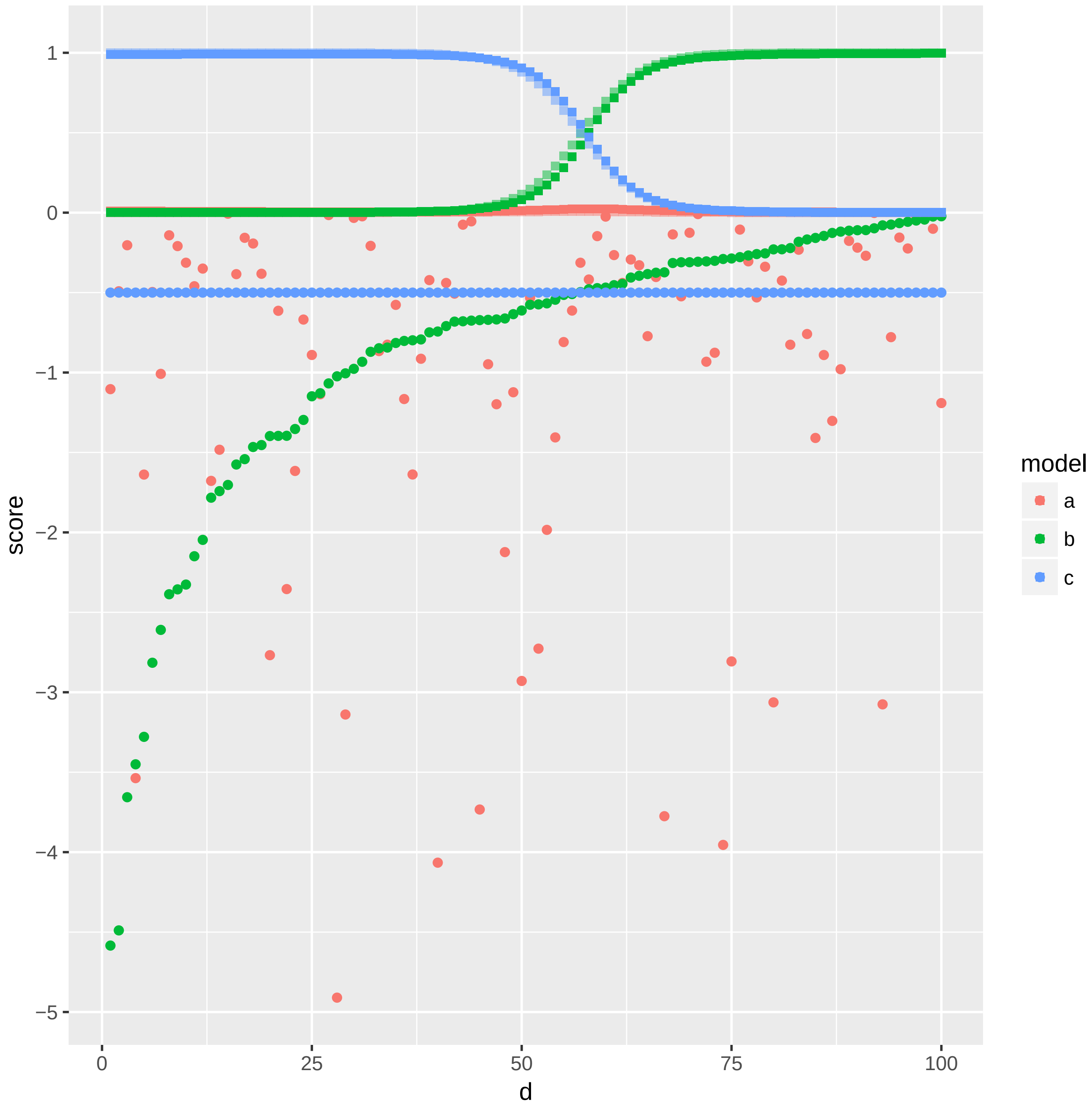
3 Simulated Application

```
library(plyr)
library(tidyr)
library(dplyr)

##
## Attaching package:  'dplyr'
## The following objects are masked from 'package:plyr':
##
##    arrange, count, desc, failwith, id, mutate, rename, summarise,
##    summarize
## The following objects are masked from 'package:stats':
##
##    filter, lag
## The following objects are masked from 'package:base':
##
##    intersect, setdiff, setequal, union

library(densitystackr)
library(ggplot2)
library(awes)

### For now, let's just make up some data for the purposes of method development
### this will need to go into test code too
set.seed(9873)
loso_pred_res <- data.frame(
  model = paste0("log_score_", rep(letters[1:3], each = 100)),
  d = rep(1:100, times = 3),
  loso_log_score = c(
    log(runif(100, 0, 1)), # model a's performance not related to d
    sort(log(runif(100, 0, 1))), # model b's performance increasing in d
    rep(-0.5, 100))  # model c's performance constant
) %>%
  spread(model, loso_log_score) %>%
  mutate(e = rnorm(100))

## Obtain stacking model fit
#debug(density_stack_splines_fixed_regularization)
fit_time <- system.time({
  stacking_fit <- density_stack_splines_fixed_regularization(
    log_score_a + log_score_b + log_score_c ~ d,
    data = loso_pred_res,
    lambda = 1,
    tol = 10^{-4},
    maxit = 10^5,
    verbose = 0)
})

cat(fit_time)

## 22.748 0.012 68.592 0 0

require(doMC)

## Loading required package:  doMC
## Loading required package:  foreach
## Loading required package:  iterators
## Loading required package:  parallel
```

```
registerDoMC(4)

#debug(density_stack)
fit_time_cv_reg <- system.time({
  stacking_fit_cv_reg <- density_stack(
    log_score_a + log_score_b + log_score_c ~ d,
    data = loso_pred_res,
    df = data.frame(d = seq(from = 3, by = 3, length = 10)),
    min_obs_weight = 1,
    tol = 10^{-3},
    maxit = 10^5,
    cv_folds = NULL,
    cv_nfolds = 10L,
    verbose = 1)
})
```

```
## Fitting cv model 1 of 10
## Fitting cv model 2 of 10
## Fitting cv model 3 of 10
## Fitting cv model 4 of 10
## Fitting cv model 5 of 10
## Fitting cv model 6 of 10
## Fitting cv model 7 of 10
## Fitting cv model 8 of 10
## Fitting cv model 9 of 10
## Fitting cv model 10 of 10
```

```
## Make a plot
component_model_scores_df <- as.data.frame(as.matrix(
      loso_pred_res[, paste0("log_score_", letters[1:3]), drop = FALSE]
    ) %>%
    `storage.mode<-`("double")) %>%
  `colnames<-`(letters[1:3]) %>%
  gather_("model", "score", letters[1:3]) %>%
  mutate(d = rep(1:100, 3))

component_model_weights_df <-
  compute_model_weights(stacking_fit,
    newdata = data.frame(d = 1:100),
    log = FALSE) %>%
  as.data.frame() %>%
  `colnames<-`(letters[1:3]) %>%
  gather_("model", "weight", letters[1:3]) %>%
  mutate(d = rep(1:100, 3))

component_model_weights_df_cv_reg <-
  compute_model_weights(stacking_fit_cv_reg,
    newdata = data.frame(d = 1:100),
    log = FALSE) %>%
  as.data.frame() %>%
  `colnames<-`(letters[1:3]) %>%
  gather_("model", "weight", letters[1:3]) %>%
  mutate(d = rep(1:100, 3))

ggplot() +
  geom_point(aes(x = d, y = score, colour = model), data = component_model_scores_df) +
  geom_point(aes(x = d, y = weight, colour = model), alpha = 0.5, shape = 15, data = component_model_weigh
  geom_point(aes(x = d, y = weight, colour = model), shape = 15, data = component_model_weights_df_cv_reg)
```



4 Real Application

```r
region <- "National"
prediction_target <- "onset"
explanatory_variables <- c("analysis_time_season_week")

awes_path <- find.package("awes")
target_loso_pred_res <- assemble_stacking_inputs(
  regions = region,
  prediction_target = prediction_target,
  component_model_names = c("kde", "kcde", "sarima"),
  explanatory_variables = explanatory_variables,
  include_model_performance = TRUE,
  preds_path = file.path(awes_path, "estimation/loso-predictions")
)

## get cross-validation groups
cv_folds <- lapply(unique(target_loso_pred_res$analysis_time_season),
  function(season_val) {
    which(target_loso_pred_res$analysis_time_season == season_val)
  })

require(doMC)
registerDoMC(4)

#debug(density_stack)
fit_formula <- as.formula(paste0(
  paste0(c("kde", "kcde", "sarima"), "_log_score", collapse = " + "),
  " ~ ",
  paste(explanatory_variables, collapse = " + ")))
fit_time_cv_reg <- system.time({
  stacking_fit_cv_reg <- density_stack(
    fit_formula,
    data = target_loso_pred_res,
    df = data.frame(
      analysis_time_season_week = c(2, seq(from = 5, by = 5, to = 30))
    ),
    min_obs_weight = 1,
    tol = 10^{-3},
    maxit = 10^5,
    cv_folds = cv_folds,
    verbose = 1)
})
```

```
## Fitting cv model 1 of 7
## Fitting cv model 2 of 7
## Fitting cv model 3 of 7
## Fitting cv model 4 of 7
## Fitting cv model 5 of 7
## Fitting cv model 6 of 7
## Fitting cv model 7 of 7
```

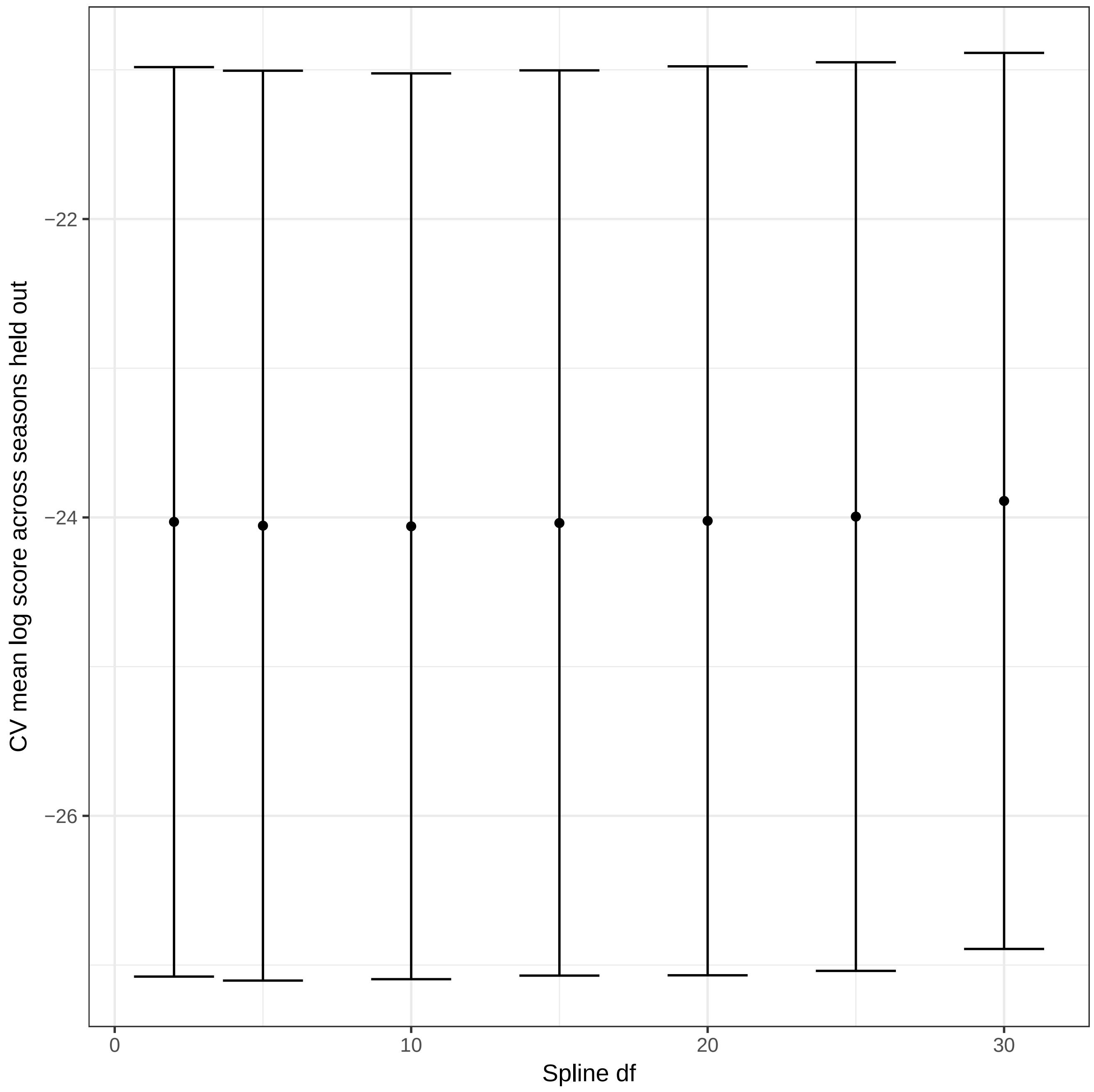
```r
stacking_fits_fixed_reg <- lapply(c(2, seq(from = 5, by = 5, to = 30)),
  function(df) {
    density_stack(
      fit_formula,
      data = target_loso_pred_res,
      df = df,
      min_obs_weight = 1,
      tol = 10^{-3},
      maxit = 10^5,
      cv_folds = cv_folds,
      verbose = 0)
  })
```

Estimation run time with leave-one-season-out cross-validation to compare spline df values

```
fit_time_cv_reg

##     user   system  elapsed
## 1412.328    1.960 1487.406
```

Plot of cross-validation results

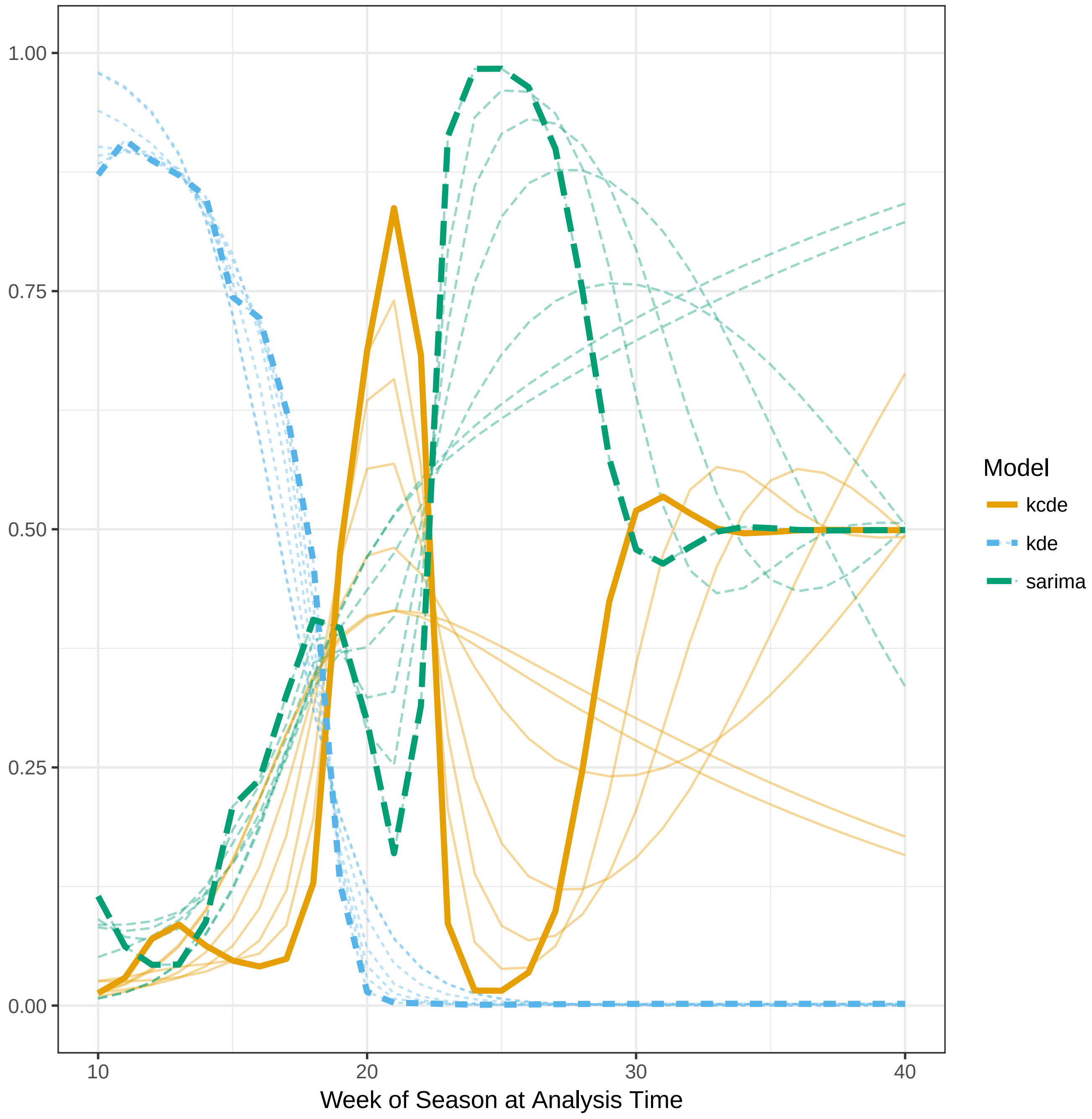
```
plot_df <- stacking_fit_cv_reg$cv_results
plot_df$se <- apply(plot_df[, 2:13], 1, function(x) sd(x) / sqrt(12))
plot_df$lb <- plot_df$cv_log_score_mean - plot_df$se
plot_df$ub <- plot_df$cv_log_score_mean + plot_df$se

ggplot(plot_df) +
  geom_point(aes(x = analysis_time_season_week, y = cv_log_score_mean)) +
  geom_errorbar(aes(x = analysis_time_season_week, ymin = lb, ymax = ub)) +
  xlab("Spline df") +
  ylab("CV mean log score across seasons held out") +
  theme_bw()
```



Plot of weights from spline fits with different degrees of freedom – best darker

```r
get_legend_grob <- function(x) {
  data <- ggplot2:::ggplot_build(x)

  plot <- data$plot
  panel <- data$panel
  data <- data$data
  theme <- ggplot2:::plot_theme(plot)
  position <- theme$legend.position
  if (length(position) == 2) {
    position <- "manual"
  }
  legend_box <- if (position != "none") {
    ggplot2:::build_guides(plot$scales, plot$layers, plot$mapping,
      position, theme, plot$guides, plot$labels)
  } else {
    ggplot2:::zeroGrob()
  }
  if (ggplot2:::is.zero(legend_box)) {
    position <- "none"
  }
  else {
    legend_width <- gtable:::gtable_width(legend_box) + theme$legend.margin
    legend_height <- gtable:::gtable_height(legend_box) + theme$legend.margin
    just <- valid.just(theme$legend.justification)
    xjust <- just[1]
    yjust <- just[2]
    if (position == "manual") {
      xpos <- theme$legend.position[1]
      ypos <- theme$legend.position[2]
      legend_box <- editGrob(legend_box, vp = viewport(x = xpos,
        y = ypos, just = c(xjust, yjust), height = legend_height,
        width = legend_width))
    }
    else {
      legend_box <- editGrob(legend_box, vp = viewport(x = xjust,
        y = yjust, just = c(xjust, yjust)))
    }
  }
  return(legend_box)
}


## get resulting model weights at a grid of values for chosen explanatory variables
if("kcde_model_confidence" %in% explanatory_variables ||
    "sarima_model_confidence" %in% explanatory_variables) {
  if(identical(prediction_target, "peak_inc")) {
    model_confidence_grid <- seq(from = 1, to = 99, by = 2)
  } else if(identical(prediction_target, "onset")) {
    model_confidence_grid <- seq_len(34)
  } else {
    ## peak_week
    model_confidence_grid <- seq_len(33)
  }
} else {
  model_confidence_grid <- 1
}

if("weighted_ili" %in% explanatory_variables) {
  weighted_ili_grid <- seq(from = 0, to = 13, by = 0.5)
} else {
  weighted_ili_grid <- 1
}

typical_analysis_time_season_week <- 17L
```

```r
if(prediction_target %in% c("onset", "peak_week")) {
  typical_model_confidence <- 5L
} else {
  typical_model_confidence <- 20L
}
typical_ili <- 2.5

newdata <- bind_rows(
  expand.grid(
    analysis_time_season_week = 10:40,
    kcde_model_confidence = model_confidence_grid,
    sarima_model_confidence = typical_model_confidence,
    weighted_ili = typical_ili,
    stringsAsFactors = FALSE)[, explanatory_variables, drop = FALSE],
  expand.grid(
    analysis_time_season_week = 10:40,
    kcde_model_confidence = typical_model_confidence,
    sarima_model_confidence = model_confidence_grid,
    weighted_ili = typical_ili,
    stringsAsFactors = FALSE)[, explanatory_variables, drop = FALSE],
  expand.grid(
    analysis_time_season_week = 10:40,
    kcde_model_confidence = typical_model_confidence,
    sarima_model_confidence = typical_model_confidence,
    weighted_ili = weighted_ili_grid,
    stringsAsFactors = FALSE)[, explanatory_variables, drop = FALSE],
  expand.grid(
    analysis_time_season_week = typical_analysis_time_season_week,
    kcde_model_confidence = model_confidence_grid,
    sarima_model_confidence = model_confidence_grid,
    weighted_ili = typical_ili,
    stringsAsFactors = FALSE)[, explanatory_variables, drop = FALSE],
  expand.grid(
    analysis_time_season_week = typical_analysis_time_season_week,
    kcde_model_confidence = model_confidence_grid,
    sarima_model_confidence = typical_model_confidence,
    weighted_ili = weighted_ili_grid,
    stringsAsFactors = FALSE)[, explanatory_variables, drop = FALSE],
  expand.grid(
    analysis_time_season_week = typical_analysis_time_season_week,
    kcde_model_confidence = typical_model_confidence,
    sarima_model_confidence = model_confidence_grid,
    weighted_ili = weighted_ili_grid,
    stringsAsFactors = FALSE)[, explanatory_variables, drop = FALSE]
  ) %>%
  distinct()



model_weights <- compute_model_weights(stacking_fit_cv_reg,
  newdata = newdata,
  log = FALSE)

# model_weights <- compute_model_weights(stacking_fit_cv_fixed,
#     newdata = newdata,
#     log = FALSE)
colnames(model_weights) <- c("kde", "kcde", "sarima")
weights <- cbind(newdata,
  model_weights) %>% as.data.frame()

weights <- weights %>%
  gather_("model", "weight", c("kde", "kcde", "sarima"))

model_weights_fixed_reg <- lapply(stacking_fits_fixed_reg,
  function(stacking_fit) {
    model_weights_one_fit <- compute_model_weights(stacking_fit,
      newdata = newdata,
      log = FALSE)

    # model_weights <- compute_model_weights(stacking_fit_cv_fixed,
    #     newdata = newdata,
    #     log = FALSE)
    colnames(model_weights_one_fit) <- c("kde", "kcde", "sarima")
    weights_one_fit <- cbind(newdata,
      model_weights_one_fit) %>% as.data.frame()

    weights_one_fit <- weights_one_fit %>%
      gather_("model", "weight", c("kde", "kcde", "sarima"))
    weights_one_fit$df <- as.character(stacking_fit$spline_fits[[1]]$df)

    return(weights_one_fit)
  }) %>%
  rbind.fill()

  color_palette <- c("#E69F00", "#56B4E9", "#009E73")

  p_onset_weights <- ggplot() +
    geom_line(aes(x = analysis_time_season_week,
        y = weight,
        colour = model,
        linetype = model,
        group = factor(paste0(df, model))
      ),
      alpha = 0.4,
      data = model_weights_fixed_reg) +
    geom_line(aes(x = analysis_time_season_week,
        y = weight,
        colour = model,
        linetype = model
      ),
      size = 1.3,
      data = weights) +
  #  facet_wrap( ~ quantity) +
    scale_linetype("Model") +
    scale_colour_manual("Model", values = color_palette) +
    ylim(c(0,1)) +
    xlab("Week of Season at Analysis Time") +
  #  ylab("Model Weight") +
  #  ggtitle("B: Component Model Weights") +
    theme_bw(base_size = 11) +
    theme(axis.title.y = element_blank(),
#       legend.position = "none",
      plot.margin = unit(c(3, 6, 0, 5), "pt"))

print(p_onset_weights)
```

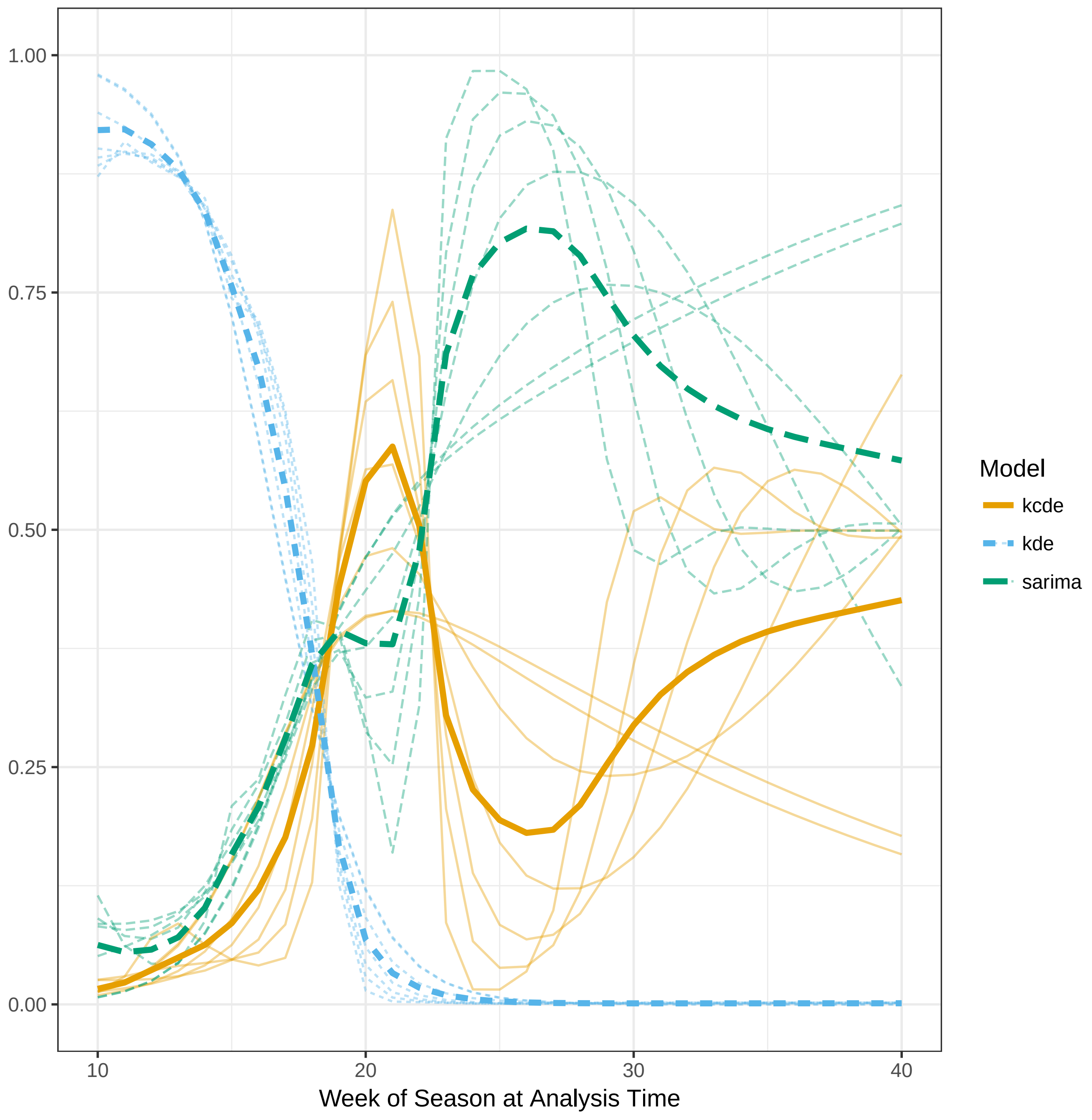


Which spline df values resulted in fits approximately as good as the best? All of them!

```
cv_results <- stacking_fit_cv_reg$cv_results[, 2:13]
best_params_ind <- which.max(stacking_fit_cv_reg$cv_results[, 14])
cv_nfolds <- 12
sapply(seq_len(nrow(cv_results)),
  function(ind) {
    t.test(
      x = as.numeric(cv_results[ind, paste0("cv_log_score_fold_", seq_len(cv_nfolds))]),
      y = as.numeric(cv_results[best_params_ind, paste0("cv_log_score_fold_", seq_len(cv_nfolds))]),
      paired = TRUE
    )$p.value >= 0.05
  })

## [1] TRUE TRUE TRUE TRUE TRUE TRUE   NA
```

Plot of weights from spline fits with different degrees of freedom – COMBINED darker

```
model_weights_fixed_reg <- lapply(stacking_fits_fixed_reg,
  function(stacking_fit) {
    model_weights_one_fit <- compute_model_weights(stacking_fit,
      newdata = newdata,
      log = FALSE)

    # model_weights <- compute_model_weights(stacking_fit_cv_fixed,
    #     newdata = newdata,
    #     log = FALSE)
    colnames(model_weights_one_fit) <- c("kde", "kcde", "sarima")
    weights_one_fit <- cbind(newdata,
      model_weights_one_fit) %>% as.data.frame()

    weights_one_fit <- weights_one_fit %>%
      gather_("model", "weight", c("kde", "kcde", "sarima"))
    weights_one_fit$df <- as.character(stacking_fit$spline_fits[[1]]$df)

    return(weights_one_fit)
  }) %>%
  rbind.fill()

weights <- model_weights_fixed_reg %>%
  group_by(analysis_time_season_week, model) %>%
  summarize(weight = mean(weight)) %>%
  ungroup()

  color_palette <- c("#E69F00", "#56B4E9", "#009E73")

  p_onset_weights <- ggplot() +
    geom_line(aes(x = analysis_time_season_week,
        y = weight,
        colour = model,
        linetype = model,
        group = factor(paste0(df, model))
      ),
      alpha = 0.4,
      data = model_weights_fixed_reg) +
    geom_line(aes(x = analysis_time_season_week,
        y = weight,
        colour = model,
        linetype = model
      ),
      size = 1.3,
      data = weights) +
  #  facet_wrap( ~ quantity) +
    scale_linetype("Model") +
    scale_colour_manual("Model", values = color_palette) +
    ylim(c(0,1)) +
    xlab("Week of Season at Analysis Time") +
  #  ylab("Model Weight") +
  #  ggtitle("B: Component Model Weights") +
    theme_bw(base_size = 11) +
    theme(axis.title.y = element_blank(),
#       legend.position = "none",
      plot.margin = unit(c(3, 6, 0, 5), "pt"))

print(p_onset_weights)
```



CODE NOT RUN:

```
library(plyr)
library(tidyr)
library(dplyr)
library(densitystackr)
library(ggplot2)
library(awes)

region <- "National"
prediction_target <- "peak_inc"
explanatory_variables <- c("analysis_time_season_week", "kcde_model_confidence", "sarima_model_confidence"
```

```r
awes_path <- find.package("awes")
target_loso_pred_res <- assemble_stacking_inputs(
  regions = region,
  prediction_target = prediction_target,
  component_model_names = c("kde", "kcde", "sarima"),
  explanatory_variables = explanatory_variables,
  include_model_performance = TRUE,
  preds_path = file.path(awes_path, "estimation/loso-predictions")
)

## get cross-validation groups
cv_folds <- lapply(unique(target_loso_pred_res$analysis_time_season),
  function(season_val) {
    which(target_loso_pred_res$analysis_time_season == season_val)
  })

require(doMC)
registerDoMC(4)

#debug(density_stack)
fit_formula <- as.formula(paste0(
  paste0(c("kde", "kcde", "sarima"), "_log_score", collapse = " + "),
  " ~ ",
  paste(explanatory_variables, collapse = " + ")))
fit_time_cv_reg <- system.time({
  stacking_fit_cv_reg <- density_stack(
    fit_formula,
    data = target_loso_pred_res,
    df = data.frame(
      analysis_time_season_week = seq(from = 5, by = 5, length = 5),
      kcde_model_confidence = seq(from = 5, by = 5, length = 5),
      sarima_model_confidence = seq(from = 5, by = 5, length = 5)
    ),
    min_obs_weight = 1,
    tol = 10^{-3},
    maxit = 10^5,
    cv_folds = cv_folds,
    verbose = 1)
})


fit_time_cv_fixed <- system.time({
  stacking_fit_cv_fixed <- density_stack(
    fit_formula,
    data = target_loso_pred_res,
    lambda = 1,
    min_obs_weight = 1,
    tol = 10^{-3},
    maxit = 10^5,
    verbose = 2)
})

## get resulting model weights at a grid of values for chosen explanatory variables
if("kcde_model_confidence" %in% explanatory_variables ||
    "sarima_model_confidence" %in% explanatory_variables) {
  if(identical(prediction_target, "peak_inc")) {
    model_confidence_grid <- seq(from = 1, to = 99, by = 2)
  } else if(identical(prediction_target, "onset")) {
    model_confidence_grid <- seq_len(34)
  } else {
    ## peak_week
    model_confidence_grid <- seq_len(33)
  }
} else {
  model_confidence_grid <- 1
}

if("weighted_ili" %in% explanatory_variables) {
  weighted_ili_grid <- seq(from = 0, to = 13, by = 0.5)
} else {
  weighted_ili_grid <- 1
}

typical_analysis_time_season_week <- 17L
if(prediction_target %in% c("onset", "peak_week")) {
  typical_model_confidence <- 5L
} else {
  typical_model_confidence <- 20L
}
typical_ili <- 2.5

newdata <- bind_rows(
  expand.grid(
    analysis_time_season_week = 10:40,
    kcde_model_confidence = model_confidence_grid,
    sarima_model_confidence = typical_model_confidence,
    weighted_ili = typical_ili,
    stringsAsFactors = FALSE)[, explanatory_variables, drop = FALSE],
  expand.grid(
    analysis_time_season_week = 10:40,
    kcde_model_confidence = typical_model_confidence,
    sarima_model_confidence = model_confidence_grid,
    weighted_ili = typical_ili,
    stringsAsFactors = FALSE)[, explanatory_variables, drop = FALSE],
  expand.grid(
    analysis_time_season_week = 10:40,
    kcde_model_confidence = typical_model_confidence,
    sarima_model_confidence = typical_model_confidence,
    weighted_ili = weighted_ili_grid,
    stringsAsFactors = FALSE)[, explanatory_variables, drop = FALSE],
  expand.grid(
    analysis_time_season_week = typical_analysis_time_season_week,
    kcde_model_confidence = model_confidence_grid,
    sarima_model_confidence = model_confidence_grid,
    weighted_ili = typical_ili,
    stringsAsFactors = FALSE)[, explanatory_variables, drop = FALSE],
  expand.grid(
    analysis_time_season_week = typical_analysis_time_season_week,
```

```r
    kcde_model_confidence = model_confidence_grid,
    sarima_model_confidence = typical_model_confidence,
    weighted_ili = weighted_ili_grid,
    stringsAsFactors = FALSE)[, explanatory_variables, drop = FALSE],
  expand.grid(
    analysis_time_season_week = typical_analysis_time_season_week,
    kcde_model_confidence = typical_model_confidence,
    sarima_model_confidence = model_confidence_grid,
    weighted_ili = weighted_ili_grid,
    stringsAsFactors = FALSE)[, explanatory_variables, drop = FALSE]
  ) %>%
  distinct()

model_weights <- compute_model_weights(stacking_fit_cv_fixed,
    newdata = newdata,
    log = FALSE)
colnames(model_weights) <- c("kde", "kcde", "sarima")
weights <- cbind(newdata,
  model_weights) %>% as.data.frame()

weights <- weights %>%
  gather_("model", "weight", c("kde", "kcde", "sarima"))

typical_season_week <- 17L
if(prediction_target %in% c("onset", "peak_week")) {
  typical_confidence <- 5L
} else {
  typical_confidence <- 20L
}

p1 <- ggplot(
  weights %>%
    filter(sarima_model_confidence == typical_confidence &
        kcde_model_confidence %% 2 == 1) %>%
    mutate(model = toupper(model))) +
  geom_raster(aes(x = analysis_time_season_week, y = kcde_model_confidence, fill = weight)) +
  scale_fill_gradient2("Model\nWeight",
    breaks = c(0, 0.25, 0.5, 0.75, 1),
    labels = as.character(c(0, 0.25, 0.5, 0.75, 1)),
    low = "#b2182b",
    mid = "#f7f7f7",
    high = "#2166ac",
    midpoint = 0.5) +
  facet_wrap(~ model, nrow = 1L) +
  xlab("Season Week at Analysis Time") +
  ylab("KCDE Model Uncertainty") +
  theme_bw()

plot_legend <- get_legend_grob(p1)

p1 <- p1 + theme(legend.position = "none")

p2 <- ggplot(
  weights %>%
    filter(kcde_model_confidence == typical_confidence &
        sarima_model_confidence %% 2 == 1) %>%
    mutate(model = toupper(model))) +
  geom_raster(aes(x = analysis_time_season_week, y = sarima_model_confidence, fill = weight)) +
  scale_fill_gradient2("Model\nWeight",
    breaks = c(0, 0.25, 0.5, 0.75, 1),
    labels = as.character(c(0, 0.25, 0.5, 0.75, 1)),
    low = "#b2182b",
    mid = "#f7f7f7",
    high = "#2166ac",
    midpoint = 0.5) +
  facet_wrap(~ model, nrow = 1L) +
  xlab("Season Week at Analysis Time") +
  ylab("SARIMA Model Uncertainty") +
  theme_bw() +
  theme(legend.position = "none")

p3 <- ggplot(
  weights %>%
    filter(
      analysis_time_season_week == typical_season_week &
      kcde_model_confidence %%2 == 1 &
      sarima_model_confidence %% 2 == 1) %>%
    mutate(model = toupper(model))) +
  geom_raster(aes(x = kcde_model_confidence, y = sarima_model_confidence, fill = weight)) +
  scale_fill_gradient2("Model\nWeight",
    breaks = c(0, 0.25, 0.5, 0.75, 1),
    labels = as.character(c(0, 0.25, 0.5, 0.75, 1)),
    low = "#b2182b",
    mid = "#f7f7f7",
    high = "#2166ac",
    midpoint = 0.5) +
  facet_wrap(~ model, nrow = 1L) +
  xlab("KCDE Model Uncertainty") +
  ylab("SARIMA Model Uncertainty") +
  theme_bw() +
  theme(legend.position = "none")


grid.newpage()
num_lines_text <- 1
pushViewport(viewport(layout =
    grid.layout(nrow = 6, ncol = 2,
      heights = unit(
        c(num_lines_text, 1, num_lines_text, 1, num_lines_text, 1),
        rep(c("lines", "null"), 3)),
      widths = unit(c(4, 1), c("null", "null")))))
grid.text("A: SARIMA Model Uncertainty Fixed at 20",
  x = unit(0, "npc"),
  just = "left",
  gp = gpar(fontsize = 12),
  vp = viewport(layout.pos.row = 1, layout.pos.col = 1:2))
grid.text("B: KCDE Model Uncertainty Fixed at 20",
  x = unit(0, "npc"),
```

```r
  just = "left",
  gp = gpar(fontsize = 12),
  vp = viewport(layout.pos.row = 3, layout.pos.col = 1:2))
grid.text("C: Season Week Fixed at 17",
  x = unit(0, "npc"),
  just = "left",
  gp = gpar(fontsize = 12),
  vp = viewport(layout.pos.row = 5, layout.pos.col = 1:2))

print(p1, vp = viewport(layout.pos.row = 2, layout.pos.col = 1))
print(p2, vp = viewport(layout.pos.row = 4, layout.pos.col = 1))
print(p3, vp = viewport(layout.pos.row = 6, layout.pos.col = 1))

pushViewport(viewport(layout.pos.row = 3:4, layout.pos.col = 2))
grid.draw(plot_legend)
upViewport()
```