Week 8: Flexible Regression

PLSC 40502 - Statistical Models

Review

$$ \require{cancel} \DeclareMathOperator*{\argmin}{arg\,min} $$

Previously

$$ \DeclareMathOperator*{\argmax}{arg\,max} $$

- **Item response theory**
  - **Factor model** for categorical/nominal outcome variables
  - Model a matrix of **individual** responses across multiple common **questions**
  - Responses are a function of a common **indivdual** latent parameter
  - Identification via Bayes (prior on the individual latent parameters defines the scale/location).

This week

- **Flexible functional forms**
  - Semi-/Non-parametric approaches to modeling CEFs of Y_i given a continuous X_i
  - Regression and smoothing splines to allow for flexible relationships between
  - Penalty term to avoid "jumpy" regressions
  - Generalized Additive Models (GAMs) that combine "parametric" and "semi-/non-parametric" components
- **Regularization**
  - Why regularize?
  - L_0, L_1, and L_2 norms
  - Value of the lasso (the L_1 norm) - "sparse" regressions
  - Interpreting regularization in Bayesian terms.

Flexible regression

Flexible regression

- A common task in statistics is estimating the conditional expectation function \(E[Y|X]\).
  - But typical methods for estimating the CEF assume that we know its functional form.
  - For example, we assume linearity -- can be trivially satisfied when \(X\) is discrete, but potentially problematic when \(X\) is continuous. $$E[Y|X] = f(X) = X\beta$$
- We want to maintain the utility of a model that is **linear in the parameters** but introduce transformations of \(X\) to capture potentially non-linear relationships between \(Y\) and \(X\).
- Define the **linear basis expansion** for a set of \(M\) basis functions \(h_m(X)\)

$$f(X) = \sum_{m=1}^M \beta_m h_m(X)$$

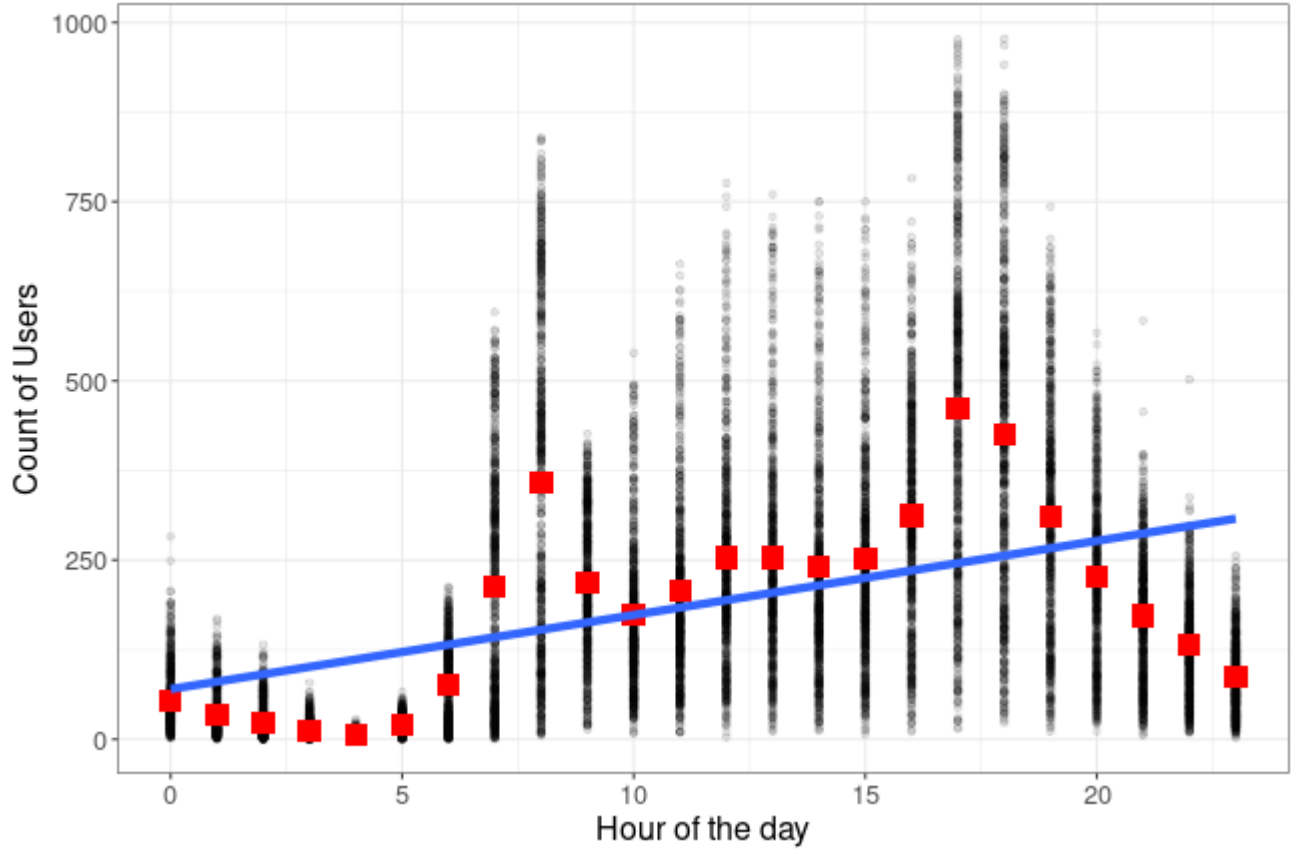
Example: Modeling Bike Rentals

- **Bikeshare usage** is highly variable from day-to-day and hour-to-hour. Capital Bikeshare in Washington D.C. recorded the hourly count of active users over a two-year period from 2011 to 2012.
  - For more on the dataset, see: Fanaee-T, Hadi, and Gama, Joao, "Event labeling combining ensemble detectors and background knowledge", *Progress in Artificial Intelligence* (2013): pp. 1-15

```
bike <- read_csv("data/bikes_hour.csv")
bike_by_hour <- bike %>% group_by(hr) %>% summarize(cnt = mean(cnt))
```

Example: Modeling Bike Rentals

- From the scatterplot of active usage vs. hour of the day, a simple linear fit (slope + intercept) seems quite poor at capturing the CEF



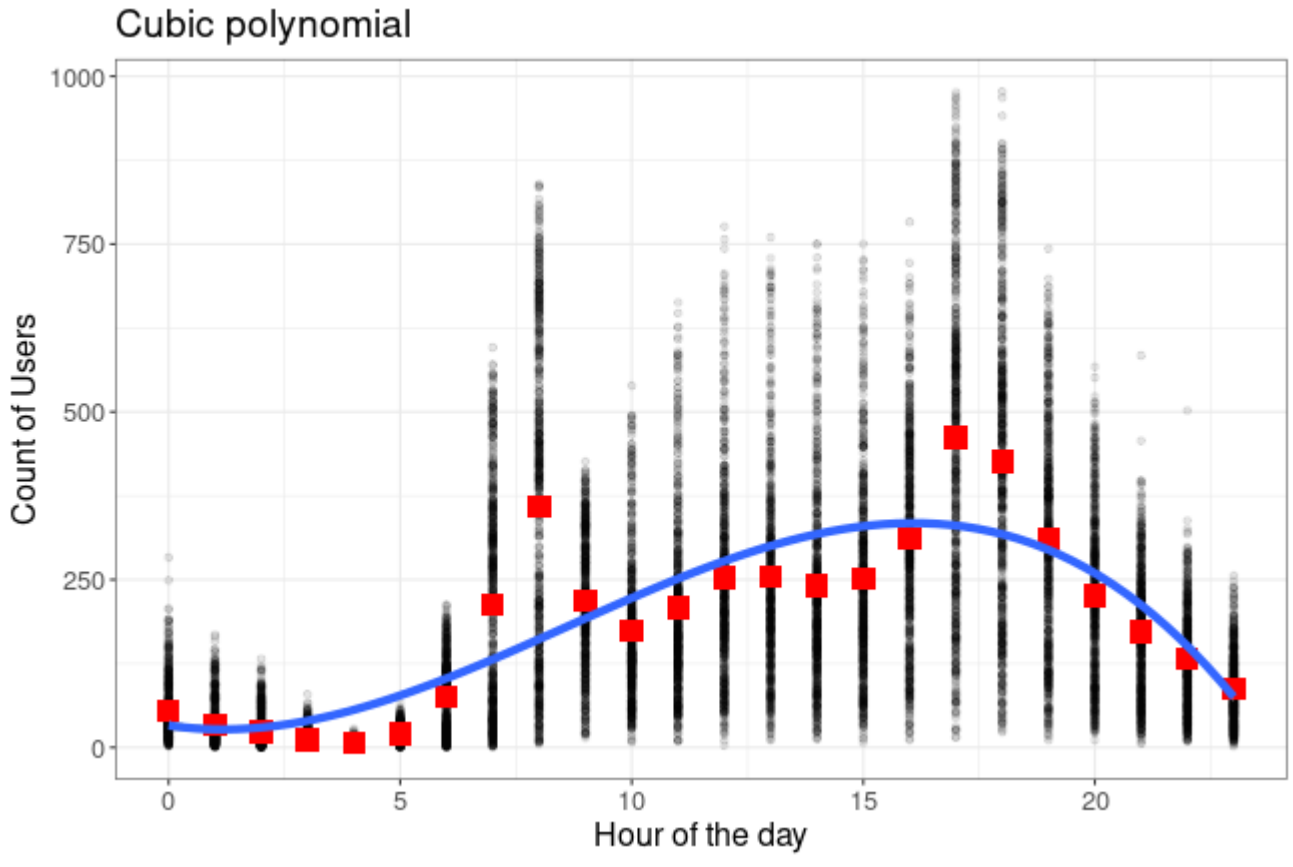
Polynomial basis

- A common set of basis functions to choose are the **global polynomial** basis
  - You've probably already done this when you've included squared terms in your regressions!
- For example, for a univariate \(X\), the basis for a global cubic polynomial is:

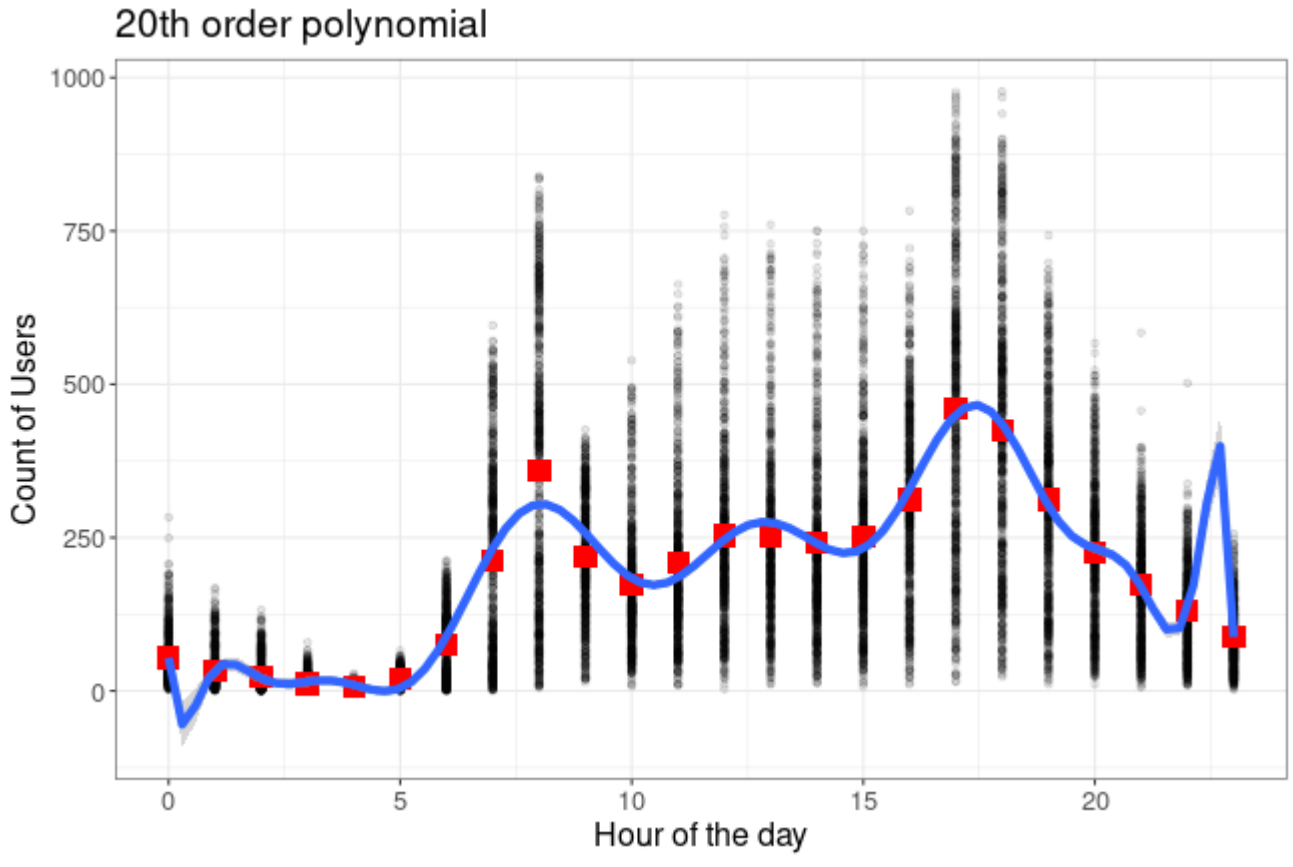
$$\begin{align*}h_1(X) &= 1\\ h_2(X) &= X\\ h_3(X) &= X^2\\ h_4(X) &= X^3\end{align*}$$

- A \(K\)th order polynomial requires \(K+1\) parameters
- However, there are some drawbacks to using a global polynomial - namely that each observation influences the **entire** curve.

Polynomial basis



Polynomial basis

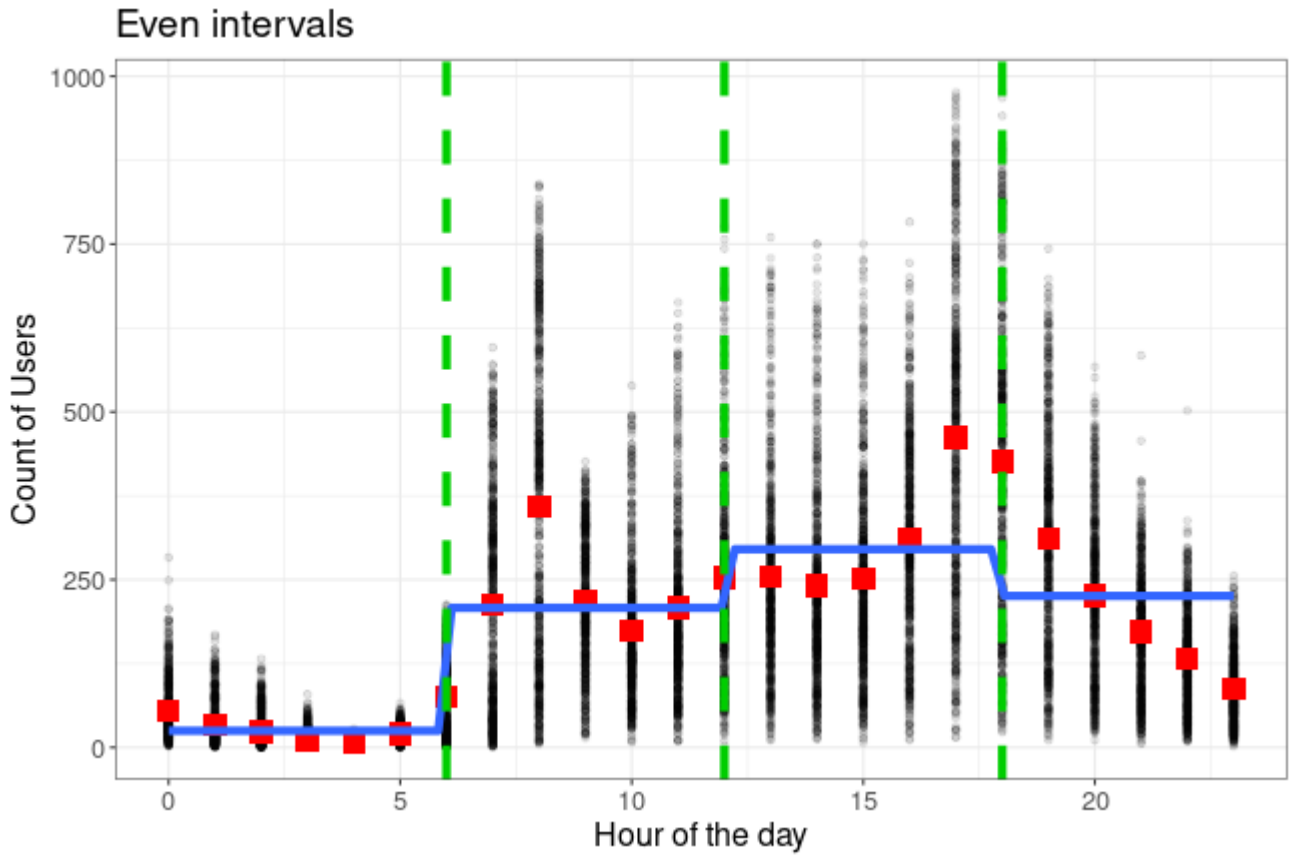


Step functions

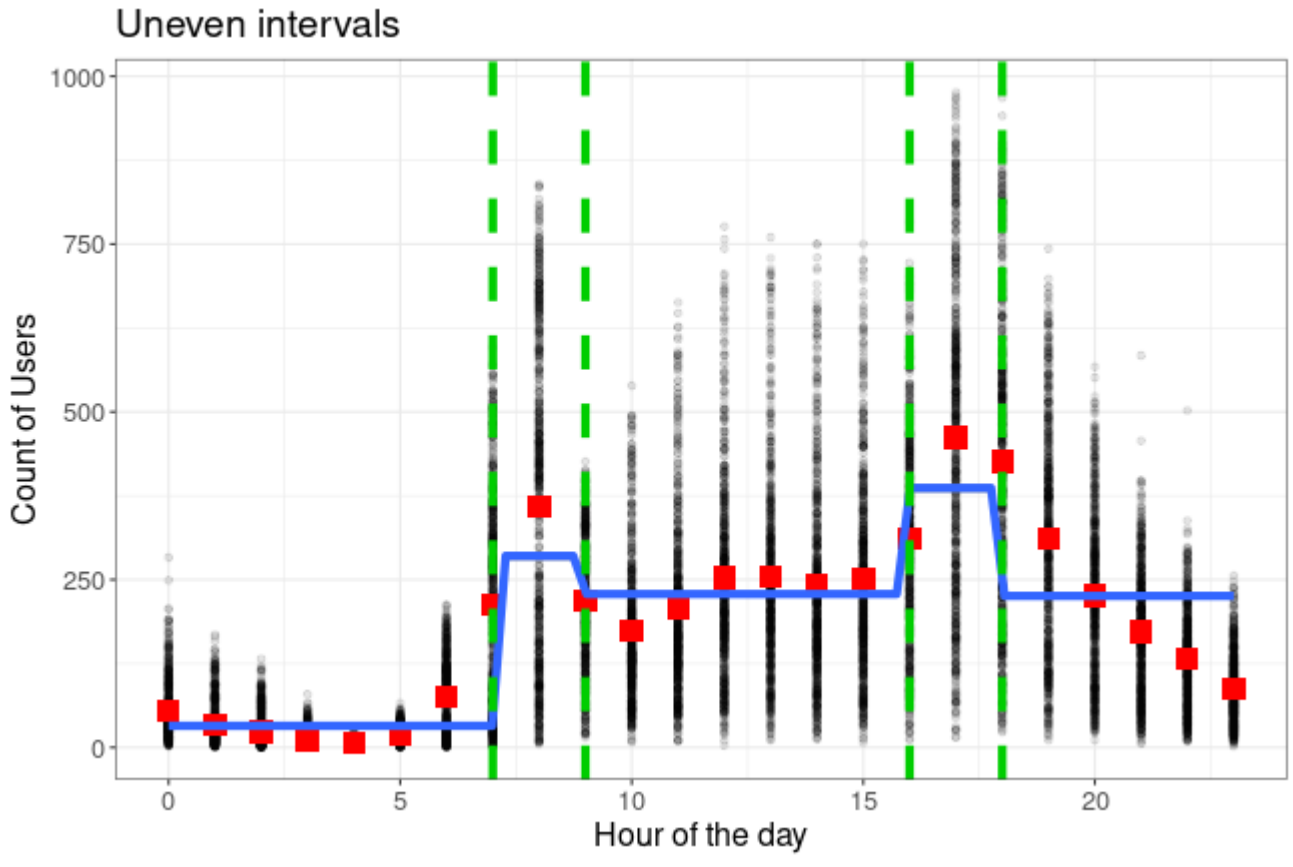
- Instead of forcing a single **global** polynomial, we might instead want to fit a set of **local** averages to different regions of \(X\)
- We could define a set of basis functions that are indicators which partition \(X\) into \(M+1\) disjoint regions based on cutpoints \(\xi_1, \xi_2, \dotsc, \xi_{M}\)

$$\begin{align*} h_1(X) &= I(X < \xi_1)\\ h_2(X) &= I(\xi_1 \le X < \xi_2)\\ h_3(X) &= I(\xi_2 \le X < \xi_3)\\ &\vdots\\ h_{M+1}(X) &= I(\xi_{M} \le X)\\ \end{align*}$$

Step functions

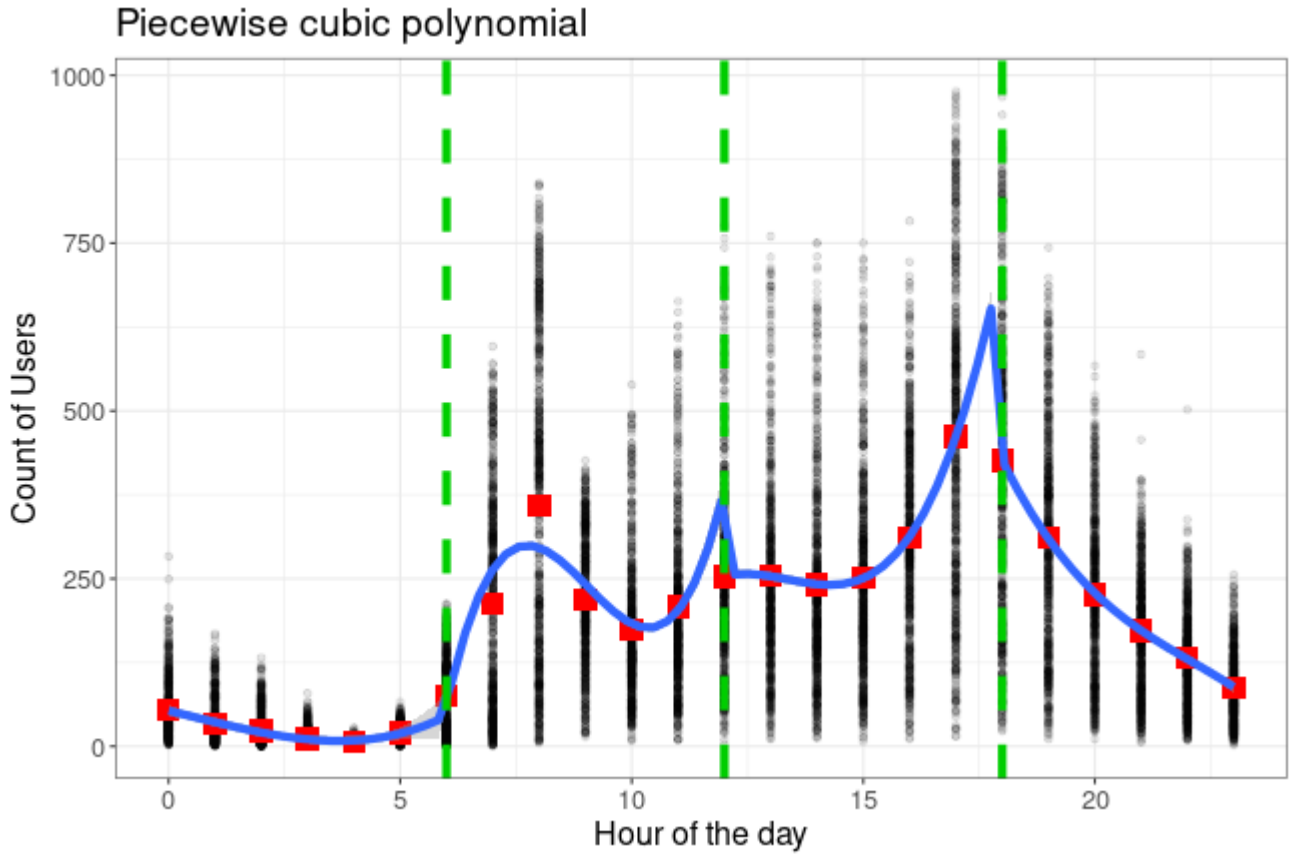


Step functions



Piecewise polynomials

- Rather than just taking the mean within each disjoint region, we could imagine fitting a polynomial to **just** that subset.



Splines

- The piecewise polynomials might fit better, but we still have these irritating discontinuities in the CEF.
  - We might want to impose some **additional** conditions on the function regarding continuity around the cutpoints.
- A $K-1$th order **spline** with M knots $\xi_1, \xi_2, \dotsc, \xi_{M}$ is a piece-wise polynomial that...
  - ...is a polynomial of degree $K-1$ on the intervals $(-\infty, \xi_1,], [\xi_1, \xi_2], [\xi_2, \xi_3], \dotsc, [\xi_m, \infty)$
  - ...has a jth derivative that is continuous at each of the knots $\xi_1, \xi_2, \dotsc, \xi_{m}$ for $j = 0, 1, 2, \dotsc, K-2$
- Intuitively, if we **also** forced the kth derivative to be continuous, we'd recover the **global** polynomial.
- Most common spline is the **cubic** spline $k = 4$ (third order polynomial).
  - Splines allow for local flexibility while still retaining continuity across X.

Splines

- There are actually multiple ways to define the basis functions for a spline. The most intuitive for understanding how they work is the **truncated power basis**

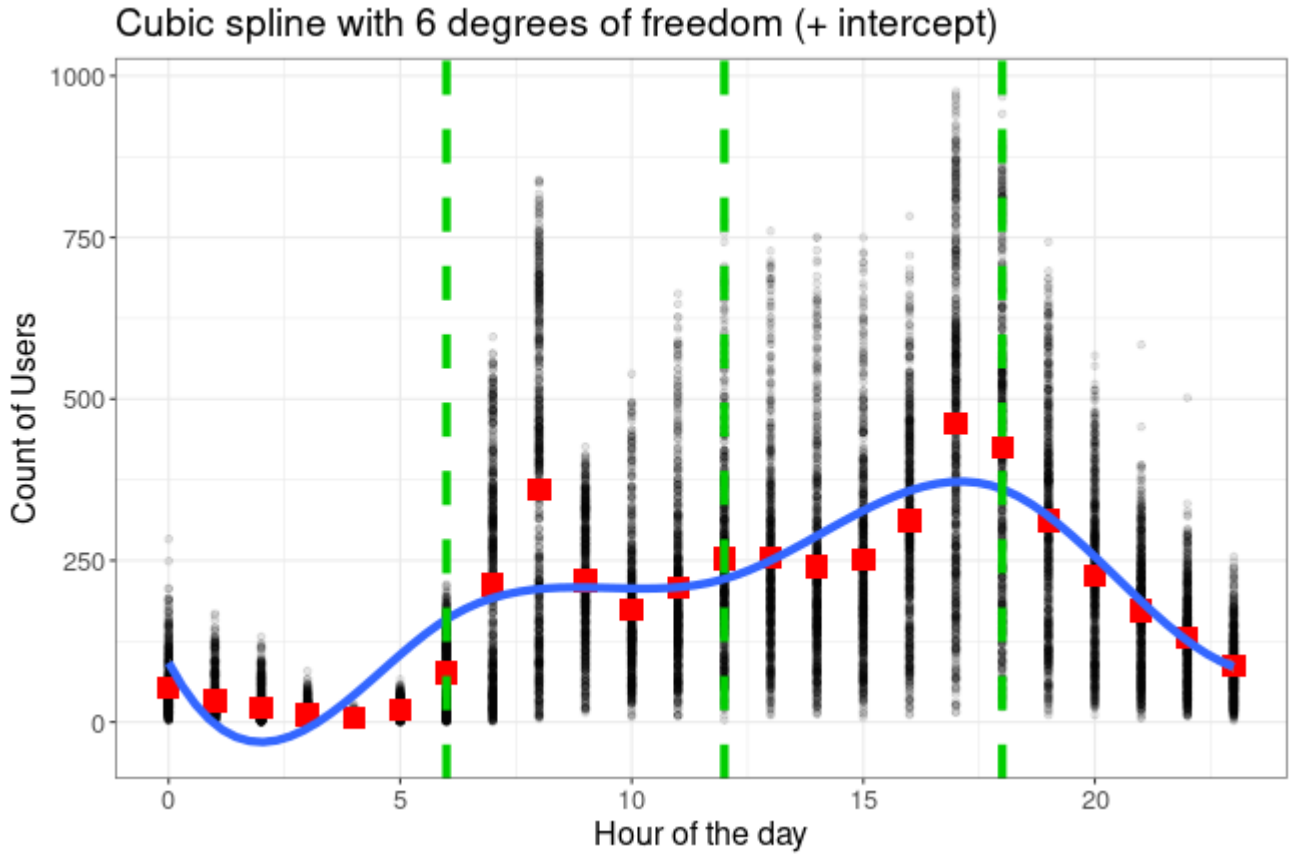
$$\begin{align*} h_{k}(X) &= x^{k-1} & \ \ & k = 1, \dotsc, K\\ h_{m+K}(X) &= (x - \xi_{m})_+^{K-1} & \ \ & m = 1, \dotsc, M \end{align*}$$

where \((\cdot)_+\) denotes a function which returns \(\text{max}(\cdot, 0)\)

- Splines have \(M + K\) "degrees of freedom"
  - "Natural" splines add the constraint that the function is **linear** beyond the constraints of the data
- When the degree and number of knots are fixed, commonly called **regression splines**
  - Contrast with **smoothing splines** where number of indirectly controlled via penalization
- **Trade-offs**
  - Higher \(K\) and higher \(M\) = better in-sample fit but risks overfitting
  - Lower \(K\) and lower \(M\) = poorer in-sample fit (baseline is a global polynomial), but potentially more robust out-of-sample.

Splines

```
bike %>% ggplot(aes(x=hr, y=cnt)) + geom_point(alpha=.1) + geom_point(data=bike_by_hour, col="r
  xlab("Hour of the day") + ylab("Count of Users") + ggtitle("Cubic spline with 6 degrees of fr
  theme_bw() + theme(text = element_text(size = 16))
```



B-splines

- Recall that there are multiple ways of defining the basis functions that construct a spline.
  - Truncated power basis is interpretable but can have poor computational properties
- Alternative: **B-spline** basis
  - Define the spline basis **recursively**
  - Advantage: Non-zero over a limited domain
- For a sequence of $K+M$ knots $\tau_1, \tau_2, \dotsc, \tau_{M+K}$

$$\begin{align*} B_{i, 1}(x) &= \begin{cases} 1 & \text{if} &\tau_i \le x < \tau_{i+1}\\ 0 & \text{otherwise} \end{cases}\\ B_{i, m}(x) &= \frac{x - \tau_i}{\tau_{i + m -1} -\tau_i} B_{i,m-1}(x) + \frac{\tau_{i+m} - x}{\tau_{i + m} - \tau_{i+1}} B_{i+1, m-1}(x) \end{align*}$$

- **Intuition**: Basis functions for higher-order splines are weighted averages of the "neighboring" lower-order basis functions

B-splines

- R will generate a b-spline basis for you using the `bs()` function in `splines`
  - You can treat these like transformations of the regressors

```
bike_bs <- bs(bike$hr, df=6) # By default it's a cubic (degree = 3) and the intercept is omitte
head(bike_bs)
```

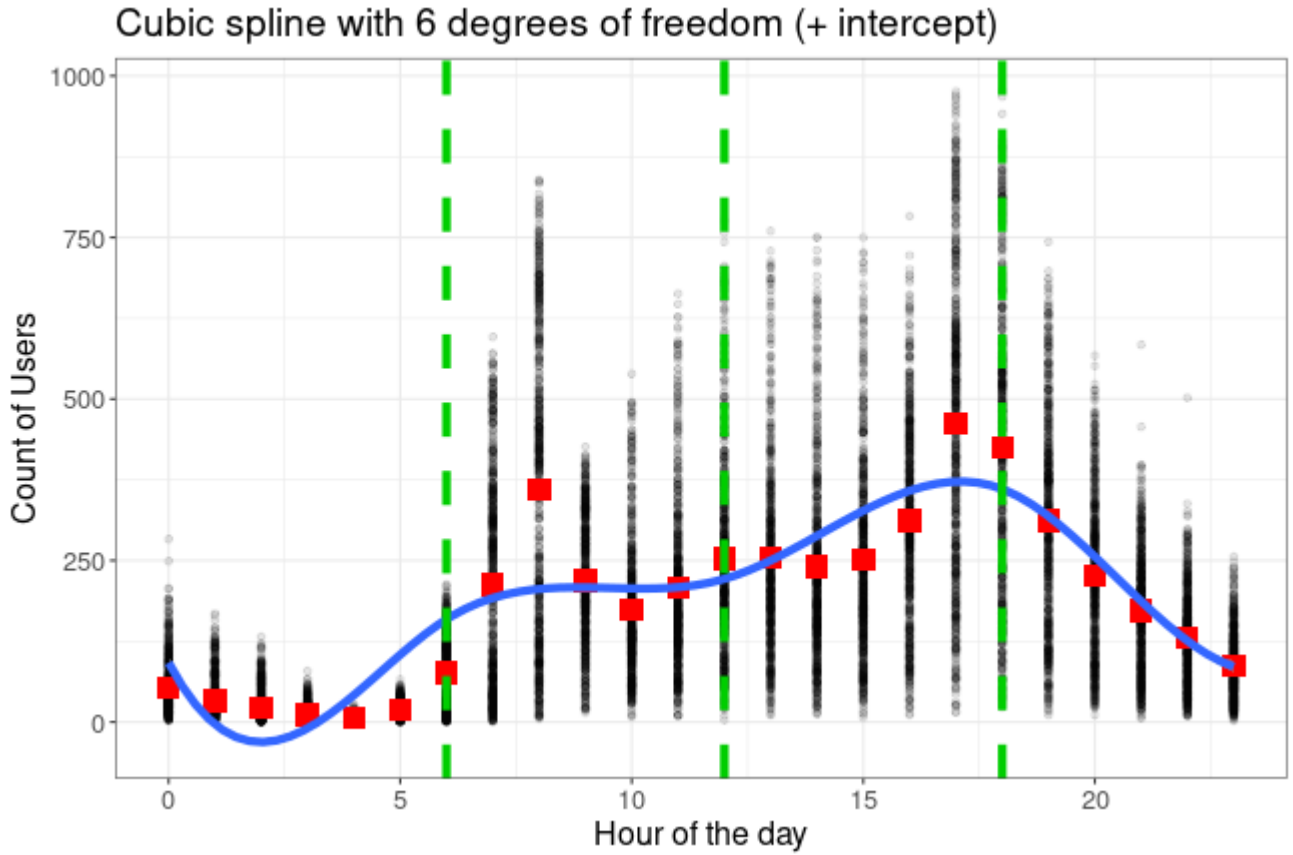
```
##           1      2        3 4 5 6
## [1,] 0.000 0.0000 0.000000 0 0 0
## [2,] 0.383 0.0374 0.000772 0 0 0
## [3,] 0.565 0.1327 0.006173 0 0 0
## [4,] 0.594 0.2604 0.020833 0 0 0
## [5,] 0.519 0.3951 0.049383 0 0 0
## [6,] 0.388 0.5112 0.096451 0 0 0
```

```
attributes(bike_bs)$knots
```

```
## 25% 50% 75%
##   6  12  18
```

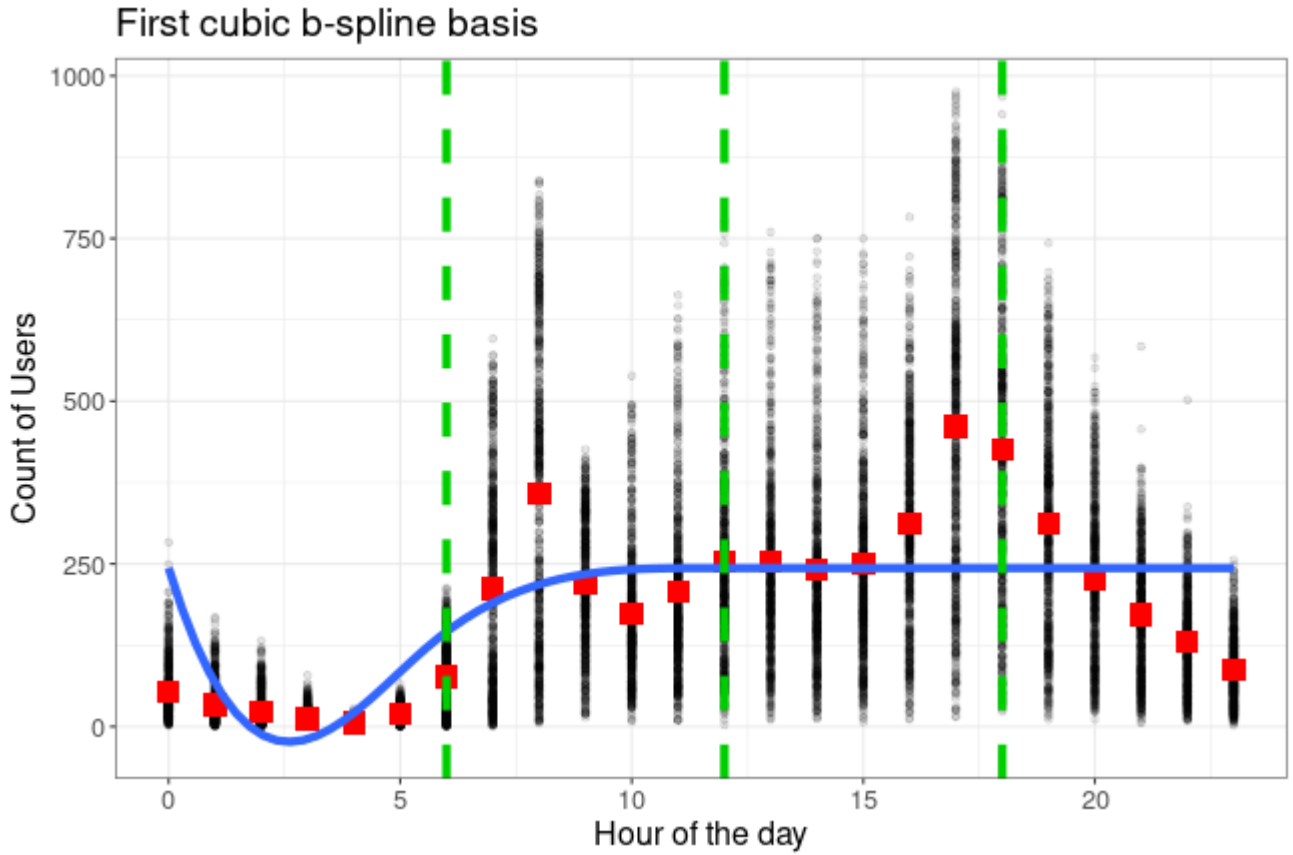
Splines

```
bike %>% ggplot(aes(x=hr, y=cnt)) + geom_point(alpha=.1) + geom_point(data=bike_by_hour, col="r
  xlab("Hour of the day") + ylab("Count of Users") + ggtitle("Cubic spline with 6 degrees of fr
  theme_bw() + theme(text = element_text(size = 16))
```



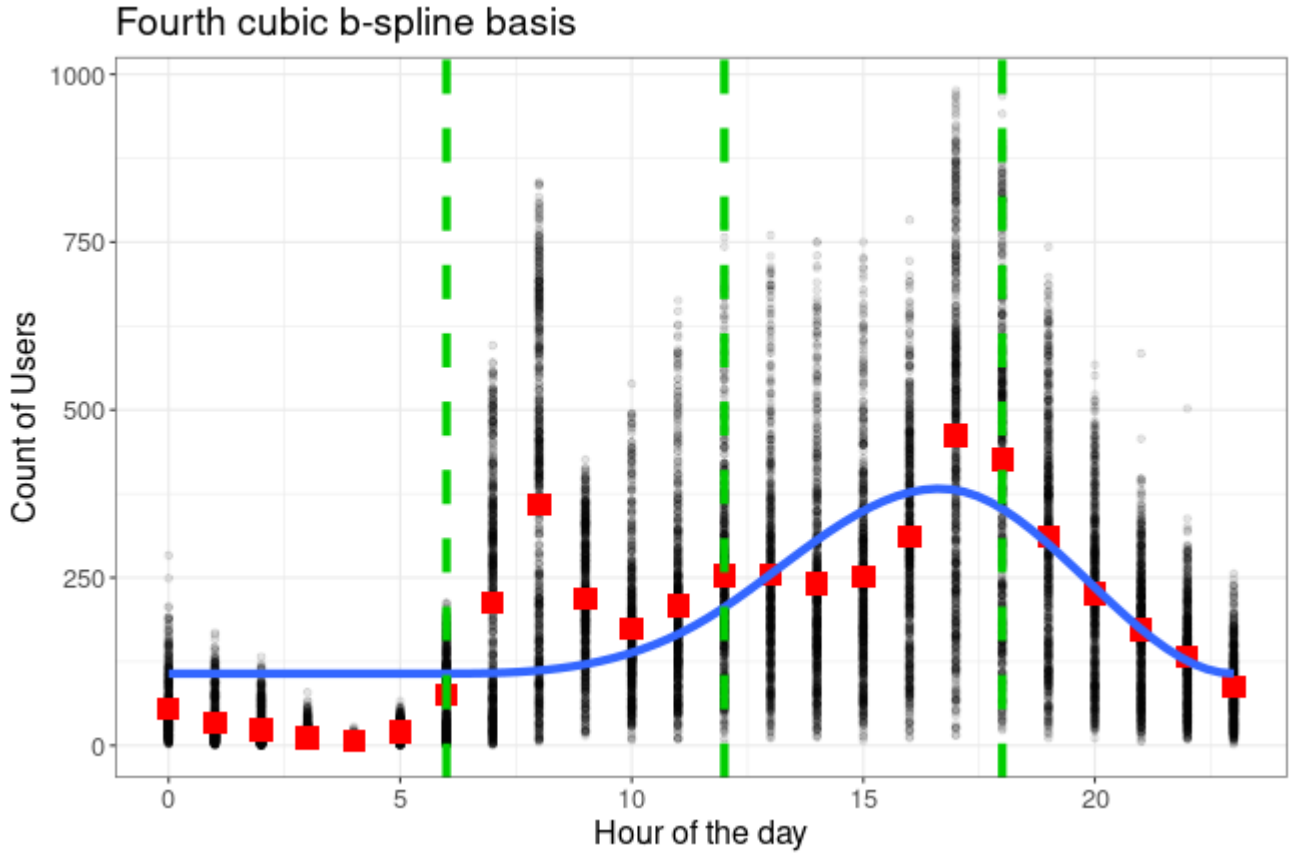
Splines

```
bike %>% ggplot(aes(x=hr, y=cnt)) + geom_point(alpha=.1) + geom_point(data=bike_by_hour, col="r
  xlab("Hour of the day") + ylab("Count of Users") + ggtitle("First cubic b-spline basis") + ge
  theme_bw() + theme(text = element_text(size = 16))
```



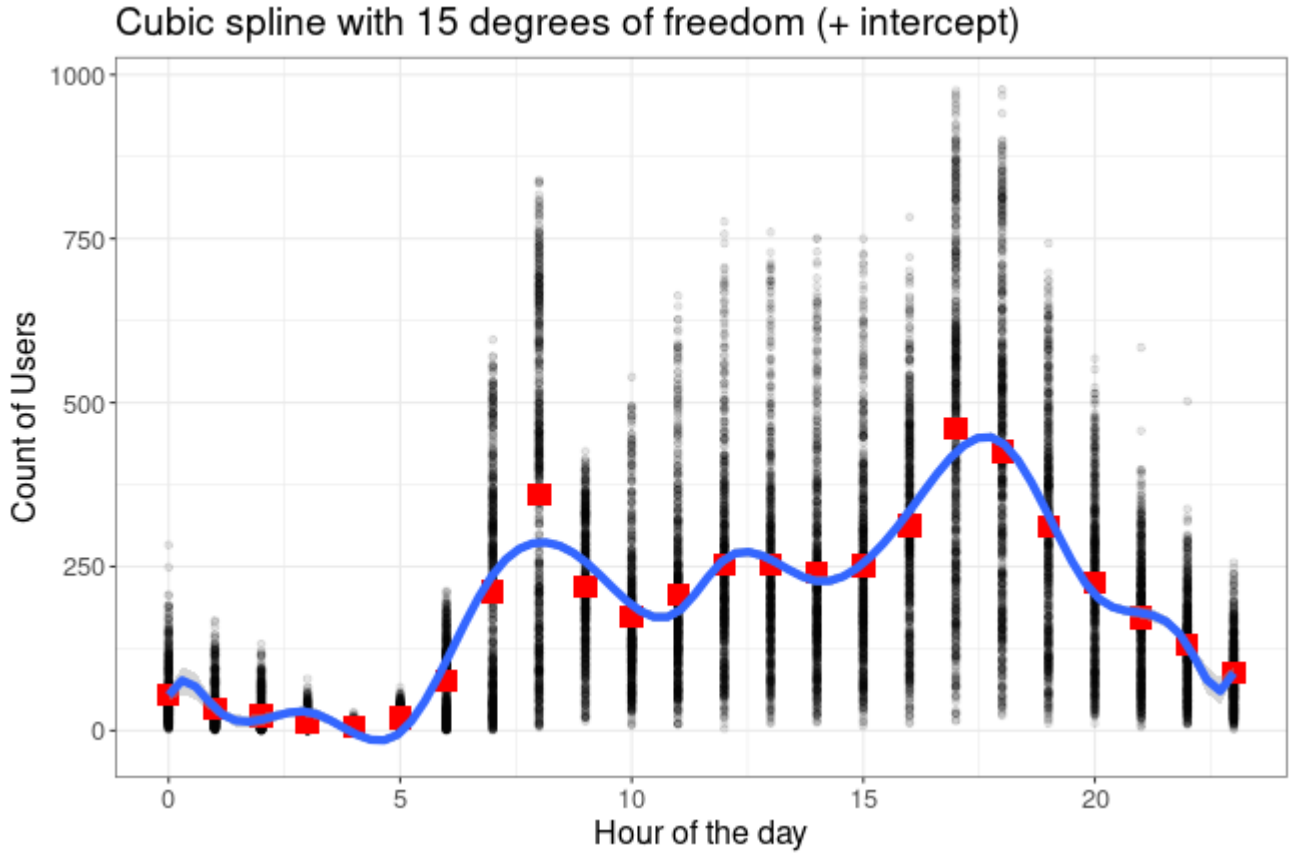
Splines

```
bike %>% ggplot(aes(x=hr, y=cnt)) + geom_point(alpha=.1) + geom_point(data=bike_by_hour, col="r
  xlab("Hour of the day") + ylab("Count of Users") + ggtitle("Fourth cubic b-spline basis") + g
  theme_bw() + theme(text = element_text(size = 16))
```



Splines

- Splines with many degrees of freedom have potentially weird behavior in areas with little data - "squiggly" interpolations



Smoothing splines

- What if we set the **maximum** possible number of knots
  - N total knots -- one for each observation
- Without penalization, we would have $N+4$ parameters for a cubic spline for N observations.
  - This is not feasible using conventional least-squares - the solution is underdetermined!
- What if we controlled the fit via some **penalty** parameter
  - All-else-equal, we'd prefer a fit where the regression function is not very "jumpy"
  - Formalize this in terms of the second-derivative $f^{\prime\prime}(x)$
- Our "smoothing spline" takes the form of a **penalized** optimization problem. We want to find the function $f(x)$ that minimizes:

$$\sum_{i=1}^n (Y_i - f(X_i))^2 + \lambda \int f^{\prime\prime}(t)dt$$

where λ is a non-negative "tuning" or "penalty" parameter.

Smoothing splines

- It turns out that the function \(f()\) that optimizes the "smoothing spline" objective function has some useful features
    1. It's a piecewise cubic polynomial
    2. It has knots at the unique values of the data \(x_1, x_2, \dotsc, x_N\)
    3. It has continuous first and second derivatives at each of the knots.
    4. It's linear outside of the knots
- It's a **natural cubic spline**
    - But with **penalized** parameter estimates (shrunken towards zero)
- \(\lambda\) chosen via cross-validation
    - Can conduct leave-one-out cross-validation very easily (formula exists to use the fit for all observations, so no need to re-fit)

Generalized Additive Models

- **Generalized Additive Models** (GAMs) allow us to extend the conventional multiple linear regression model to accomodate the non-linear transformations of \(X_i\).
- Instead of our original linear model:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \dotsc + \epsilon_{i}$$

- We fit:

$$Y_i = \beta_0 + f_1(X_{i1}) + f_2(X_{i2}) + \dotsc + \epsilon_{i}$$

- With conventional regression splines for \(f_1(), f_2(), f_3()\), etc..., this just becomes a giant linear regression with the spline bases substituted for the original regressors
- With *smoothing splines*, slightly more complicated - can't use OLS, but conventional software (`gam()` in R) implements the "backfitting" algorithm.
- Can extend to other functions \(f_i()\) such as local regressions or just plain polynomials

Generalized Additive Models

- Fitting our smoothing spline using the `gam()` function in the `mgcv` library

```
hour_fit <- gam(cnt ~ s(hr, bs="cr"), data = bike)
summary(hour_fit)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## cnt ~ s(hr, bs = "cr")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  189.463      0.992     191   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##       edf Ref.df    F p-value
## s(hr)   9      9 1785  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =   0.48   Deviance explained =   48%
## GCV =  17111  Scale est. = 17101     n = 17379
```

Generalized Additive Models

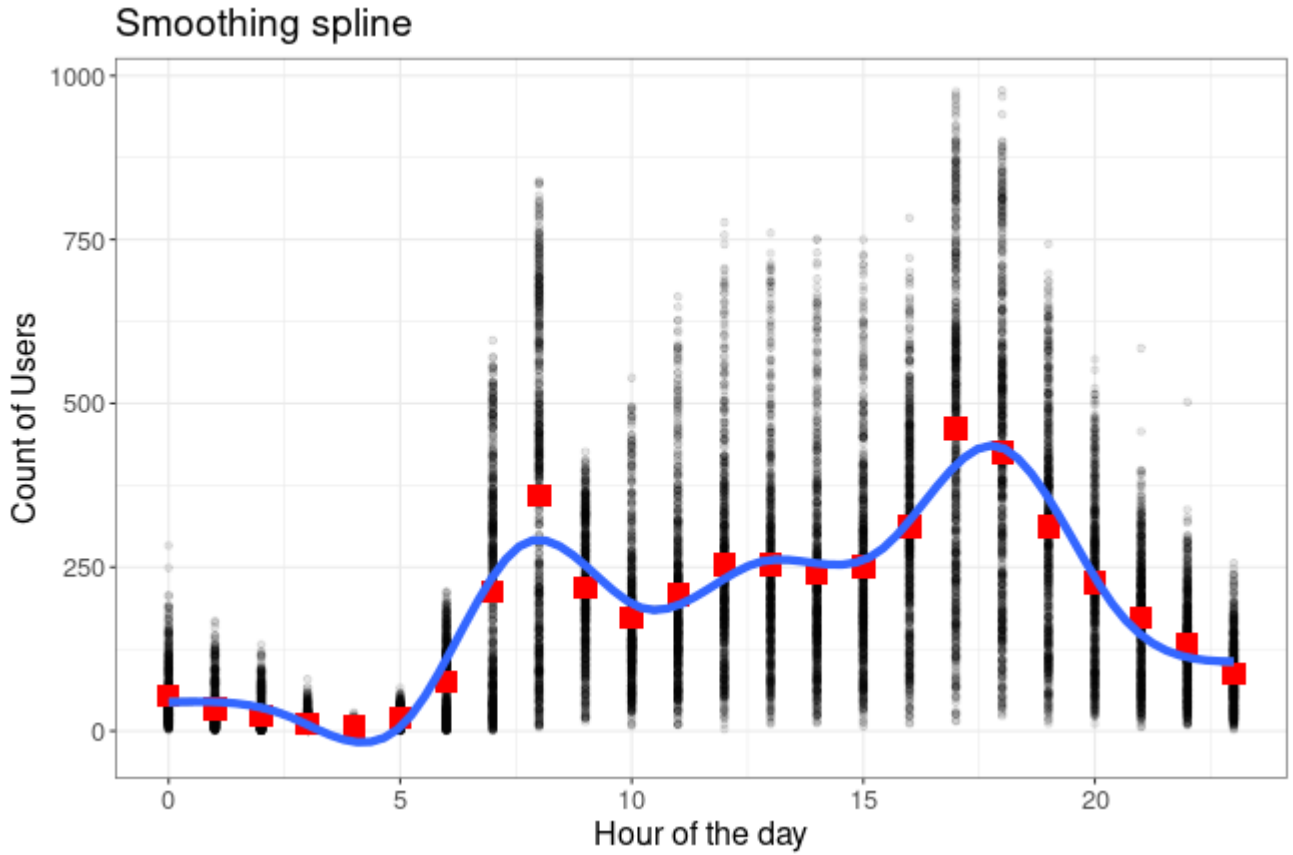
- We can combine "parametric" and "non-parametric" terms in the `gam()` function

```
hour_work_fit <- gam(cnt ~ workingday + s(hr, bs="cr"), data=bike)
summary(hour_work_fit)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## cnt ~ workingday + s(hr, bs = "cr")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   181.73       1.76  103.27  < 2e-16 ***
## workingday     11.33       2.13    5.32  1.1e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##       edf Ref.df    F p-value
## s(hr)   9      9 1787  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.481   Deviance explained = 48.1%
```

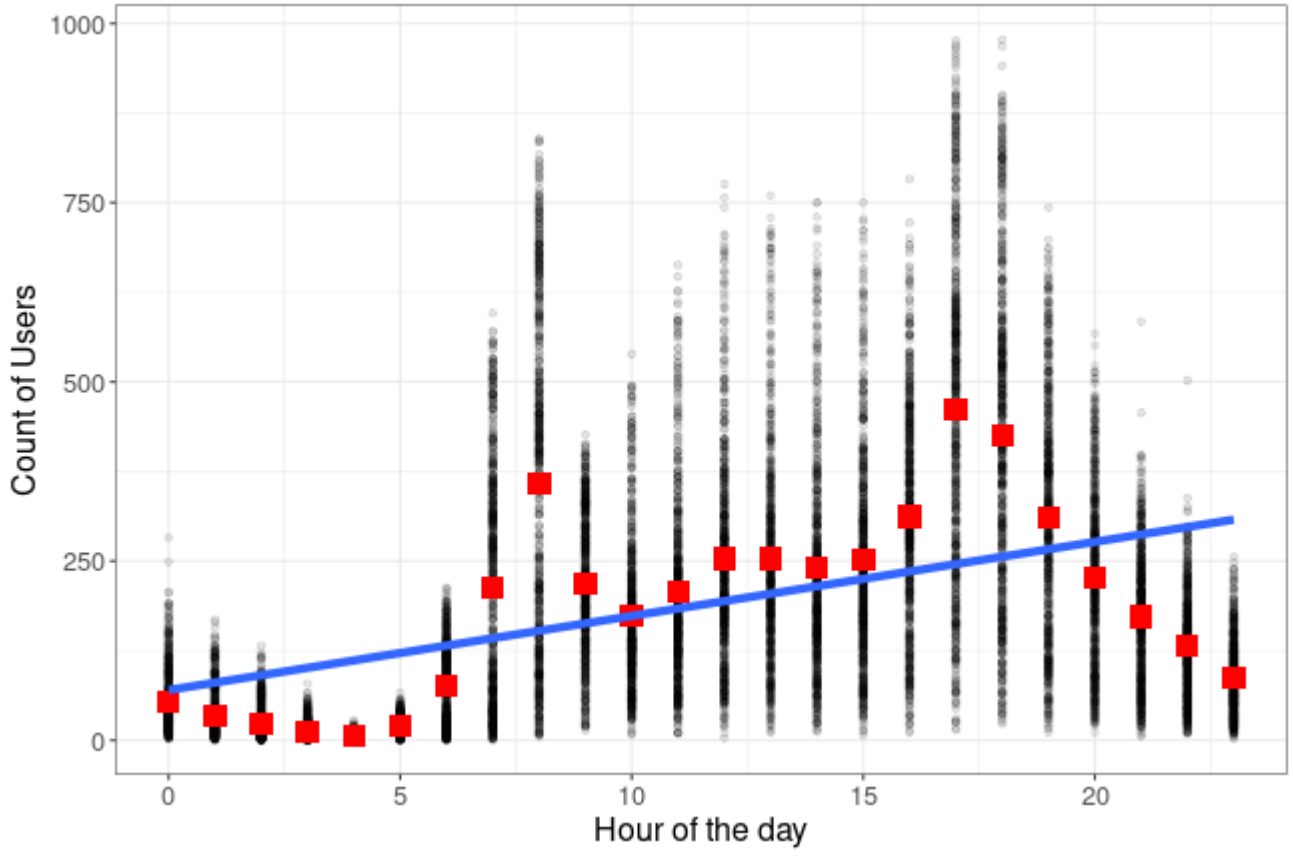
Generalized Additive Models

- Smoothing splines are also the default in `geom_smooth()` for large datasets

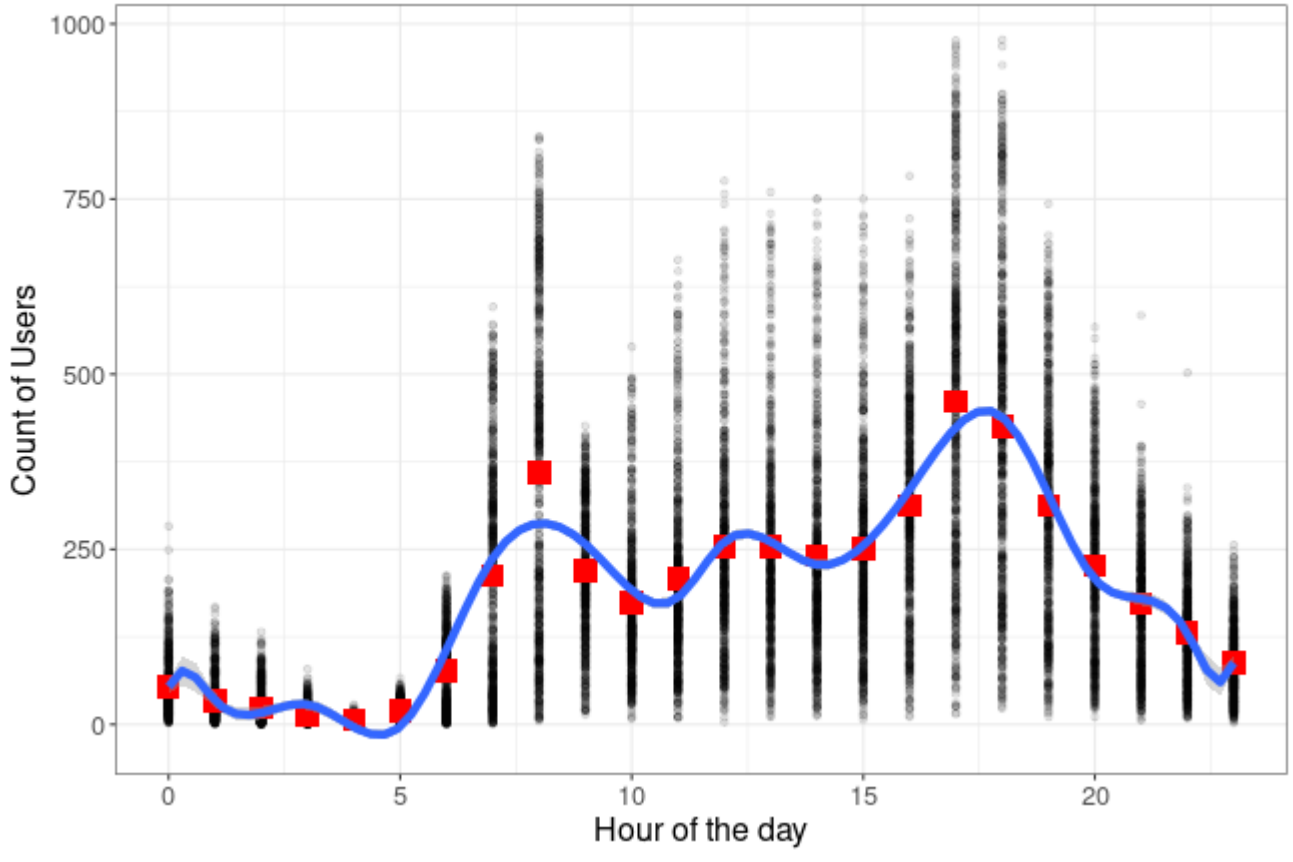


Regularization

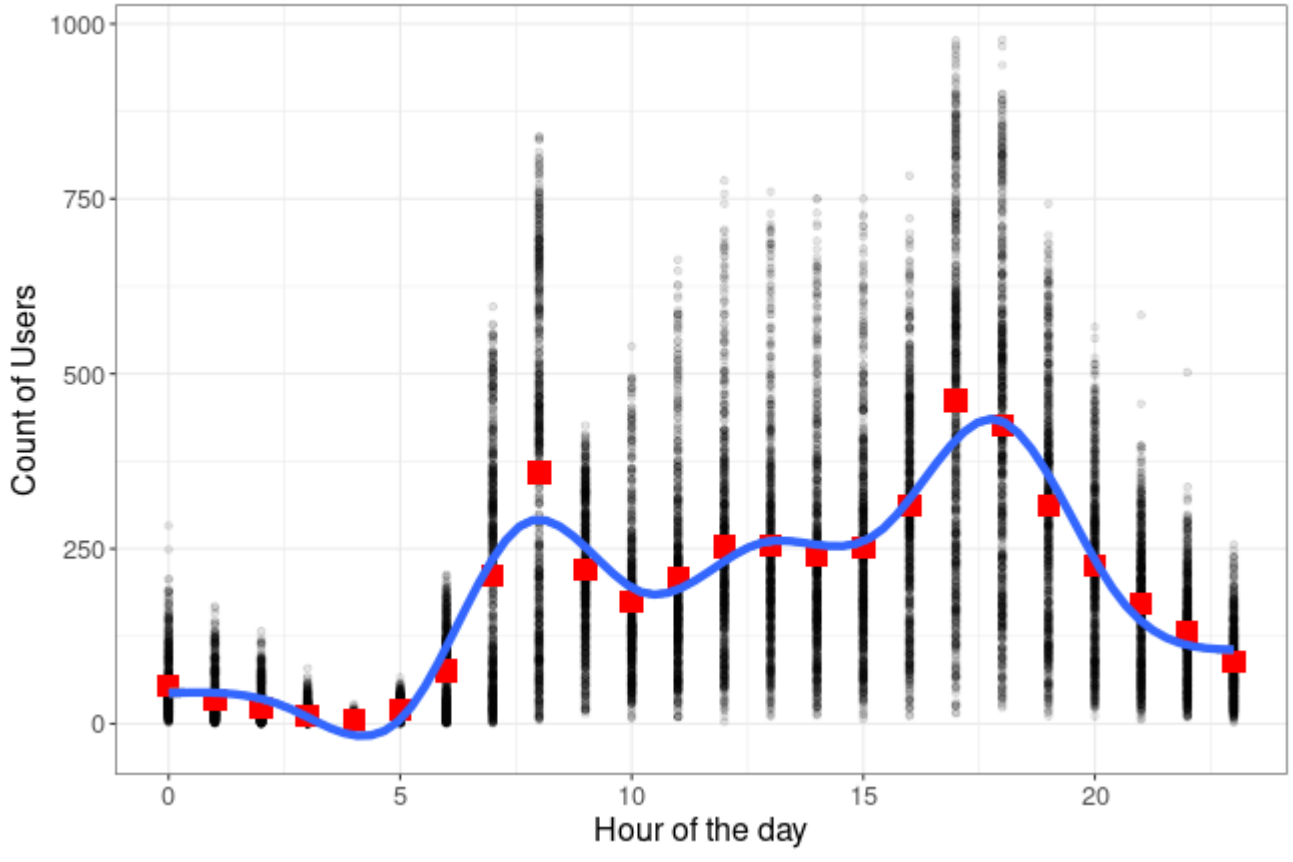
Regularization



Regularization



Regularization



Regularization

- The **error** of an estimator is a function of both its **bias** and its **variance**
  - We often talk about **unbiased** estimation as a goal of statistics?
  - Why? - Because we can accurately characterize the remaining error - the variance.
- But in other settings, we just want to know whether our point estimate is close to the truth
  - In this setting we might be willing to accept some **bias** in exchange for a reduction in variance.
- We've already done this to some extent:
  - Setting an **informative prior** in Bayesian regression setting pulls the estimates to our existing beliefs
  - Sharper prior \(\leadsto\) lower variance posterior distributions
- What sort of bias do we want to induce?
  - In a **regression**: bias the coefficients towards *zero*
  - Intuition: *betting on sparsity* - most regressors are irrelevant and their coefficients should be pulled towards zero.

Mean Squared Error

- Recall that the **mean squared error** of an estimator $\hat{\theta}$ of some parameter θ can be decomposed into the (squared) **bias** and the **variance**

$$E\bigg[(\hat{\theta} - \theta)^2\bigg] = E\bigg[\theta^2\bigg] - 2\theta E\bigg[\hat{\theta}\bigg] + \theta^2$$

- Add and subtract $E[\hat{\theta}]^2$

$$E\bigg[(\hat{\theta} - \theta)^2\bigg] = \bigg(E[\theta^2]- E[\hat{\theta}]^2\bigg) + E[\hat{\theta}]^2 - 2\theta E[\hat{\theta}] + \theta^2$$

- And factoring to get the bias squared

$$\underbrace{E\bigg[(\hat{\theta} - \theta)^2\bigg]}_{\text{MSE}} = \underbrace{\bigg(E[\theta^2]- E[\hat{\theta}]^2\bigg)}_{\text{Variance}} + \underbrace{\bigg(E[\hat{\theta} - \theta]\bigg)^2}_{\text{Squared Bias}}$$

Notation notes

- We're going to work with a standard regression setting with outcome Y_i and covariate matrix $\mathbf{X}$.
  - We'll assume the covariates are **standardized** to have mean zero and variance 1 (we don't want our regularizer to depend on the scale of the inputs).
- Let $||\cdot||_p$ denote the L_p norm.
  - The L_2 norm is the **euclidean norm**: $|| \beta ||_2 = \sqrt{\sum_{k=1}^K |\beta|^2}$
  - The L_1 norm is the sum of the absolute values $|| \beta ||_1 = \sum_{k=1}^K |\beta|$
  - More generally, the L_p norm is: $|| \beta ||_p = \bigg(\sum_{k=1}^K |\beta|^p\bigg)^{\frac{1}{p}}$

Example: Predicting Peace Agreements

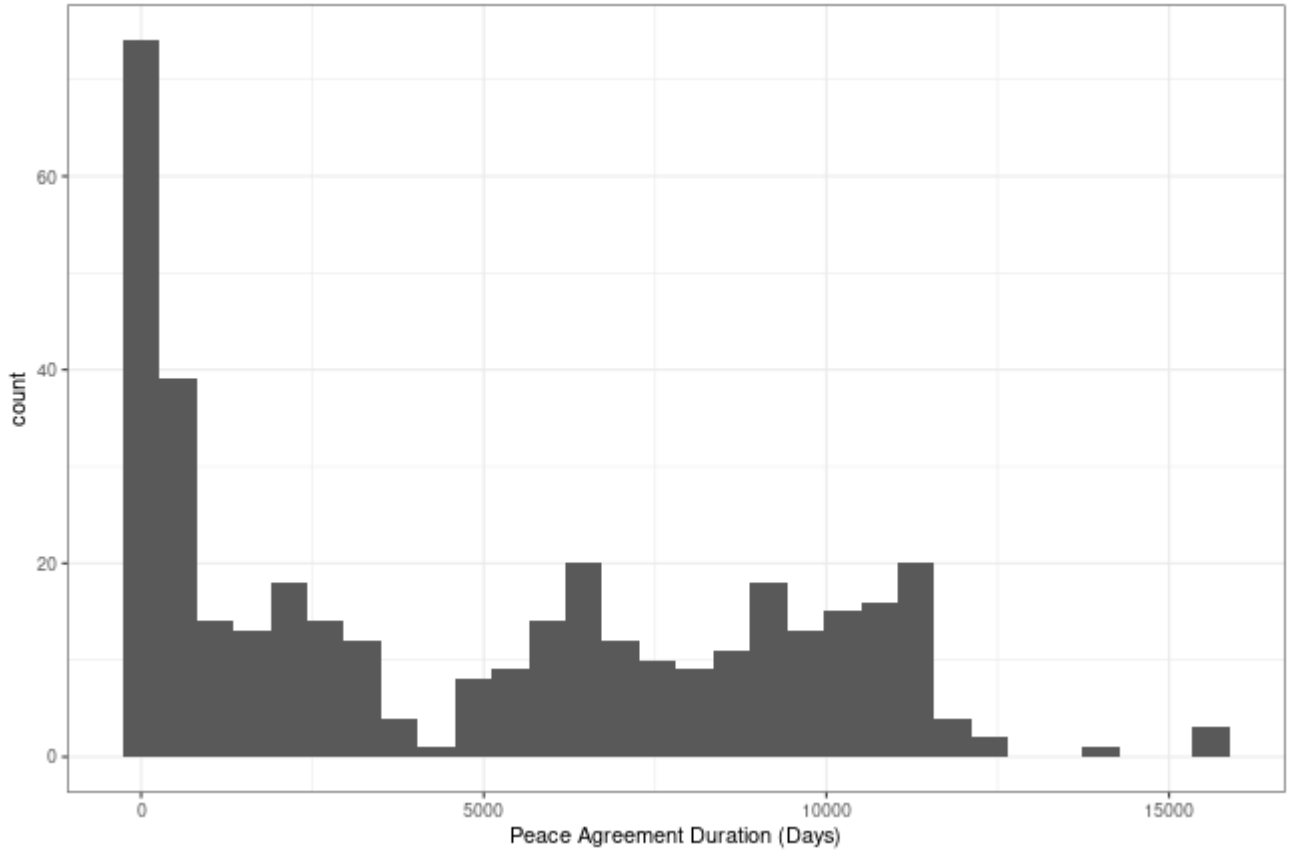
- Many predictive involve **large numbers of covariates** relative to the total number of observations
  - If all of the covariates are in some sense "relevant" then you're probably out of luck.
  - But in most settings, only a handful of the covariates actually matter...the problem is that you don't know which ones ex-ante.
- The **UCDP Peace Agremeents dataset** contains information on peace agreements to armed conflicts from the period 1975-2021
  - We may want to know what factors predict durable peace agreements.

```
pa <- read_csv("data/ucdp-peace-agreements-221.csv")
```

- Some brief pre-processing of the outcome

```
# If agreement remains, then duration = "now"
pa$duration_complete <- pa$duration
pa$duration[is.na(pa$duration)] <- ymd("2021-12-31")
pa$durationDays <- as.numeric(pa$duration - pa$pa_date)
# Fill in missing "ended' data
pa$ended_full <- as.numeric(pa$ended)
pa$ended_full[is.na(pa$ended_full)] <- 0
```

Example: Predicting Peace Agreements



Example: Predicting Peace Agreements

- Our covariates are a large number of coded features for each agreement

```
pa_x <- model.matrix(durationDays ~ as.factor(region) + year + as.factor(incompatibility) +
                       cease + intarmy + ddr + withd + mil_prov + pp + intgov + intciv +
                       elections + interim + natalks + shagov + pol_prov + aut + fed + ind +
                       ref + shaloc + regdev + cul + demarcation + locgov + terr_prov + amn +
                       pris + recon + return + justice_prov + reaffirm + outlin + pko + gender
                       co_impl + termdur + txt + as.factor(inclusive) + as.factor(pa_type) +
                       as.factor(out_iss) + as.factor(frame), data=pa)[,-1]
```

- We have 55 covariates across 374 observations.
  - We *can* get an OLS estimate, but our sense is that these will be incredibly noisy.

Ridge regression

- Our typical (gaussian) linear regression estimator finds the \(\beta\) that **minimizes** the residual sum of squares (RSS)

$$\text{RSS} = \sum_{i=1}^N (Y_i - X_i^{\prime}\beta)^2$$

- For non-gaussian models, replace the residual sum of squares with the log-likelihood (the RSS *is* the negative log-likelihood for a gaussian)

- **Ridge regression** augments the objective function by adding a penalty term

$$\sum_{i=1}^N (Y_i - X_i^{\prime}\beta)^2 + \lambda\sum_{k=1}^K \beta_k^2$$ where \(\lambda\) is a user-specified penalty parameter

- Note that we'll typically avoid regularizing the **intercept** \(\beta_0\).

Bayesian interpretation

- Ridge regression has an interpretation in terms of a **bayesian regression**
- Consider our normal model with:

$$Y_i |X_i, \beta \sim \text{Normal}(X_i^{\prime}\beta, \sigma^2)$$

- The log-posterior can be written as

$$\log p(\beta | \mathbf{X}, Y) \propto \log p(Y | \mathbf{X}, \beta) + \log p(\beta)$$

- In a gaussian model, the first part is just the (negative) residual sum of squares

$$\log p(\beta | \mathbf{X}, Y) \propto - \frac{1}{2\sigma^2}\sum_{i=1}^N (Y_i - X_i^{\prime}\beta)^2 + \log p(\beta)$$

Bayesian interpretation

- Now, suppose we put a conventional normal prior on β

$$\beta \sim \text{Normal}(0, \tau^2I)$$

- $\log p(\beta)$ can then be written as:

$$\begin{align*}\log p(\beta) &= -\frac{1}{2\tau^2} \sum_{k=1}^K (\beta_k - 0)^2\\ &= -\frac{1}{2\tau^2} \sum_{k=1}^K \beta_k^2\end{align*}$$

- Plugging back in to the log-posterior, we have

$$\log p(\beta | \mathbf{X}, Y) \propto - \frac{1}{2\sigma^2}\sum_{i=1}^N (Y_i - X_i^{\prime}\beta)^2 - \frac{1}{2\tau^2} \sum_{k=1}^K \beta_k^2$$

Bayesian interpretation

- Rearranging terms and dropping scaling constants, we have

$$\log p(\beta | \mathbf{X}, Y) \propto - \sum_{i=1}^N (Y_i - X_i^{\prime}\beta)^2 - \frac{\sigma^2}{\tau^2} \sum_{k=1}^K \beta_k^2$$

- With \(\lambda = \frac{\sigma^2}{\tau^2}\), this is just the negative of the ridge regression objective function
  - Maximizing the log-posterior (finding the MAP estimate) is equivalent to minimizing a ridge regression objective!

Visualizing the penalty

- Penalized regression can also be interpreted as a **constrained optimization problem**
- Find the \(\beta\) that minimizes:

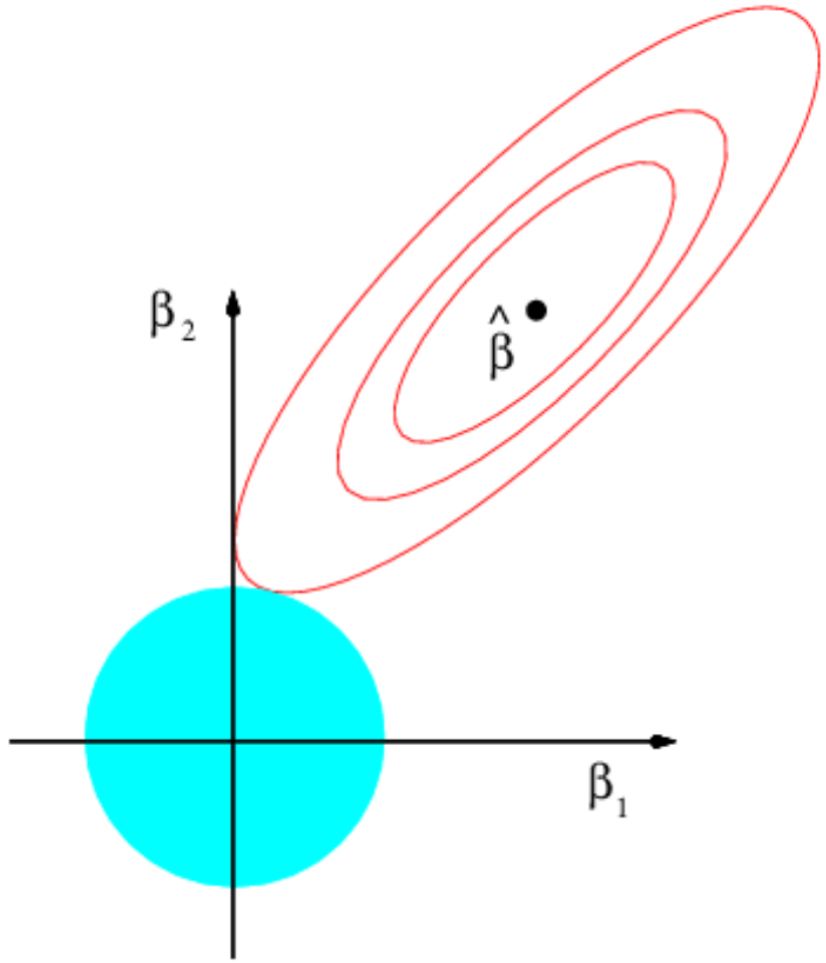
$$\text{RSS} = \sum_{i=1}^N (Y_i - X_i^{\prime}\beta)^2$$

- Subject to

$$\sum_{k=1}^K \beta_k^2 \le s$$

- We prefer the unconstrained formulation with \(\lambda\) for computation, but the constrained view allows us to get a sense of how the ridge regularizer is operating.

Visualizing the penalty



Example: Predicting Peace Agreements

- The standard library for estimating penalized regressions in R is `glmnet`
  - By default, R will fit the model for a **regularization sequence** of λ
  - Actually choosing a λ is done via cross-validation

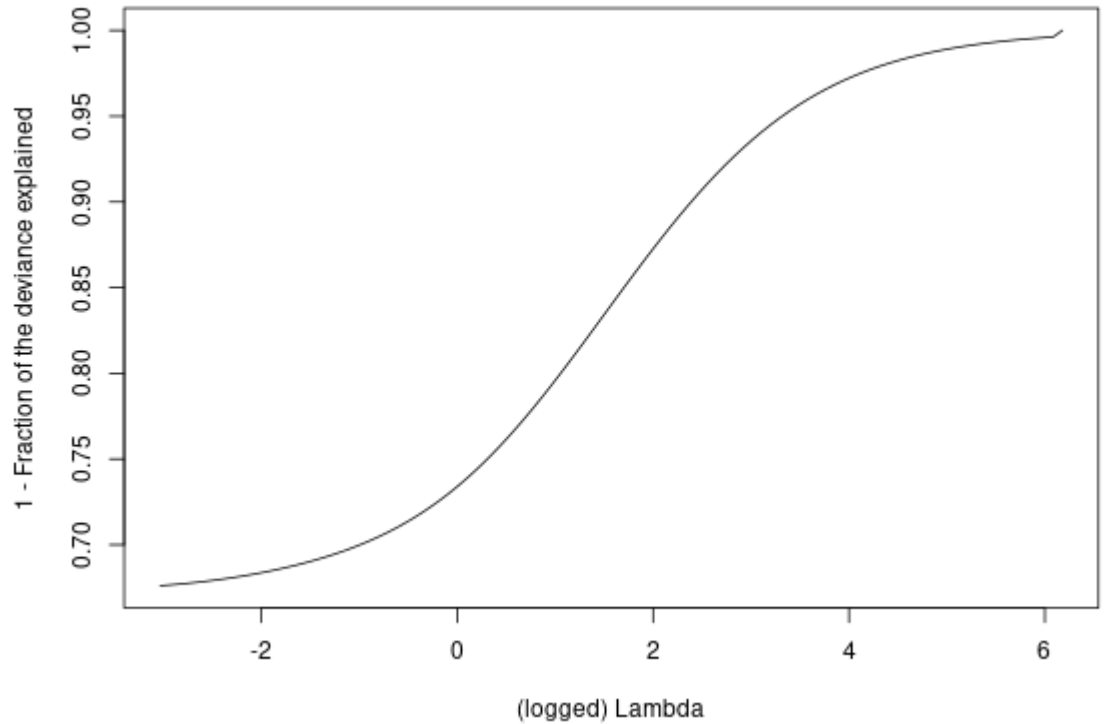
- Let's fit a gaussian model (we could do poisson as well, but gaussian is easy)

```
# alpha=0 is the ridge penalty
conflict_ridge <- glmnet(x=pa_x, y=log(pa$durationDays), family="gaussian", alpha=0, standardiz
```

Example: Predicting Peace Agreements

- How is the in-sample fit?

```
plot(x=log(conflict_ridge$lambda), y=1-conflict_ridge$dev.ratio, type='l', xlab="(logged) Lambd
```



Selecting the penalty

- \(\lambda\) is typically selected via cross-validation

```
conflict_ridge_cv <- cv.glmnet(x=pa_x, y=log(pa$durationDays), family="gaussian", alpha=0, star
print(conflict_ridge_cv)
```

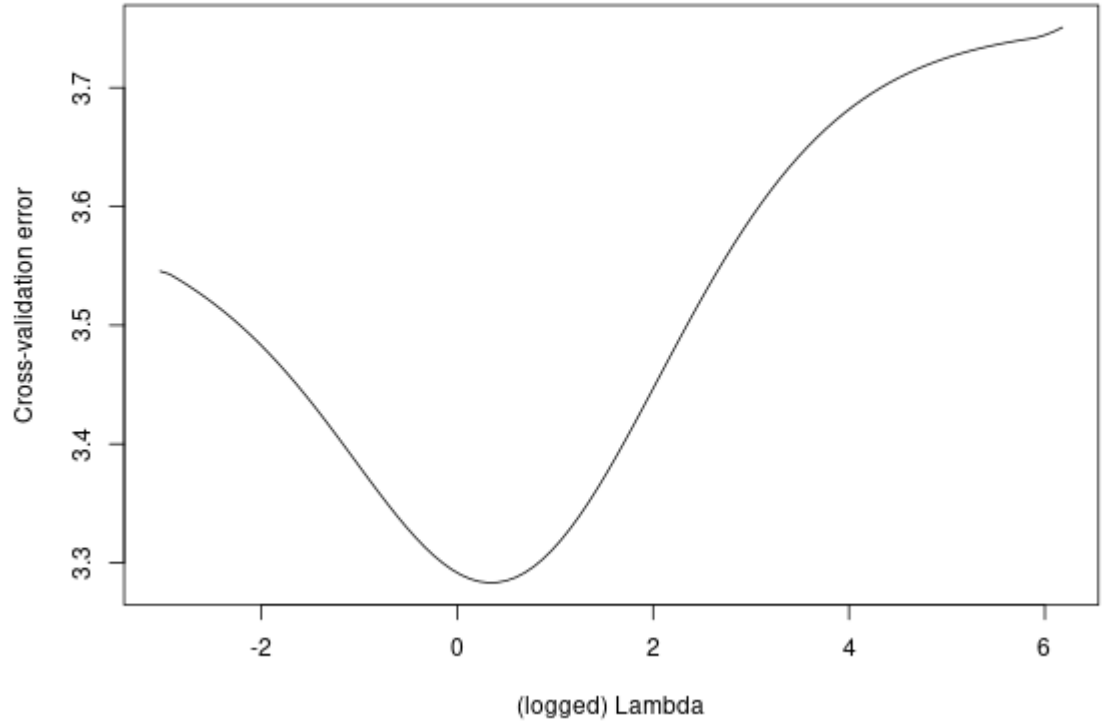
```
##
## Call:  cv.glmnet(x = pa_x, y = log(pa$durationDays), family = "gaussian",      alpha = 0, standardize
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure    SE Nonzero
## min   1.37    64    3.28 0.182      54
## 1se   8.04    45    3.46 0.203      54
```

- `cv.glmnet` typically provides two choices of \(\lambda\)
  - The first being the value that minimizes the cross-validation error (`min`)
  - The second being the largest value of \(\lambda\) that is still within 1 standard error from the minimum (`1se`)

Selecting the penalty

- Cross-validation error is not strictly increasing with \(\lambda\) - low penalties can lead to **bad** out-of-sample fit!

```
plot(x=log(conflict_ridge_cv$lambda), y=conflict_ridge_cv$cvm, type='l', xlab="(logged) Lambda'
```



Lasso

- **Ridge regression** regularizes the coefficients towards zero, but it does not guarantee that some coefficients are exactly zeroed out.
  - In other words, it does not do **model selection**
- We would like to have a penalty term that results in **sparse** solutions - ones where many coefficients are explicitly set to 0
  - The L_1 penalty accomplishes this!
  - The **least absolute shrinkage and selection operator** or lasso is the same penalized regression as the ridge regression but with the sum of the **absolute** rather than squared coefficients in the penalty $$\sum_{i=1}^N (Y_i - X_i^{\prime}\beta)^2 + \lambda\sum_{k=1}^K |\beta_k|$$
- The L_1 norm is non-differentiable with respect to the covariates at 0, which leads to the lasso acting as a "selection" operator on the elements of β

Visualizing the lasso

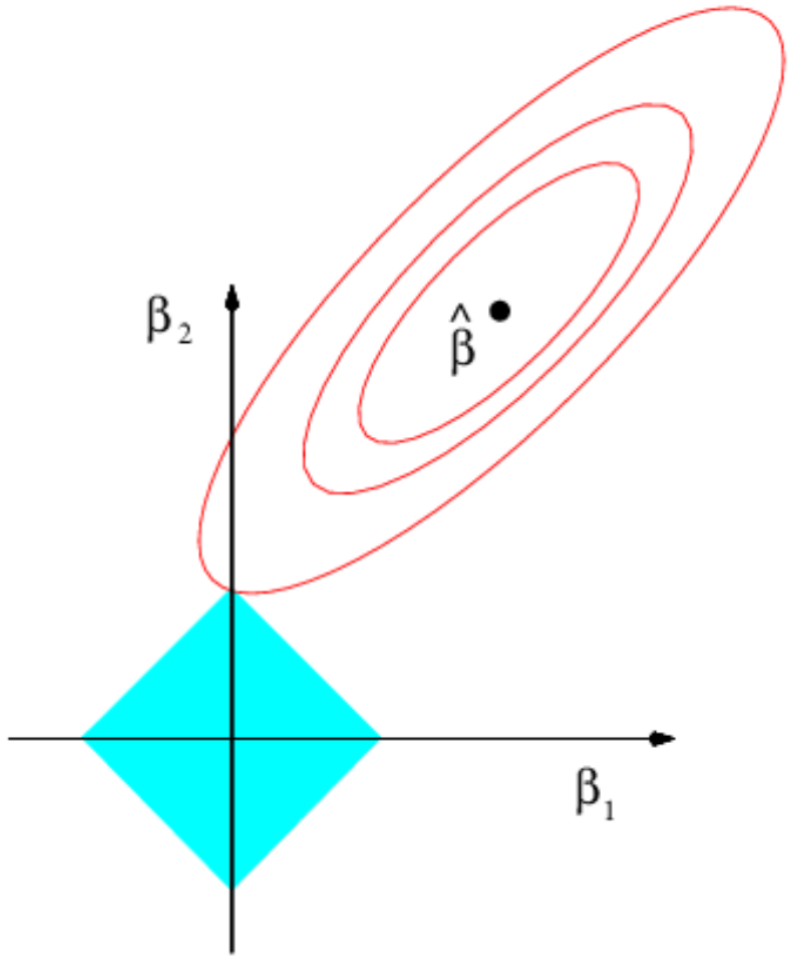
- As with ridge, the lasso has an expression in terms of a constrained optimization problem:
- Find the \(\beta\) that minimizes:

$$\text{RSS} = \sum_{i=1}^N (Y_i - X_i^{\prime}\beta)^2$$

- Subject to

$$\sum_{k=1}^K |\beta_k| \le s$$

Visualizing the lasso



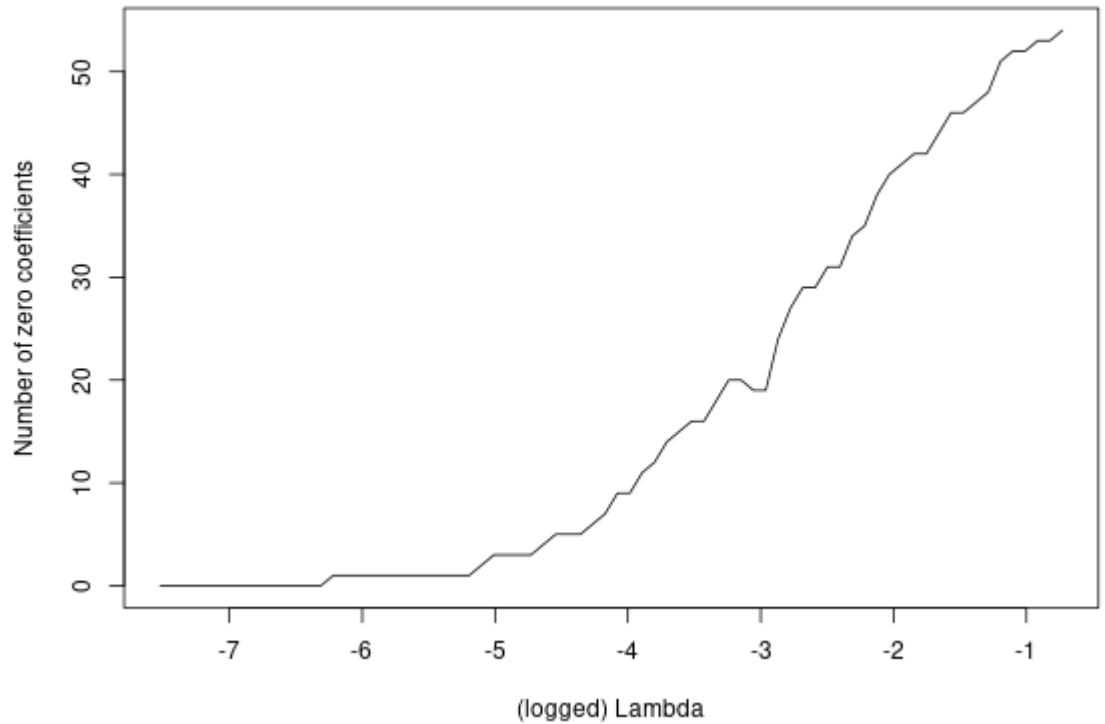
Example: Predicting Peace Agreements

```
# alpha=1 is the lasso penalty
conflict_lasso <- glmnet(x=pa_x, y=log(pa$durationDays), family="gaussian", alpha=1)
```

Example: Predicting Peace Agreements

- We can count the number of coefficients set to \(0\) as \(\lambda\) increases

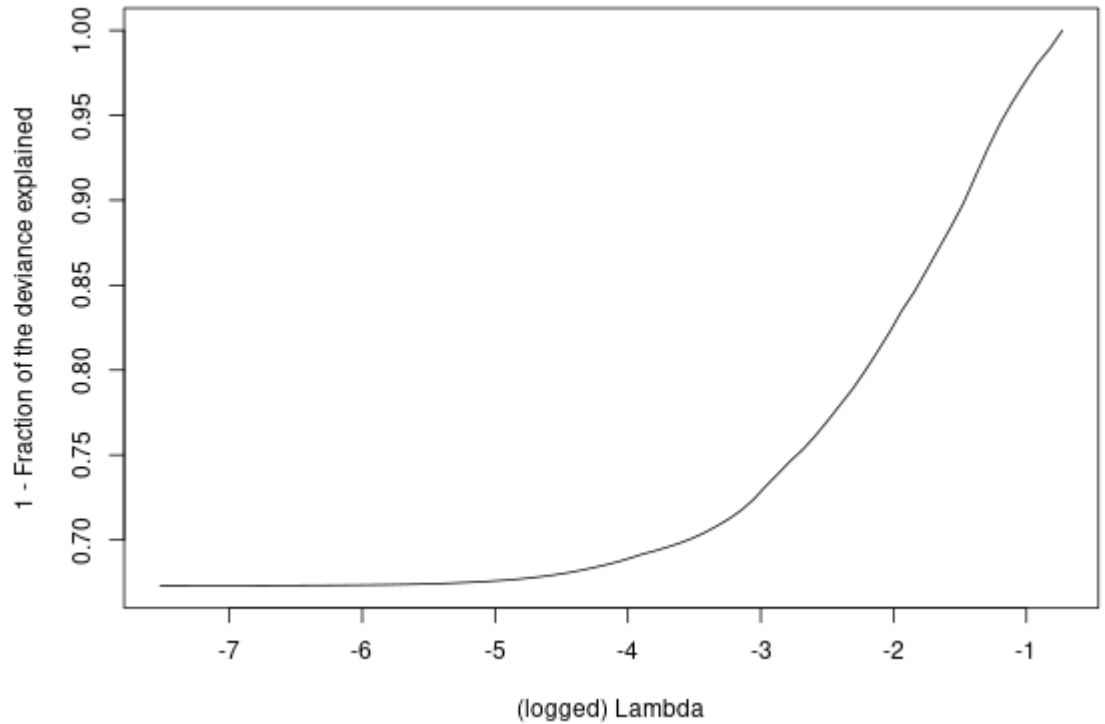
```
plot(x=log(conflict_lasso$lambda), y=colSums(coef(conflict_lasso) == 0), type='l', xlab="(logge
```



Example: Predicting Peace Agreements

- In-sample fit improves as we reduce \(\lambda\)

```
plot(x=log(conflict_lasso$lambda), y=1- conflict_lasso$dev.ratio, type='l', xlab="(logged) Lamb
```



Example: Predicting Peace Agreements

- We again choose \(\lambda\) via cross-validation

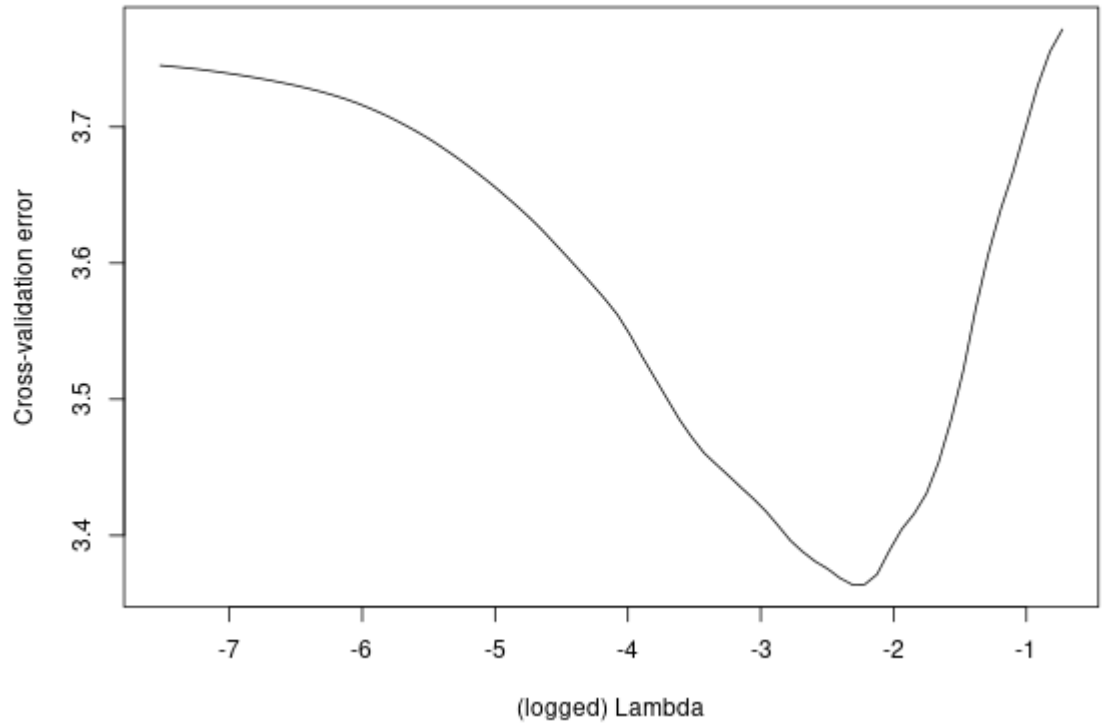
```
conflict_lasso_cv <- cv.glmnet(x=pa_x, y=log(pa$durationDays), family="gaussian", alpha=1)
print(conflict_lasso_cv)
```

```
##
## Call:  cv.glmnet(x = pa_x, y = log(pa$durationDays), family = "gaussian",      alpha = 1)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure    SE Nonzero
## min 0.0992    18    3.36 0.240      20
## 1se 0.2515     8    3.57 0.264       7
```

Example: Predicting Peace Agreements

- Cross-validation error does not strictly improve with lower \(\lambda\)

```
plot(x=log(conflict_lasso_cv$lambda), y=conflict_lasso_cv$cvm, type='l', xlab="(logged) Lambda'
```



Lasso as variable selection

- **Variable selection** is a common problem in statistics
  - We assume a world where among our covariates \(X\), some small subset have actual non-zero coefficients (they "matter") but most don't.
  - The problem is that ex-ante, we have no idea what the relevant covariates actually are!
- Classical approaches
  - **Forward selection** - Start with an empty model and add variables one-by-one.
  - **Backward selection** - Start with a full model and remove the least correlated variables one-by-one.
- The lasso can be thought of as an alternative to these techniques
  - Advantages in computational time/feasibility as well as potential **predictive** benefits from shrinkage.

Example: Predicting Peace Agreements

- Let's fit the lasso with the "best" cross-validated \(\lambda\)

```
conflict_lasso_best <- glmnet(x=pa_x, y=log(pa$durationDays), family="gaussian", alpha=1, lambd
```

- Which coefficients are non-zero?

```
coef(conflict_lasso_best)[,1][coef(conflict_lasso_best)[,1] != 0]
```

```
##           (Intercept)    as.factor(region)2    as.factor(region)4
##               7.72511               -0.27508               -0.47508
##                cease                intciv               natalks
##              -0.20605                0.03974               -1.14074
##                   ref                   amn                  pris
##               0.64016                0.02579               -0.05113
##                return               reaffirm                gender
##               0.19774                0.39678               -0.17827
##               co_impl               termdur                   txt
##               0.04615                0.00393                0.27287
## as.factor(inclusive)2   as.factor(out_iss)1   as.factor(out_iss)3
##              -0.18618                0.11482               -0.61865
##   as.factor(out_iss)4   as.factor(out_iss)5     as.factor(frame)2
##              -0.00594               -0.76880                0.08045
```

Example: Predicting Peace Agreements

- What if we took the "selected" variables and just ran OLS?

```
pa_x_use <- cbind(1, pa_x[,names(coef(conflict_lasso_best)[,1][coef(conflict_lasso_best)[,1] !=
beta_ols <- as.vector(solve(t(pa_x_use)%*%pa_x_use)%*%t(pa_x_use)%*%log(pa$durationDays))
names(beta_ols) <- names(coef(conflict_lasso_best)[,1][coef(conflict_lasso_best)[,1] != 0])
beta_ols
```

```
##            (Intercept)    as.factor(region)2    as.factor(region)4
##                7.65115              -0.89269              -0.69743
##                 cease                intciv                natalks
##               -0.36180               0.26985              -1.33445
##                    ref                   amn                  pris
##                0.95699               0.26881              -0.45066
##                 return              reaffirm                gender
##                0.44617               0.62609              -0.56304
##                co_impl               termdur                   txt
##                0.23087               0.00844               0.64263
## as.factor(inclusive)2   as.factor(out_iss)1   as.factor(out_iss)3
##               -0.34966               0.17245              -0.98263
##   as.factor(out_iss)4   as.factor(out_iss)5    as.factor(frame)2
##               -0.17640              -1.12314               0.35385
```

- The OLS coefficients post-selection are larger in magnitude!
  - Why? Lasso induces a **bias** towards zero