Regression and regularization

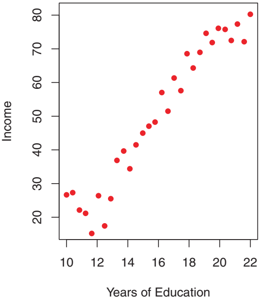
Jake Hofman

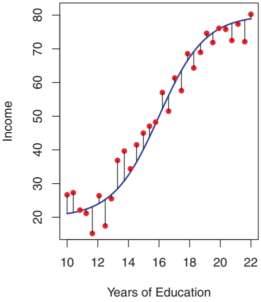
Microsoft Research

March 16, 2018

Regression

Regression is a *supervised* learning task by which we aim to *predict a real-valued outcome* for an example given its features



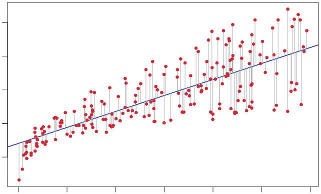


e.g., predict someone's income given their education

Linear regression

We'll use a **linear model** to make predictions $\hat{y}$ given features x:

$$\hat{y} = mx + b$$



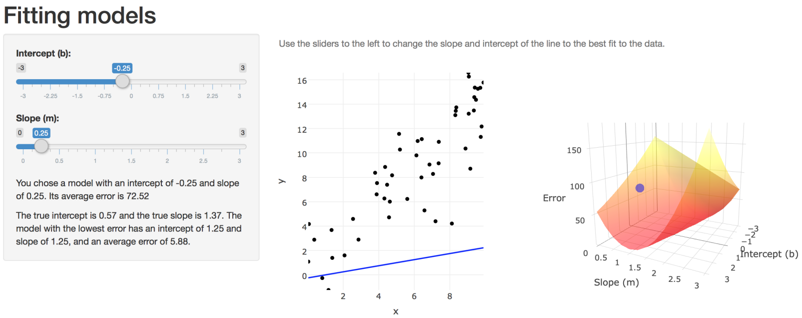
And we'll measure the **mean squared error** between the predicted and actual value of each observation:

$$\mathrm{MSE(m, b)} = \sum_i (\hat{y}_i - y_i)^2 = \sum_i (mx_i + b - y_i)^2$$

Linear regression

We'd like the **slope** m and **intercept** b with the **lowest error**:

$$(\hat{b}, \hat{m}) = \arg\min_{(b,m)} \mathrm{MSE}(\mathrm{m}, \mathrm{b}) = \sum_i (mx_i + b - y_i)^2$$

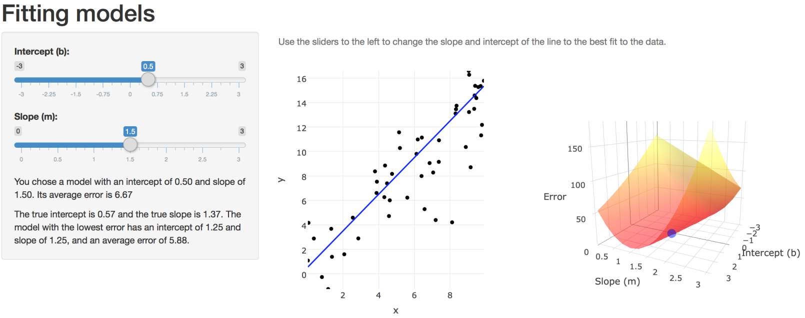


With just two parameters, we can manually search for the best fit

Linear regression

We'd like the **slope** m and **intercept** b with the **lowest error**:

$$(\hat{b}, \hat{m}) = \arg\min_{(b,m)} \mathrm{MSE(m, b)} = \sum_i (mx_i + b - y_i)^2$$



With just two parameters, we can manually search for the best fit

Linear regression

We'd like the **slope** m and **intercept** b with the **lowest error**:

$$(\hat{b}, \hat{m}) = \arg\min_{(b,m)} \mathrm{MSE(m, b)} = \sum_i (mx_i + b - y_i)^2$$

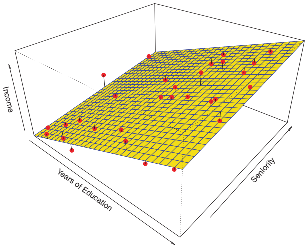
But notice the above is quadratic in m and b, so we can solve for the exact minimum:

$$\begin{aligned} \hat{m} &= \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} \\ \hat{b} &= \bar{y} - \hat{m}\bar{x} \end{aligned}$$

Multiple linear regression

We can extend this to making predictions $\hat{y}$ from multiple features $x_1, x_2, \ldots$:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_K x_K = \beta \cdot x$$



e.g., predict income given education and time at company

Multiple linear regression

We can extend this to making predictions $\hat{y}$ from multiple features $x_1, x_2, \ldots$:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_K x_K = \beta \cdot x$$

And there's still a closed form solution:

$$\hat{\beta} = \arg\min_{\beta} \sum_i (\beta \cdot x_i - y_i)^2 = (X^T X)^{-1} X^T y$$

Where:

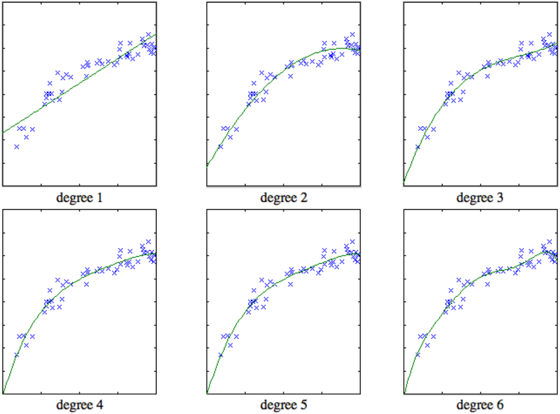
$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1p} \\ x_{21} & x_{22} & \ldots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{np} \end{pmatrix}.$$

Multiple linear regression

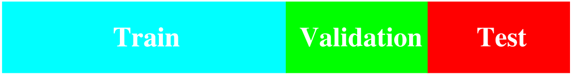
We can even use **non-linear features**, for instance to fit a polynomial:

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_K x^K$$

So **what degree polynomial** should we pick?

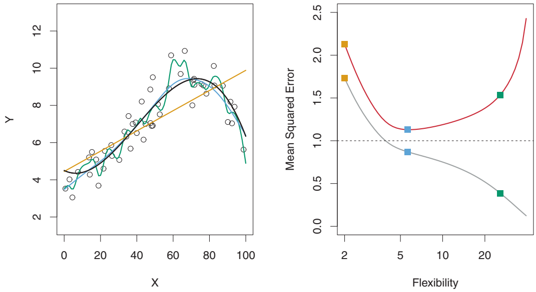


Cross-validation



- Randomly split our data into three sets
- For each polynomial degree k:
  - Fit a model to the training set
  - Evaluate on the validation set
- Select the model with the lowest validation error
- Quote final performance of this model on the test set

Cross-validation



- Randomly split our data into three sets
- For each polynomial degree k:
  - Fit a model to the training set
  - Evaluate on the validation set
- Select the model with the lowest validation error
- Quote final performance of this model on the test set

Regularization

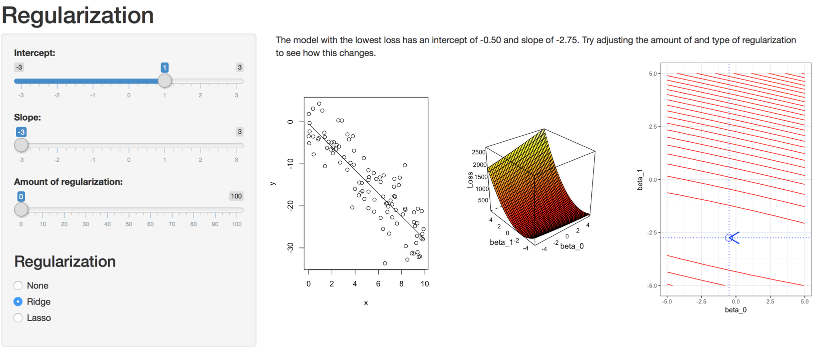
Instead of looping over many models and picking the best, we can fit one big model, balancing model fit and complexity

$$\hat{\beta} = \arg\min_{\beta} \underbrace{MSE(\beta)}_{\text{fit to the training data}} + \lambda \underbrace{||\beta||^2}_{\text{“complexity” of the model}}$$

λ controls the tradeoff between fit and complexity

Ridge regression

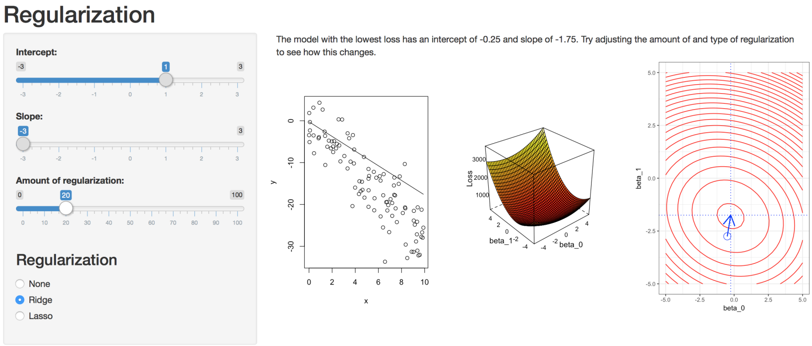
When λ is small, this is just ordinary regression



$$\hat{\beta} = \arg\min_{\beta} \underbrace{MSE(\beta)}_{\text{fit to the training data}} + \lambda \underbrace{||\beta||^2}_{\text{“complexity” of the model}}$$

Ridge regression

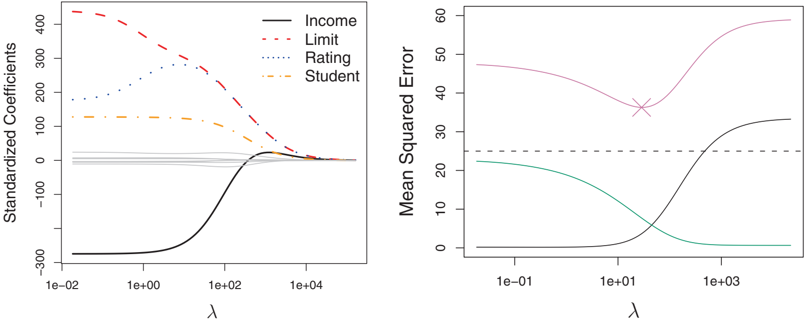
When λ is large, we "shrink" coefficients towards zero



$$\hat{\beta} = \arg\min_{\beta} \underbrace{MSE(\beta)}_{\text{fit to the training data}} + \lambda \underbrace{||\beta||^2}_{\text{“complexity” of the model}}$$

Ridge regression

Regularization improves predictions by trading variance for bias



But we still need cross-validation to determine the best tradeoff

Ridge regression

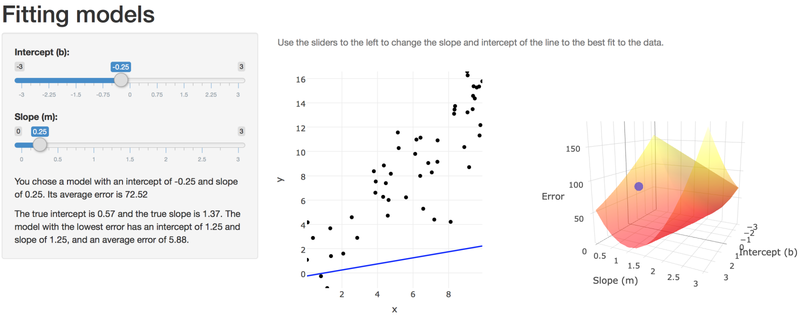
Regularization also lets us fit large models with more parameters than observations

$$
\begin{aligned}
\hat{\beta} &= \arg\min_{\beta} MSE(\beta) + \lambda ||\beta||^2 \\
&= \arg\min_{\beta} \sum_i (\beta \cdot x_i - y_i)^2 + \lambda \sum_k ||\beta_k||^2 \\
&= (X^T X + \lambda)^{-1} X^T y
\end{aligned}
$$

But what if we can't compute this solution in closed form?

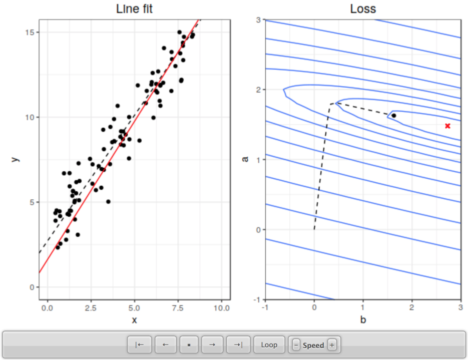
Gradient descent

We can use gradient descent to start from an initial solution and take iterative steps towards the best fit



Gradient descent

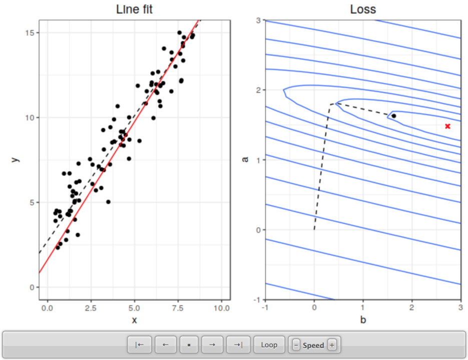
We can use gradient descent to start from an initial solution and take iterative steps towards the best fit



$$\beta \leftarrow \beta - \eta \frac{\partial \mathcal{L}}{\partial \beta}$$

Gradient descent

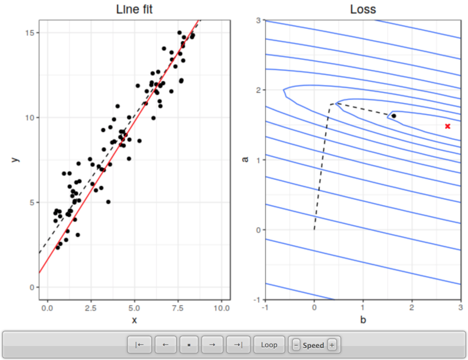
We can use gradient descent to start from an initial solution and take iterative steps towards the best fit



$$\beta \leftarrow \beta - \eta X^T (y - X\beta)$$

Gradient descent

We can use gradient descent to start from an initial solution and take iterative steps towards the best fit



$$\beta \leftarrow (1 - \eta\lambda)\beta - \eta X^T(y - X\beta)$$

Computational cost

Method	Space	Time	Comments
Invert normal equations	$NK + K^2$	K^3	Good for medium-sized datasets with a relatively small number (e.g., hundreds or thousands) of features
Gradient descent	NK	NK per step	Good for larger datasets that still fit in memory but have more (e.g., millions) features; requires tuning learning rate
Stochastic gradient descent	K	K per step	Good for datasets that exceed available memory; more sensitive to learning rate schedule

Conclusions

Bigger models $\neq$ Better models

Conclusions

Our models should be complex enough to explain the past, but simple enough to generalize to the future

Conclusions

Unbiased models $\neq$ Better models

Performance of Different Algorithms in Predicting House Values

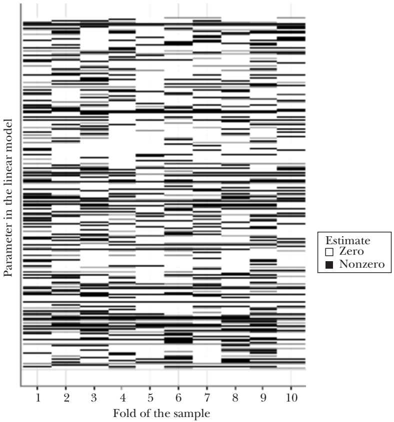
Method	*Prediction performance (R^2)* *Training sample*	*Hold-out sample*	*Relative improvement over ordinary least squares by quintile of house value* 1st	2nd	3rd	4th	5th
Ordinary least squares	47.3%	41.7% [39.7%, 43.7%]	–	–	–	–	–
Regression tree tuned by depth	39.6%	34.5% [32.6%, 36.5%]	−11.5%	10.8%	6.4%	−14.6%	−31.8%
LASSO	46.0%	43.3% [41.5%, 45.2%]	1.3%	11.9%	13.1%	10.1%	−1.9%
Random forest	85.1%	45.5% [43.6%, 47.5%]	3.5%	23.6%	27.0%	17.8%	−0.5%
Ensemble	80.4%	45.9% [44.0%, 47.9%]	4.5%	16.0%	17.9%	14.2%	7.6%

Mullainathan & Spiess, JEP 2017

Conclusions

But these models can be more difficult to interpret

Selected Coefficients (Nonzero Estimates) across Ten LASSO Regressions



Mullainathan & Spiess, JEP 2017

Conclusions

Simple methods (e.g., linear models) work surprisingly well, especially with lots of data

Conclusions

The trick is scaling them up to handle many features and observations