Regression

Machine Learning: Jordan Boyd-Graber
University of Maryland
SLIDES ADAPTED FROM LAUREN HANNAH
Data are the set of inputs and outputs, $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$
In *linear regression*, the goal is to predict $y$ from $x$ using a linear function.
Linear Regression

Examples of linear regression:

- given a child’s age and gender, what is his/her height?
- given unemployment, inflation, number of wars, and economic growth, what will the president’s approval rating be?
- given a browsing history, how long will a user stay on a page?
**Linear Regression**

The equation for linear regression is:

\[ f(x) = \beta_0 + \beta_1 x \]

The data points \((x_i, y_i)\) are plotted on the graph, with the linear model line drawn as a red line through the data points.
**Multiple Covariates**

Often, we have a vector of inputs where each represents a different *feature* of the data

\[ \mathbf{x} = (x_1, \ldots, x_p) \]

The function fitted to the response is a linear combination of the covariates

\[ f(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} \beta_j x_j \]
Multiple Covariates

- Often, it is convenient to represent $\mathbf{x}$ as $(1, x_1, \ldots, x_p)$
- In this case $\mathbf{x}$ is a vector, and so is $\boldsymbol{\beta}$ (we’ll represent them in bold face)
- This is the dot product between these two vectors
- This then becomes a sum (this should be familiar!)

$$\boldsymbol{\beta} \cdot \mathbf{x} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j$$
Hyperplanes: Linear Functions in Multiple Dimensions

Hyperplane
Covariates

- Do not need to be raw value of $x_1, x_2, \ldots$
- Can be any feature or function of the data:
  - Transformations like $x_2 = \log(x_1)$ or $x_2 = \cos(x_1)$
  - Basis expansions like $x_2 = x_1^2$, $x_3 = x_1^3$, $x_4 = x_1^4$, etc
  - Indicators of events like $x_2 = 1\{\text{if } -1 \leq x_1 \leq 1\}$
  - Interactions between variables like $x_3 = x_1 x_2$
- Because of its simplicity and flexibility, it is one of the most widely implemented regression techniques
Fitting a Linear Regression

Idea: minimize the Euclidean distance between data and fitted line

\[ RSS(\beta) = \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta x_i)^2 \]
How to Find $\beta$

- Use calculus to find the value of $\beta$ that minimizes the RSS
- The optimal value is

$$\hat{\beta} = \frac{\sum_{i=1}^{n} y_i x_i}{\sum_{i=1}^{n} x_i^2}$$
Prediction

- After finding $\hat{\beta}$, we would like to predict an output value for a new set of covariates
- We just find the point on the line that corresponds to the new input:

$$\hat{y} = \beta_0 + \beta_1 x$$

(1)
Prediction

- After finding $\hat{\beta}$, we would like to predict an output value for a new set of covariates.
- We just find the point on the line that corresponds to the new input:

$$\hat{y} = \beta_0 + \beta_1 x$$ (1)
Prediction

- After finding \( \hat{\beta} \), we would like to predict an output value for a new set of covariates.
- We just find the point on the line that corresponds to the new input:

\[
\hat{y} = 1.0 + 0.5x
\]  

(1)
Prediction

- After finding $\hat{\beta}$, we would like to predict an output value for a new set of covariates.
- We just find the point on the line that corresponds to the new input:

$$\hat{y} = 1.0 + 0.5 \times 5$$ (1)
Prediction

- After finding $\hat{\beta}$, we would like to predict an output value for a new set of covariates.
- We just find the point on the line that corresponds to the new input:

$$\hat{y} = 3.5$$  \hspace{1cm} (1)

$$y = 1.0 + 0.5x$$

$x = 5.0$

$y = 1.0 + 0.5x$
Probabilistic Interpretation

- Our analysis so far has not included any probabilities
- Linear regression does have a \textit{probabilistic} (probability model-based) interpretation
Probabilistic Interpretation

- Linear regression assumes that response values have a Gaussian distribution around the linear mean function,

\[ Y_i | x_i, \beta \sim N(x_i \beta, \sigma^2) \]

- This is a *discriminative model*, where inputs \( x \) are not modeled

- Minimizing RSS is equivalent to maximizing conditional likelihood
Example: Old Faithful
Example: Old Faithful

We will predict the time that we will have to wait to see the next eruption given the duration of the current eruption.
Example: Old Faithful

We can plot our data and make a function for new predictions

```r
> # Plot a line on the data
> abline(fit.lm, col="red", lwd=3)
>
> # Make a function for prediction
> faithful.fit <- function(x) fit.lm$coefficients[1] + fit.lm$coefficients[2]*x
> x.pred <- c(2.0, 2.7, 3.8, 4.9)
> faithful.fit(x.pred)
[1] 54.93368 62.44443 74.24703 86.04964
```
Example: Old Faithful
Multivariate Linear Regression

Example: \( p = 1 \), have 2 points

- Have \( p + 1 \) or fewer points, line hits all (or \( p \) with mean 0 data)
- \( \geq p + 1 \) (but still close to that number), line goes *close* to all points
Noise, Bias, Variance Tradeoff

- **Noise**: Lower bound on performance
- **Bias**: Error as a result as choosing the wrong model
- **Variance**: Variation due to training sample and randomization
Noise, Bias, Variance Tradeoff

- **Noise**: Lower bound on performance
- **Bias**: Error as a result as choosing the wrong model
- **Variance**: Variation due to training sample and randomization

- No model is perfect
- More complex models are more susceptible to errors due to variance
Multivariate Linear Regression

Why linear regression:

- has few parameters to estimate \((p)\)
- really restrictive model—low variance, higher bias

- should be good for data with few observations, large number of covariates...
- ... but we can’t use it in this situation
**Multivariate Linear Regression**

Idea: if we have a large number of covariates compared to observations, say $n < 2p$, **best to estimate most coefficients as 0!**

- not enough info to determine all coefficients
- try to estimate ones with strong signal
- set everything else to 0 (or close)

Coefficients of 0 may not be a bad assumption...

*If we have 1,000s of coefficients, are they all equally important?*
Gene Expression

Example: microarray gene expression data

- gene expression: want to measure the level at which information in a gene is used in the synthesis of a functional gene product (usually protein)
- can use gene expression data to determine subtype of cancer (e.g. which type of Lymphoma B?) or predict recurrence, survival time, etc
- problem: thousands of genes, hundreds of patients, $p > n$!

Intuition: only a handful of genes should affect outcomes
- gene expression levels are continuous values
- data: observation $i$ is gene expression levels from patient $i$, attached to outcome for patient (survival time)
- covariates: expression levels for $p$ genes
- collinearity: does it matter *which* gene is selected for *prediction*? No!
- overfitting: now fitting $p'$ non-0 coefficients to $n$ observations with $p' \ll n$ means less fitting of noise
Regularized Linear Regression

Regularization:
- still minimize the RSS
- place a *penalty* on large values for $\beta_1, \ldots, \beta_p$ (why not $\beta_0$? can always easily estimate mean)
- add this penalty to the objective function
- solve for $\hat{\beta}$!

New objective function:

$$
\hat{\beta} = \arg\min_{\beta} \frac{1}{2} \sum_{i=1}^{n} (y_i - x_i \beta)^2 + \lambda \sum_{j=1}^{p} \text{penalty}(\beta_j)
$$

$\lambda$ acts as a weight on penalty: low values mean few coefficients near 0, high values mean many coefficients near 0
Regularized Linear Regression

Regularization:

- still minimize the RSS
- place a \textit{penalty} on large values for $\beta_1, \ldots, \beta_p$ (why not $\beta_0$? can always easily estimate mean)
- add this penalty to the objective function
- solve for $\hat{\beta}$!

New objective function:

$$
\hat{\beta} = \arg\min_{\beta} \frac{1}{2} \sum_{i=1}^{n} (y_i - x_i \beta)^2 + \lambda \sum_{j=1}^{p} \text{penalty}(\beta_j)
$$

$\lambda$ acts as a weight on penalty: low values mean few coefficients near 0, high values mean many coefficients near 0.
Regularized Linear Regression

Regularization:
- still minimize the RSS
- place a *penalty* on large values for $\beta_1, \ldots, \beta_p$ (why not $\beta_0$? can always easily estimate mean)
- add this penalty to the objective function
- solve for $\hat{\beta}$!

New objective function:

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{2} \sum_{i=1}^{n} (y_i - x_i \beta)^2 + \lambda \sum_{j=1}^{p} \text{penalty}(\beta_j)$$

$\lambda$ acts as a weight on penalty: low values mean few coefficients near 0, high values mean many coefficients near 0
Regularized Linear Regression

Regularization:
- still minimize the RSS
- place a *penalty* on large values for $\beta_1, ..., \beta_p$ (why not $\beta_0$? can always easily estimate mean)
- add this penalty to the objective function
- solve for $\hat{\beta}$!

New objective function:

$$
\hat{\beta} = \arg\min_{\beta} \frac{1}{2} \sum_{i=1}^{n} (y_i - x_i \beta)^2 + \lambda \sum_{j=1}^{p} \text{penalty}(\beta_j)
$$

$\lambda$ acts as a weight on penalty: low values mean few coefficients near 0, high values mean many coefficients near 0.
Regularized Linear Regression

**Regularization**: what is a good penalty function?

Same as penalties used to fit errors:

- **Ridge regression** (squared penalty):

  \[
  \hat{\beta}^{Ridge} = \arg\min_{\beta} \frac{1}{2} \sum_{i=1}^{n} (y_i - x_i \beta)^2 + \lambda \sum_{j=1}^{p} \beta_j^2
  \]

- **Lasso regression** (absolute value penalty):

  \[
  \hat{\beta}^{Lasso} = \arg\min_{\beta} \frac{1}{2} \sum_{i=1}^{n} (y_i - x_i \beta)^2 + \lambda \sum_{j=1}^{p} |\beta_j|
  \]
## Comparing Ridge and Lasso

<table>
<thead>
<tr>
<th>Objective</th>
<th>Ridge</th>
<th>Lasso</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimator</td>
<td>$\frac{1}{2} \sum_{i=1}^{n} (y_i - x_i \beta)^2 + \lambda \sum_{j=0}^{p} \beta_j^2 \left( X^T X + \lambda I \right)^{-1} X^T y$</td>
<td>$\frac{1}{2} \sum_{i=1}^{n} (y_i - x_i \beta)^2 + \lambda \sum_{j=0}^{p}</td>
</tr>
<tr>
<td>Coefficients</td>
<td>most close to 0</td>
<td>most exactly 0</td>
</tr>
<tr>
<td>Stability</td>
<td>robust to changes in $X, y$</td>
<td>not robust to changes in $X, y$</td>
</tr>
</tbody>
</table>

Regularized linear regression is fantastic for low signal datasets or those with $p >> n$

- **Ridge**: good when many coefficients affect value but not large (gene expression)
- **Lasso**: good when you want an *interpretable* estimator
Choosing $\lambda$

Both Ridge and Lasso have a tunable parameter, $\lambda$

- use cross validation to find best $\lambda$

$$\hat{\lambda} = \arg \min_{\lambda} \sum_{i=1}^{n} (y_i - x_i \hat{\beta}_{-i, \lambda})^2$$

- try out many values
- see how well it works on “development” data
Regression

- Workhorse technique of data analysis
- Fundamental tool that we saw before (“Logistic Regression”)
- Important to understand interpretation of regression parameters