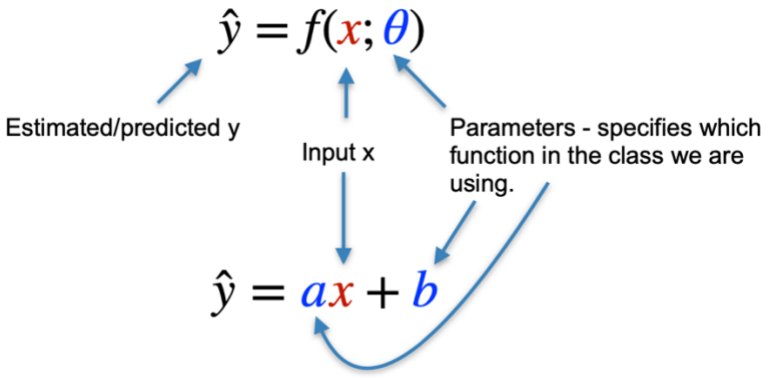
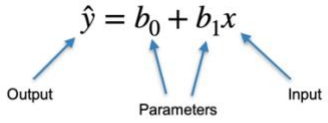
W1

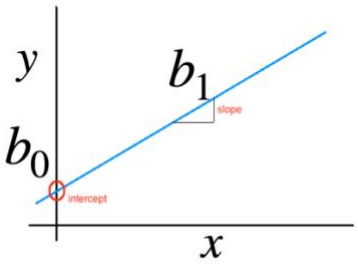
Model

- a function that relates some inputs (x) to some outputs (y)
- Most models have parameters ("θ), which allows them to represent whole classes of functions



Linear regression





Fitting a Model

1. Define the model: Choose the "class" of functions that relates the inputs (x) to the output (y)
2. Define your training loss
3. Find the function in your class/form that gives the smallest training loss

Loss functions

- A training loss function measures the deviation of the model fits from the observed data
- A large loss indicates a poor fit to the training data
- Different loss functions penalize different deviations differently
- We find the parameters that minimize our loss function given the training data

Residuals

- The difference between the observed value of the dependent variable (y) and the predicted value (ŷ) is called the residual (e)
- the errors from the model fit: Data = Fit + Residual

a criterion for the best line

- L1: Minimize the sum of magnitudes (absolute values) of residuals: The L1-norm
  - $$L(\theta) = \sum_{i=1}^{n} |y_i - \hat{y}_i| = \sum_{i=1}^{n} |r_i| = \|\mathbf{r}\|_1$$ LAD: Least Absolute Deviation
- L2: minimize the sum of squared residuals: the squared L2-norm
  - $$L(\theta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} r_i^2 = \|\mathbf{r}\|_2^2$$ OLS: Ordinary Least Squares
- The most commonly used is least squares
  - Motivated by normal distribution of errors
  - Solutions can be easily computed
  - Big errors count relatively more than small errors

Optimization

- Finding the best function is the same problem as finding the best parameters.
- Parameter estimation is the process of minimizing the training loss by trying different values of the parameters
- The setting of parameters that gives you the smallest loss is the best estimate of the parameters, and the best function in your class

Using the derivative of the loss

- By providing the derivative of the loss function in respect to the parameters, optimization can be sped up.

Fit: $\hat{y}_i = b_0 + b_1 x_i$

Residual: $r_i = y_i - \hat{y}_i$

$$\frac{\partial \sum f_i(\theta)}{\partial \theta} = \sum \frac{\partial f_i(\theta)}{\partial \theta}$$

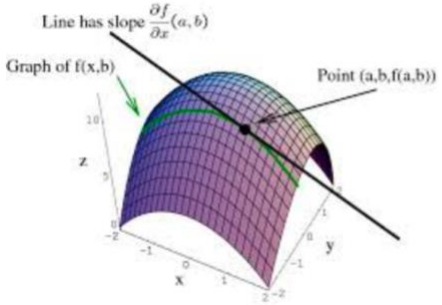
Loss: $L = \sum_{i=1}^{N} (y_i - b_0 - b_1 x_i)^2$

$$\frac{\partial f(g(\theta))}{\partial \theta} = \frac{\partial f(g)}{\partial g} \frac{\partial g(\theta)}{\partial \theta}$$

Derivative b_0: $\frac{\partial L}{\partial b_0} = -2 \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i) = -2 \sum_{i=1}^{n} r_i$

Derivative b_1: $\frac{\partial L}{\partial b_1} = -2 \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i) x_i = -2 \sum_{i=1}^{n} r_i x_i$

- The vector of partial derivatives is called a gradient or Jacobian.



$$\nabla_\theta L = \begin{bmatrix} \frac{\partial L}{\partial \theta_0} \\ \frac{\partial L}{\partial \theta_1} \end{bmatrix}$$

the least absolute deviation loss function for a linear model

Prediction: $\hat{y}_i = b_0 + b_1 x_i$

Residual: $r_i = y_i - \hat{y}_i$

Loss: $\mathrm{L} = \sum_{i=1}^{n} |y_i - (b_0 + b_1 x_i)|$

Derivative b_0: $\frac{\partial \mathrm{L}}{\partial b_0} = -\sum_{i=1}^{n} \mathrm{sgn}(y_i - (b_0 + b_1 x_i)) = -\sum_{i=1}^{n} \mathrm{sgn}(r_i)$

Derivative b_1: $\frac{\partial \mathrm{L}}{\partial b_0} = -\sum_{i=1}^{n} \mathrm{sgn}(y_i - (b_0 + b_1 x_i)) \cdot x_i = -\sum_{i=1}^{n} \mathrm{sgn}(r_i) \cdot x_i$

Evaluating model fit

- The quality of the fit of a linear regression model is most commonly evaluated using R2 the coefficient of determination.
- R2 is calculated from the ratio of residual sum of squares – total sum of squares.
  - $R^2 = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}} = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$ Residual sum of squares (RSS); Total sum of squares (TSS); mean
  - It tells us what percent of variability in the response variable is explained by the model. (0=no fit, 1=perfect fit)
  - The remainder of the variability is explained by variables not included in the model or by inherent randomness in the data.
  - Because OLS is miming the RSS, it will always have the highest R2 value possible for that class of models.

L1-loss and median regression Robust techniques

Impact of the loss function

- non-robust: Answers that change a lot with exclusion of a few data points

Robust statistics

- Robust statistics seek to provide methods that emulate popular statistical methods, but which are not unduly affected by outliers or other small departures from model assumptions.
- Mean is a measure of central tendency that is sensitive to outliers
- Median is a measure of central tendency that is robust against outliers

Robust regression

- find the point that minimize the sum of absolute errors
  - $L = \sum_{i=1}^{n} |y_i - b_0|$

    $L = \sum_{i=1}^{n} \begin{cases} (y_i - b_0) \text{ if } y_i > b_0 \\ -(y_i - b_0) \text{ if } y_i \le b_0 \end{cases}$

    $\frac{\partial L}{\partial b_0} = \sum_{i=1}^{n} \begin{cases} -1 \text{ if } y_i > b_0 \\ 1 \text{ if } y_i \le b_0 \end{cases}$

    The minimum is reached at the **median**

- Median regression is a robust regression technique

W2 Statistics, prediction, and Maximum likelihood

Summary

- Probabilities and Events
- Random Variables
  - Discrete, Continuous
- Distributions
- PMF, PDF, CDF
- Joint, Marginal, Conditional
- Expectations
- Likelihoods
- Maximum Likelihood Regression

Probabilities and Events

Sample Spaces and Events

- Sample space $\mathcal{S}$ is the set of all possible events we might observe. Depends on context.
  - Coin flips: $\mathcal{S}$ = {h, t} $\geqslant 0$
  - Person's height in cm: $\mathcal{S}$ = R>=0
- An event is a subset of the sample space.
- Observe heads: {h}
  - Observe height of at least 183cm: $[183.0, \infty)$
  - Observe a person whose height is between 170cm and 190cm and weight is between 65kg and 72kg: [170,190] × [65,72].

Event Probabilities

- Any event can be assigned a probability between 0 and 1 (inclusive).
  - Pr({h}) = 0.5
  - Pr([170,190] × [65,72]) = 0.10
- Probability of the observation falling somewhere in the sample space is 1.
  - Pr($\mathcal{S}$) = 1

Interpreting probability: Objectivist view

- Suppose we observe n replications of an experiment.
- Let n(A) be the number of times event A was observed within the replications
- $\frac{n(A)}{n}$ converges to $\Pr(A)$ as $n \to \infty$
- This is (loosely) Borel's Law of Large Numbers
- Subjective interpretation is possible as well. ("Bayesian" statistics is related to this idea.)

Data Generating Processes and Random Variables

Where did the data come from?

- Each item (xi, yi) is an observation
- Probabilistic approach:
  - Describe the distribution of the (xi, yi) using the formalisms of probability, e.g. with generative models
  - Use the models to reveal structure and relationships
  - Use the models to make predictions

Replicates

- Common assumption is that data consists of replicates that are "the same."
- Come from "the same population"
- Come from "the same process"
- The goal of data analysis is (usually) to understand what the data tell us about the population they come from so we can generalize what we learn.

Abstraction of data-generating process: Random Variable

- We often reduce data to numbers. • "1 means heads, 0 means tails."
- A *random variable* is a mapping from the event space to a number (or vector.)
- Usually rendered in uppercase *italics:* X,Y,Z.
- "Realizations" written in lower case: x_1, x_2, ...
- We will write the set of possible realizations as: $\mathcal{X}$ for X, $\mathcal{Y}$ for Y, and so on.

Distributions of random variables

- Realizations are observed according to probabilities specified by the distribution of X
- Can think of X as an "infinite supply of data"
- Separate realizations of the same r.v. X are independent and identically distributed (i.i.d.)
- Formal definition of a random variable requires measure theory, not covered here

Probabilities for random variables

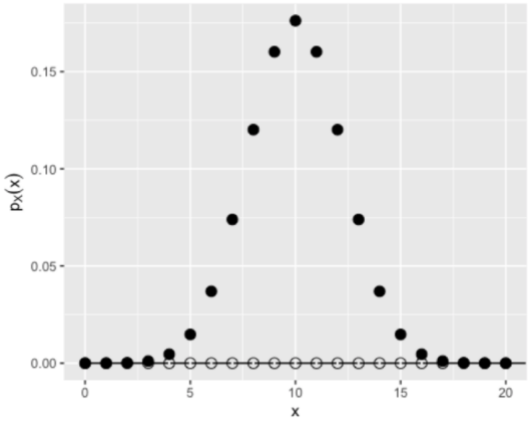
- Random variable X, realization x.
- What is the probability we see x?
    - Pr(X = x), (if lazy, Pr(x), but don't do this)
- Subsets of the domain of a random variable correspond to events.
    - Pr(X > 0) probability that I see a realization that is positive.

Discrete Random Variables

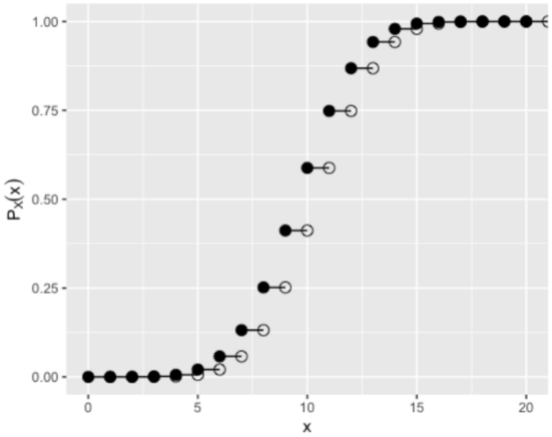
- Discrete random variables take values from a countable set
    - Coin flip X •$\mathcal{X} = \{0,1\}$
    - Number of snowflakes that fall in a day Y •$\mathcal{Y} = \{0,1,2,...\}$

Probability Mass Function (PMF)

- For a discrete X, $p_X(x)$ gives $\Pr(X = x)$.
- Requirement: $\sum_{x \in \mathcal{X}} p_X(x) = 1.$
    - Note that the sum can have an infinite number of terms.
- Example
    - X is number of "heads" in 20 flips of a fair coin
        - $\mathcal{X}$ = {0,1,...,20}



Cumulative Distribution Function (CDF)

- For a discrete X, PX(x) gives Pr(X $\leqslant$ x).
- Requirements:
  - P is nondecreasing
  - $\sup_{x \in \mathcal{X}} P_X(x) = 1$
- Note
  - $$P_X(b) = \sum_{x \leq b} p_X(x)$$
  - $$\Pr(a < X \leq b) = P_X(b) - P_X(a)$$
- Same example
  - 

Continuous Random Variables

- Continuous random variables take values in intervals of R
- Mass M of a star → M = (0, ∞)
- Oxygen saturation S of blood → $\mathcal{S}$ = [0, 1]
- For a continuous r.v. X, Pr(X = x) = 0 for all x. → There is no probability mass function.
  - Probability is 0 but it doesn't mean it won't happen
- However, Pr(X ∈ (a, b)) ≠ 0 in general.

Probability Density Function (PDF)

- For continuous X, Pr(X = x) = 0 and PMF does not exist.

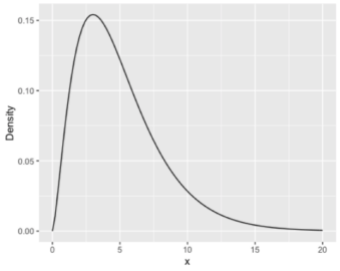
- However, we define the Probability Density Function f :

$$\Pr(a \le X \le b) = \int_a^b f_X(x)\,\mathrm{d}x$$

- Requirement:

$$\forall x\, f_X(x) > 0, \int_{-\infty}^{\infty} f_X(x)\,\mathrm{d}x = 1$$

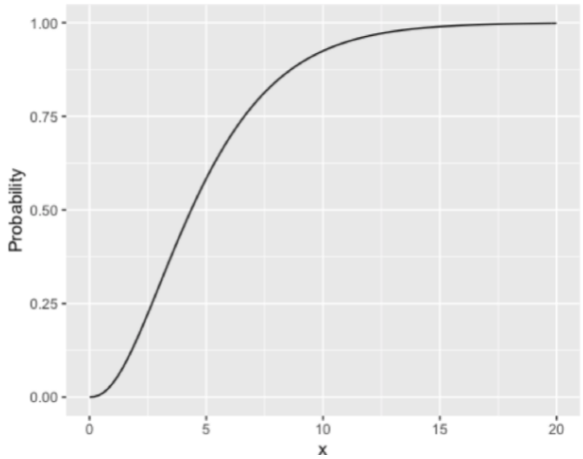
- Ex



Cumulative Distribution Function (CDF)

- For a continuous X, $F_X(x)$ gives $\Pr(X \le x) = \Pr(X \in (-\infty, x])$.
- Requirements:
  - F is nondecreasing
  - $\sup_{x \in \mathcal{X}} F_X(x) = 1$
- Note:
  - $F_X(x) = \int_{-\infty}^{x} f_X(x)\,\mathrm{d}x$
  - $\Pr(x_1 < X \le x_2) = F_X(x_2) - F_X(x_1)$

- Ex

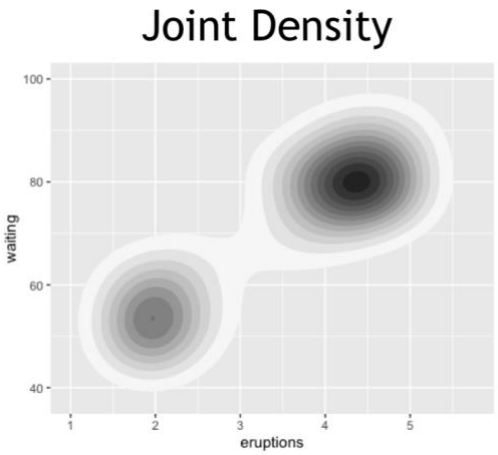


Random Vectors and Joint Distributions

Joint Distributions

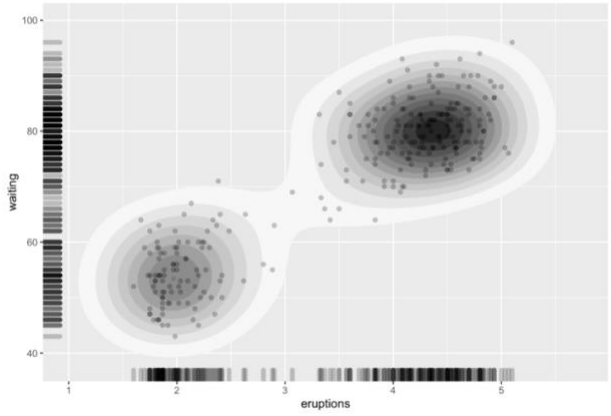
- Two random variables X and Y have a joint distribution if their realizations come together as a pair. (X, Y) is a random vector, and realizations may be written (x1, y1), (x2, y2)
- Joint Cumulative Distribution Function (CDF)
    - We define the Joint Cumulative Distribution Function FX,Y:
        - $\Pr(X \leq b, Y \leq d) = F_{X,Y}(b, d)$ x <=b and y <= d
- Joint Probability Density Function (PDF)
    - We define the *Joint Probability Density Function* $f_{X,Y}$:
    - $\Pr[(X, Y) \in \mathcal{A} \subseteq \mathcal{X} \times \mathcal{Y}] = \int_{\mathcal{A}} f_{X,Y}(x, y)\mathrm{d}x\mathrm{d}y$
    -
- Ex

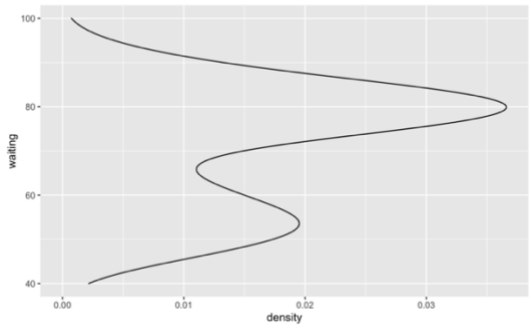
	eruptions	waiting
1	3.600	79
2	1.800	54
3	3.333	74
4	2.283	62
5	4.533	85
6	2.883	55
7	4.700	88
8	3.600	85
9	1.950	51
10	4.350	85



Marginal Distributions

- (X, Y) is a random vector
- If we ignore X, the distribution of Y is its marginal distribution.
    - Marginal density of Y

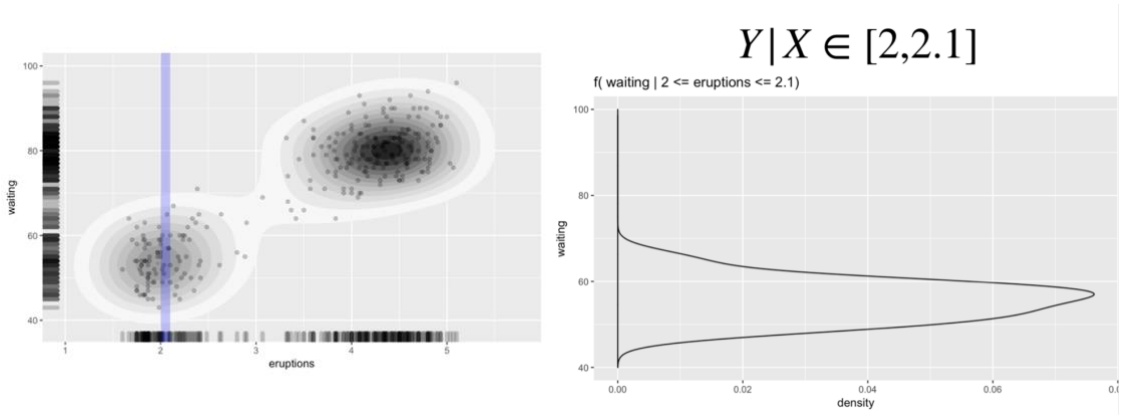


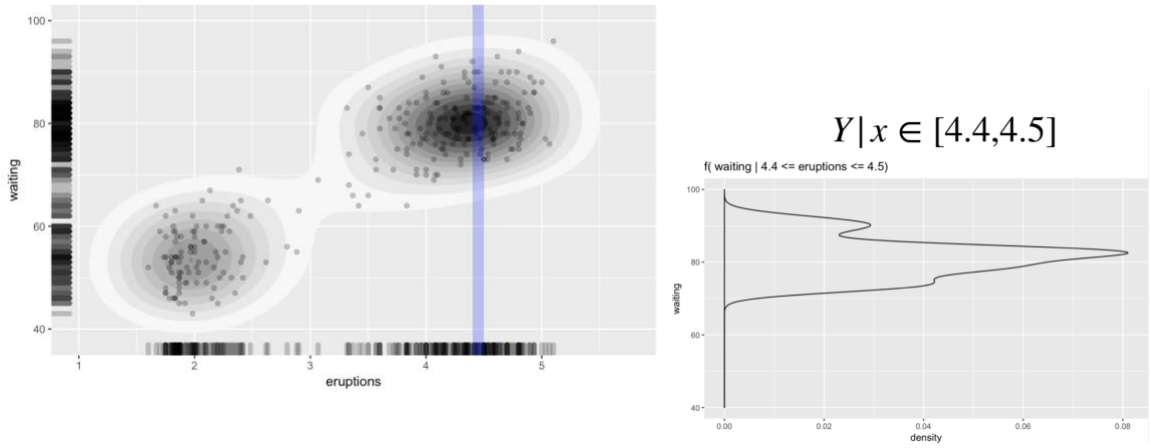


Conditional distributions

- (X, Y) is a random vector
- Suppose we only look at Y values for which X = 5.

- This is a new random variable, which we write as Y|X = 5
- The distribution describing this random variable is the conditional distribution of Y given X = 5.

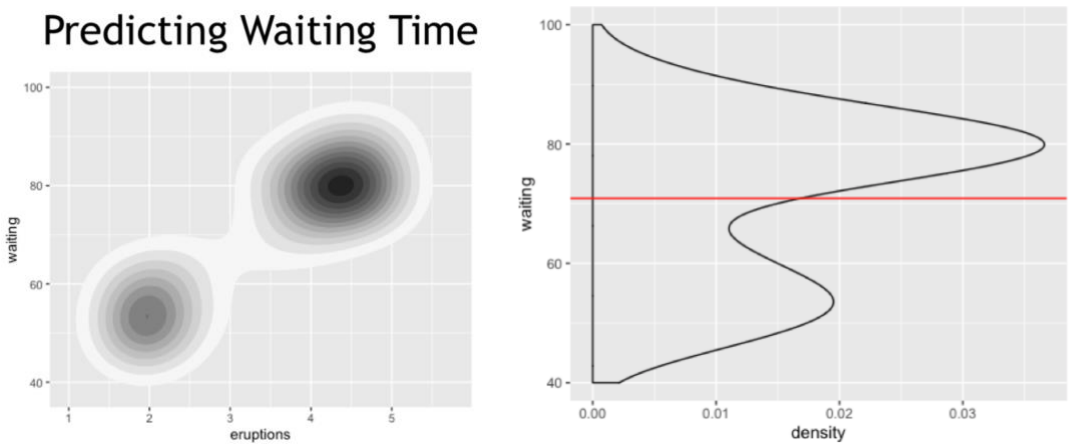




Predictions from Conditional Distributions

Expectations

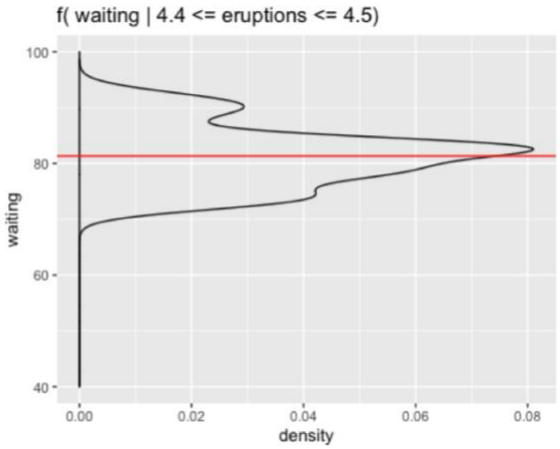
- The expected value of a discrete random variable X is denoted
  - Is influenced by which of these values have a really high probability
  - Location of the distribution
  - $E[X] = \sum_{x \in \mathcal{X}} x \cdot p_X(X = x)$
- The expected value of a continuous random variable Y is denoted
  - Integrate value * density
  - $E[Y] = \int_{y \in \mathcal{Y}} y \cdot f_Y(Y = y)\, dy$
- E[X] is called the mean of X, often denoted μ or μX.
- Measure of location of the distribution.



Sample Mean: 70.90

Conditional predictions

```
## Mean: 81.33
```



Regression

- Regression methods learn a model, e.g. $\hat{y} = b0 + b1x$
- $\hat{y}$ is often an estimate of $E[Y|X = x]$

Parametric Distributions and Likelihood

- estimate distribution

Probabilistic Model Estimation

- Find a good, compact description of the distribution of the data
- Use that to
    - Infer structure in the data
    - Make predictions, possibly using some partial information

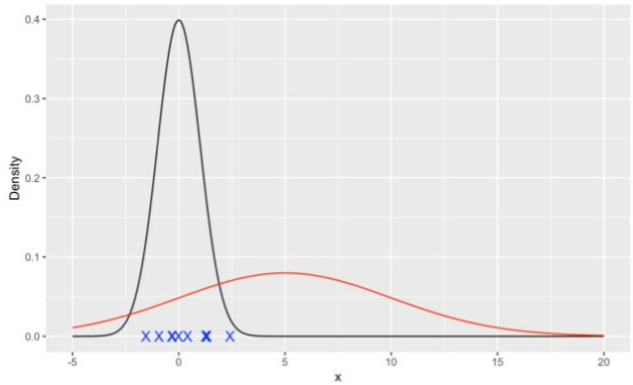
Normal (Gaussian) Distributions

- Density is a function possible realization of random variable (y)

$$f_Y(y) = \frac{1}{\sqrt{2\pi\sigma_Y^2}} e^{-\frac{(y-\mu_Y)^2}{2\sigma_Y^2}}$$

ameters: μ_Y, σ_Y^2.

Mean (location) Variance (scale)



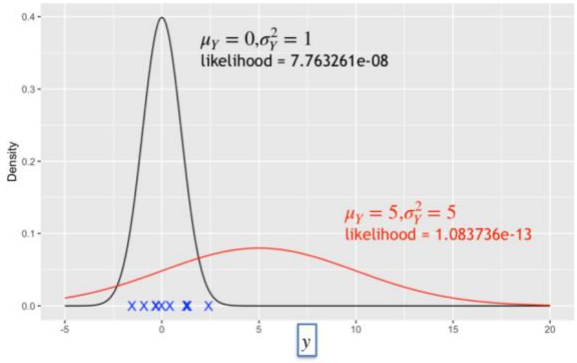
Likelihood

- "Given a distribution with parameters θ, how likely are my data?"
- If data items are independent and identically distributed (i.i.d.) then the probability of the seeing all the realizations is the product of the probability of each realization

$$\mathscr{L}(\theta; y_1, y_2, \ldots, y_n) = \prod_i p_Y(\theta; y_i) \text{ (if discrete)}$$

$$\mathscr{L}(\theta; y_1, y_2, \ldots, y_n) = \prod_i f_Y(\theta; y_i) \text{ (if continuous)}$$

  -
-



-

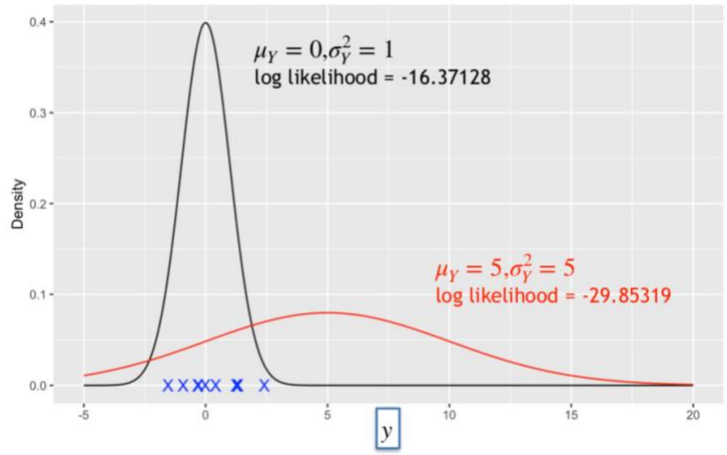
Log likelihood

- Practically, Data Scientists mostly deal with log likelihoods
  - Maximizing the likelihood = maximizing the log likelihood
    - a>b ⇔ log(a)>log(b)
- Logs turn products of probabilities into sums of log-probabilities log(a × b) = log(a) +log(b)

    - Products of many small numbers get small very fast -> Problems with numerical accuracy. Logs get negative, but grow much more slowly
- For exponential-family distributions, the log likelihood has a simpler functional form.

$$\ell(\theta; y_1, y_2, \ldots, y_n) = \sum_i \log(p_Y(\theta; y_i)) \text{ (if discrete)}$$

$$\ell(\theta; y_1, y_2, \ldots, y_n) = \sum_i \log(f_Y(\theta; y_i)) \text{ (if continuous)}$$



The Maximum Likelihood Principle

Maximum Likelihood Principle

- Identify a set of possible distributions that could describe the data. ("Statistical model.") E.g. set of all normal distributions.
- Find the specific distribution in the set that makes the data appear most likely
- Infer and predict using that model

Normal Log Likelihood

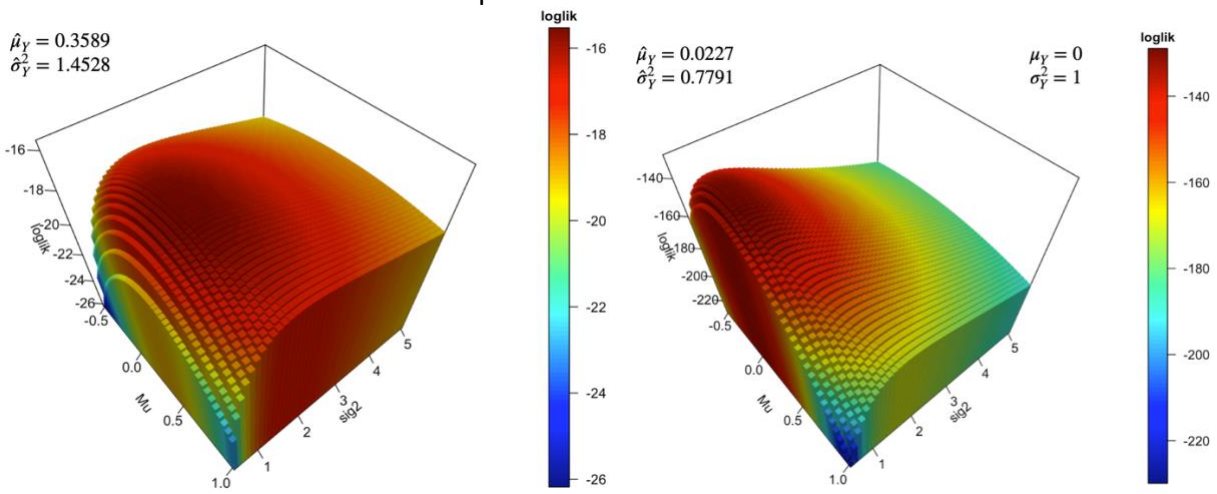
- $$f_Y(y) = \frac{1}{\sqrt{2\pi\sigma_Y^2}} e^{-\frac{(y-\mu_Y)^2}{2\sigma_Y^2}} \quad \log(f_Y(y)) = \log\left(\frac{1}{\sqrt{2\pi\sigma_Y^2}} e^{-\frac{(y-\mu_Y)^2}{2\sigma_Y^2}}\right)$$
- $$\log(f_Y(y)) = -\frac{1}{2}\log(2\pi) - \frac{1}{2}\log\left(\sigma_Y^2\right) - \frac{1}{2}\frac{(y-\mu_Y)^2}{\sigma_Y^2}$$
- $$\ell(\mu_Y, \sigma_Y^2; y_1, \ldots, y_n) = \sum_{i=1}^{n}\left[-\frac{1}{2}\log(2\pi) - \frac{1}{2}\log\left(\sigma_Y^2\right) - \frac{1}{2}\frac{(y_i-\mu_Y)^2}{\sigma_Y^2}\right]$$
- $$\ell(\mu_Y, \sigma_Y^2; y_1, \ldots, y_n) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log\left(\sigma_Y^2\right) - \frac{1}{2}\frac{\sum_{i=1}^{n}(y_i-\mu_Y)^2}{\sigma_Y^2}$$

Maximum Likelihood Estimation

- Minimize square errors → maximize likelihood
- What μY gives the highest likelihood?
  - $$\frac{\partial \ell}{\partial \mu_Y} = \frac{1}{\sigma_Y^2} \sum_{i=1}^{n} (y_i - \mu_Y)$$
  - $$\frac{\partial \ell}{\partial \mu_Y} = 0 \iff \mu_Y = \frac{\sum_{i=1}^{n} y_i}{n}$$
- What σ_Y^2 gives the highest likelihood?
  - $$\frac{\partial \ell}{\partial \sigma_Y^2} = -\frac{n}{2\sigma_Y^2} + \frac{\sum_{i=1}^{n} (y_i - \mu_Y)^2}{2\sigma_Y^4}$$
  - $$\frac{\partial \ell}{\partial \sigma_Y^2} = 0 \iff \sigma_Y^2 = \frac{\sum_{i=1}^{n} (y_i - \mu_Y)^2}{n}$$

Log likelihood surface

- more data → more and more peeked around the correct value



Maximum Likelihood Regression

Why least square?

- It corresponds to maximum likelihood model

Maximum Likelihood Regression

- Choose the form of the function for
  - $$\mu_{Y|X=x} = E[Y|X = x]$$
- Choose the distribution of the error
  - $$Y|X = x \quad - \quad \mu_{Y|X=x}$$
- Use these to determine the likelihood of the data
- Use an optimizer to find the parameters that maximize the likelihood

- See also: “Generalized Linear Models” (GLMs)

The Power of the Maximum Likelihood Approach

- Maximum likelihood gives a way of adapting the model- fitting approach to many different types of observations y
- If model assumptions are correct, the maximum likelihood parameter estimates have good theoretical properties.
- Even if assumptions aren’t exactly correct, it is often pretty good.
- Maximum likelihood estimates are used as a “foundation” that can be adjusted, e.g. by regularization (later lesson)

W3

Classification

- Supervised learning
- $\mathcal{Y}$ is discrete and finite.

Classification vs. Regression

Same

- Both want to predict y given x
- Learn from “labeled data” (x, y) pairs
- Can be achieved by optimizing a loss function/likelihood

Different

- Space of labels is often not numeric but categorical - have to decide how to encode as a number
- Appropriate loss functions are different
- Measuring performance can be more complicated (loss of “false positive” and “false negative” may be different)

Logistic Regression

- Classification algorithm

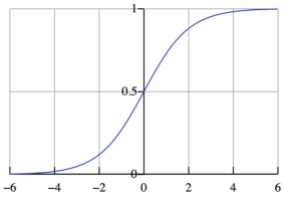
Probabilistic Classifier

- In classification, sometimes knowing x of a realization (x, y) enables perfect prediction of y; sometimes it doesn’t.
- Probabilistic classifiers assume imperfection, estimate Pr(Y|X = x)
    - Assume the probability of y given x
- Given new x, classifier estimates the probabilities of each possible label.

Bernoulli Random Variable

- Discrete random variable
- Y ∈ 0,1
- Parameter p is Pr(Y = 1)
- (1−p) is Pr(Y=0)

Sigmoid Function

- 

$$\varsigma(z) = \frac{1}{1 + e^{-z}}$$
Inputs: $(-\infty, \infty)$ Outputs: $(0,1)$

Logistic regression

$$p(x;\mathbf{b}) = \varsigma(b_0 + b_1 x) = \frac{1}{1 + e^{-(b_0+b_1 x)}}$$

- $$p(x;\mathbf{b}) = \Pr(Y = 1 \mid X = x;\mathbf{b})$$
  $$1 - p(x;\mathbf{b}) = \Pr(Y = 0 \mid X = x;\mathbf{b})$$

Logistic Regression Likelihood

Log Likelihood for Training Set

- $$\text{data} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$
- $$\ell(\mathbf{b}; \text{data}) = \sum_i \log(\Pr(Y = y_i \mid X = x_i;\mathbf{b}))$$
- $$\ell(\mathbf{b}; \text{data}) = \sum_i \log \begin{cases} p(x_i;\mathbf{b}) \text{ if } y_i = 1 \\ 1 - p(x_i;\mathbf{b}) \text{ if } y_i = 0 \end{cases}$$
- $$\ell(\mathbf{b}; \text{data}) = \sum_i \log \left[y_i p(x_i;\mathbf{b}) + (1 - y_i)(1 - p(x_i;\mathbf{b}))\right]$$

Logistic Regression summary

- Define: $$\Pr(Y = 1 \mid X = x) = \frac{1}{1 + e^{-(b_0+b_1 x)}}$$
- Choose b to maximize
  - $$\ell(\mathbf{b}; y_1, y_2, \ldots, y_n) = \sum_i \log \left[y_i p(x_i;\mathbf{b}) + (1 - y_i)(1 - p(x_i;\mathbf{b}))\right]$$

Regularization for Logistic Regression

Maximizing Likelihood

- $$\Pr(Y = 1 \mid X = x) = p(x;\mathbf{b}) = \frac{1}{1 + e^{-(b_0+b_1 x)}}$$
- Choose $\mathbf{b}$ to maximize
  $$\ell(\mathbf{b}; y_1, y_2, \ldots, y_n) = \sum_i \log \left[y_i p(x_i;\mathbf{b}) + (1 - y_i)(1 - p(x_i;\mathbf{b}))\right]$$
- Consider a tiny dataset: (x1 = 1,y1 = 1), (x2 = − 1,y2 = 0)
- Log Likelihood:
  - $\log(p(1;\mathbf{b})) + \log(1 - p(-1,\mathbf{b}))$
  - $$\log\left(\frac{1}{1 + e^{-(b_0+b_1)}}\right) + \log\left(1 - \frac{1}{1 + e^{-(b_0-b_1)}}\right)$$
  - $$\log\left(\frac{1}{1 + e^{-(b_0+b_1)}}\right) + \log\left(1 - \frac{1}{1 + e^{-(b_0-b_1)}}\right)$$
- Fix b0 to make things easy. What is the best b1?
- b1= ∞ !(Oraslargeab1asyoucanstand...)
- Likelihood approaches 0 from below as b1 increases.
- No matter what the value of b0 is.
- Means that there is no best b0 because for any choice we can make likelihood as close to 0 as we want by increasing b1 .

Regularization

- Idea: Instead of optimizing likelihood, optimize likelihood plus penalty on the size of the parameters
- The penalty prevents infinite optimal solutions
- $\text{objective}(\mathbf{b}) = -\ell(\mathbf{b}; \text{data}) + \text{penalty}(\mathbf{b})$
- Common penalties:
    - The "L2" or "Ridge" penalty
        - $\sum_j b_j^2 = \|\mathbf{b}\|^2$
    - The "L1" or "LASSO" penalty
        - $\sum_j |b_j| = \|\mathbf{b}\|_1$

In Practice

- Statisticians typically don't regularize when number of parameters is small.
- they don't penalize the intercept.
    - B0
- The sklearn package penalizes by default, including the intercept.

Criteria for Evaluating Classifiers

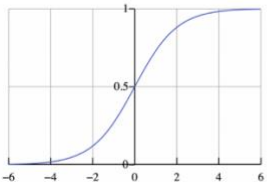
Evaluating Using a Data Set

- Take a classifier
- Apply to a dataset with known labels
- Compare the classifier outputs to the actual labels

Obtaining Labels

- Likelihood or cross-entropy is nice for training but difficult to interpret
- Classifiers are usually evaluated in terms of when they are "right" or "wrong."
- To do so, probabilistic classifiers must be thresholded - e.g.
    - $$\hat{y} = \begin{cases} 1 \text{ if } \Pr(Y = 1 \mid X = x) \geq 0.5 \\ 0 \text{ if } \Pr(Y = 1 \mid X = x) < 0.5 \end{cases}$$

Thresholding Logistic Regression

- When is $\Pr(Y = 1 \mid X = x) \geq 0.5$?
    - Remember:
    - $$\Pr(Y = 1 \mid X = x) = \frac{1}{1 + e^{-(b_0 + b_1 x)}}$$
    - 

    Curve is ≥ 0.5 when $b_0 + b_1 x \geq 0$

Decision Boundaries

- When is $\Pr(Y = 1 \mid X = x) = 0.5$?
- When $b_0 + b_1 x = 0$
- If choosing a threshold of 0.5, this is the decision boundary.
- Other methods, like Support Vector Machines, try to learn a decision boundary directly; they can be preferable when data are separable.

Obtaining a Ranking

$$\text{data} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

- Some classifiers can provide a ranking of a dataset from "most positive" to "most negative"
- Rankings that put "most" of the positives near the beginning are preferable.
- Important criteria use the whole ranking (not just the labels) to assess the classifier.

Using Probabilities

- Some evaluation criteria require the classifier to produce probabilities (like logistic regression).
- Intuition: Penalize misclassifications that are highly confident and wrong.
    - If classifier says $\Pr(Y = 1 \mid X = x) = 0.99$ but $y = 0$, that is worse than it saying $\Pr(Y = 1 \mid X = x) = 0.55$ but $y = 0$
- Cross-entropy does this; sometimes it is called log loss. Popular in Kaggle competitions.
- Note: If your model gives a label probability that is exactly 1, but then is wrong, loss is ∞

Label-based Criteria

- These criteria only need a label from the classifier. (Not a rank or probability.)
- Easy to interpret
- Still can provide information about different kinds of errors

Accuracy / 0-1 Loss

$$\text{Accuracy} = \frac{1}{n}\sum_{i=1}^{n}\begin{cases} 1 \text{ if } y_i = \hat{y}_i \\ 0 \text{ if } y_i \neq \hat{y}_i \end{cases}$$

- 0-1 Loss is (1 − Accuracy);
- this is sometimes called "error rate"
- Intuitive! How often is the model correct, on average
- But unhelpful or misleading when classes are imbalanced

Imbalanced classes

- Suppose in the true population, 95% are negative.
    - Classifier that always outputs negative is 95% accurate. This is the baseline accuracy.
- Accuracy is not very meaningful if we don't know the class distribution.

Threshold Affects Accuracy

- 0.50 threshold: 95% Accuracy
- 0.05 threshold: 70% Accuracy
- lower threshold is more appealing

Label-based Criteria False Positives and False Negatives

Confusion Matrix

50% Threshold Classifier	Truth:+	Truth:-
Pred:+	0	0
Pred:-	50	950

5% Threshold Classifier	Truth:+	Truth:-
Pred:+	35	288
Pred:-	15	662

Definitions: True/False Positives/Negatives

		True class	
	Total population	class positive	class negative
Predicted class	Predicted class positive	True positive	False positive (Type I error)
	Predicted class negative	False negative (Type II error)	True negative

- Careful! A “false positive” is actually a negative and a “false negative” is actually a positive.

Precision and Recall, F-measure

$$\text{Precision} = \frac{\#\text{true positive}}{\#\text{predicted positive}} \quad \text{Recall} = \frac{\#\text{true positive}}{\#\text{class positive}}$$

- In Information Retrieval, typically very few positives, many negatives. (E.g. billion webpages, dozen relevant to search query.) Focus is on correctly identifying positives.
- Recall: What proportion of the positives in the population do I correctly identify?
- Precision: What proportion of the instances I labeled positive are actually positive?
- $$\text{F-measure} = 2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

F-measure Example

- For the 0.5 threshold classifier, recall = 0, precision = 0, so F-measure = 0.
- For the 0.05 threshold classifier, F-measure = 0.188
- NOTE that F-measure is not “symmetric”; it depends on the definition of the positive class. Typically used when positive class is rare but important e.g. information retrieval.

Sensitivity and Specificity, Balanced Accuracy

$$\text{Sensitivity} = \frac{\#\text{true positive}}{\#\text{class positive}} \quad \text{Specificity} = \frac{\#\text{true negative}}{\#\text{class negative}}$$

- Sensitivity = #True positive / #Class positive
- Specificity = #True negative / #Class negative
  - Sensitivity: What proportion of the positives in the population do I correctly label?
  - Specificity: What proportion of the negatives in the population do I correctly label?
- $$\text{BalancedAccuracy} = \frac{1}{2}(\text{Sensitivity} + \text{Specificity})$$

Balanced accuracy Example

- For the 0.5 threshold classifier, sensitivity = 0, specificity = 1, balanced accuracy = 0.5.
- For the 0.05 threshold classifier, balanced accuracy = 0.698

Many Measures

- Different criterias want you to use different measures

		True class		
	Total population	class positive	class negative	Prevalence = $\frac{\Sigma \text{ class positive}}{\Sigma \text{ Total population}}$	
Predicted class	Predicted class positive	**True positive**	**False positive** (Type I error)	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted positive}}$
	Predicted class negative	**False negative** (Type II error)	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted negative}}$
	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	True positive rate (TPR), Sensitivity, Recall = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Class positive}}$	False positive rate (FPR), Fall-out = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Class negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Class positive}}$	True negative rate (TNR), Specificity (SPC) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Class negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$	

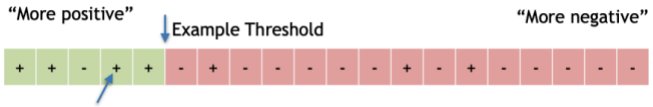
Rank-based Criteria

Summary

- Can only be applied to classifiers that can “score” or rank examples
- ROC focuses on true positive rate and false positive rate, better for somewhat balanced
- data
- Precision-Recall curve focuses on precision and recall, better for rare positive class
- Pay close attention to definitions

Ranks from Classifiers

- Suppose classifier can rank inputs according to “how positive” they appear to be.
- By adjusting the “threshold” value for deciding an instance is positive, we can obtain different confusion matrices.
- So really, rank-based techniques rely on label-based techniques.



Threshold	Truth:+	Truth:-
Pred:+	4	1
Pred:-	3	12

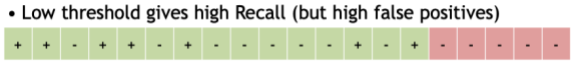
- Suppose classifier can rank inputs according to “how positive” they appear to be.
- By adjusting the “threshold” value for deciding an instance is positive, we can obtain different confusion matrices.
- So really, rank-based techniques rely on label-based techniques.

Threshold	Truth:+	Truth:-
Pred:+	5	3
Pred:-	2	10

Receiver operating characteristic (ROC)

• High threshold gives low Recall (but low false positives)

• Low threshold gives high Recall (but high false positives)

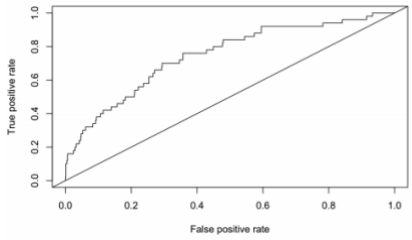


ROC curve: Try all possible thresholds,

True Positive Rate (Sensitivity, Recall) on y-axis. TP/P

False Positive Rate (1 - Specificity) on x-axis. FP/N

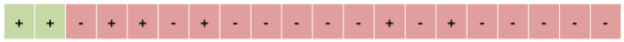
- Reading an ROC Curve
    - Think: "If I fix FPR at 0.4, what is my TPR?"
    - Obviously, higher is better. Random rankings give an ROC curve near the line withhigh probability.
    - If the area under the curve (AUROC) is 1, we have a perfect classifier. AUROC of 0.5 is worst possible.
    - Interpretation: If I randomly generate a positive example and a negative example what is the probability my classifier puts them "in the right order?" (c-statistic)



AUROC=0.799

Precision-Recall Curve (PRC)

• High threshold gives low Recall (but high Precision)



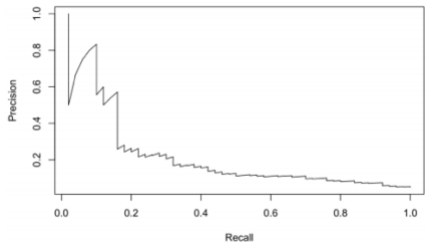
• Low threshold gives high Recall (but low Precision)

• PRC curve: Try all possible thresholds,

• Precision on y-axis. TP/(TP+FP)

• Recall (Sensitivity, True Positive Rate) on x-axis. FP/N

- Reading a Precision-Recall Curve
    - Think: "If I fix Recall at 0.2, what is my Precision?" Obviously, higher is better.
    - If the area under the curve (AUPRC) is 1, we have a perfect classifier.



Multivariate Models

Multivariate Regression

- As examples, we have stuck to models where predictions are a function of b0 + b1x
- It is straightforward to extend predictions to use an arbitrary number of features x0, x1, x2, . . . , xm

Vector notation

- x to denote a column vector of features, with the first one possibly always being to provide an intercept term
- b to denote a column vector of parameters of the same dimension as x
- For logistic regression
  - $p(\mathbf{x}; \mathbf{b}) = \varsigma(\mathbf{x}^\top \mathbf{b})$
  - the sum of the products of corresponding elements of the vectors, and produces a single number (scalar.)

Foreshadowing: Choosing Features

- When using many features relative to the amount of training data, it is possible to build models that fit the training data but do not generalize
- This is overfitting and will be an ongoing discussion in this course.

Multiclass Classification

- Many different methods; we'll discuss logistic regression here

Evaluation

- Present the whole K × K confusion matrix
- "Take turns" letting each class be the positive class, and average over classes

Multiclass Probabilistic Classifier

Define K classes, numbered $1, \ldots, K$

Define K binary random variables $Y_1, \ldots, Y_K$, where $y_k = 1$ if and only if the observed class is class k, and 0 otherwise. This is sometimes called *one-hot encoding*. Write a realization as $\mathbf{y}$

Define $p_k(\mathbf{x}; \mathbf{B}) = \Pr(Y_k = 1 \mid X = \mathbf{x})$ for $j = 1 \ldots k$

E.g., for binary logistic regression, if we consider class 1 to be positive and class 2 to be negative:

- $p_1(\mathbf{x}; \mathbf{B}) = p(\mathbf{x}; \mathbf{b})$
- $p_2(\mathbf{x}; \mathbf{B}) = 1 - p(\mathbf{x}; \mathbf{b})$

Log Likelihood for Multiclass

$\text{data} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$

$\mathbf{y}_i$ is the ith label, in one-hot encoding
$y_{ik} = 1$ if example i has class label k, otherwise it is 0

$$\ell(\mathbf{B}; \text{data}) = \sum_i \sum_k y_{ik} \log(\Pr(Y_{ik} = 1 \mid X = \mathbf{x}_i; \mathbf{B}))$$

Multiclass Probabilities

- Secret weapon: softmax (or softargmax)
  - Takes vector of real numbers, outputs vector of probabilities
  - Larger inputs give larger corresponding probabilities

$$\varsigma(\mathbf{z})_k = \frac{e^{z_k}}{\sum_j e^{z_j}}$$

-

- Generalization of sigmoid

Multiclass Logistic Regression

$$p_k(\mathbf{x};\mathbf{B}) = \varsigma(\mathbf{z})_k \quad \text{where} \quad z_k = \mathbf{x}^{\mathsf{T}}\mathbf{b}_k \text{ for } k = 1 \ldots K-1, \quad z_K = 0$$

- Basically a linear "score" for each class, except the last one. Same as in binary setting if last class is "0"

Evaluating Multiclass Classifiers

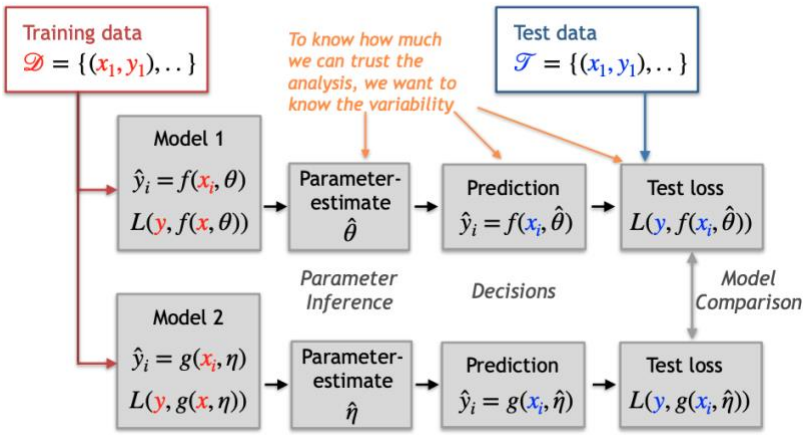
- Every classification performance measure we saw applies to each class.
- Consider the application - what information is useful to the user?
    - If manageable, providing the confusion matrix is good practice.
    - Otherwise, provide "average" quantities, where the "average" is taken over each class taking a turn as the "positive class."

Training versus Testing

- If we evaluate the model using the same data, estimates will have optimistic bias. Generally, this bias gets stronger the more features are used with a fixed amount of data.
    - This is called (among other things) double dipping.

W4

Supervised learning – overview.



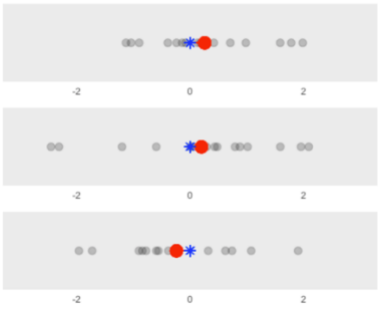
Statistics, Parameters, and Estimation

- A parameter is a value that characterizes the population that we want to study. Parameters are often expectations (like the mean) or values that describe the relationship between input and output (slope of a linear model).
- A statistic is any summary of a dataset. (E.g.$\bar{x}_n$)
- Estimation uses a statistic (e.g. $\bar{x}_n$) to estimate a parameter (μ_X) of the distribution of a random variable.
    - Estimate: value obtained from a specific dataset
    - Estimator: function (e.g. "sum-and-divide by n" or "compute the maximum likelihood estimate") used to compute the estimate
    - Estimand: parameter of interest

A simple parameter: mean

- A simple model is one that predicts the mean $\hat{y} = \theta$
- Given a ***dataset*** (collection of realizations) $x_1, x_2, \ldots, x_n$ of X, the estimate for this parameter is the **sample mean**:
- $\hat{\theta} = \frac{\sum_{i=1}^{n} x_i}{n}$
- Given a dataset, $\hat{\theta}$ is a fixed number.
- Across possible randomly drawn dataset of size n, the mean is a ***random variable*** $\bar{X}_n$

Sampling Distributions

- The distribution of an estimator is called its sampling distribution.
  - 

Bias

- The expected difference between estimator $(\hat{\theta})$ and estimand/parameter (θ)
- For example
  - $E[\bar{X}_n - \mu_X]$
    - Note: by convention $\mu_X = E[X]$, the mean of r.v. X.
    - If 0, estimator is **unbiased**.
  - Sometimes, $\bar{x}_n > \mu_X$, sometimes $\bar{x}_n < \mu_X$, but the long run average of these differences will be zero.

Variance

- The expected squared difference between estimator and its mean.
- For example
  - $E[(\bar{X}_n - E[\bar{X}_n])^2]$
    - Positive for all non-trivial estimators. Higher variance
    - means distribution of estimates is more "spread out."
  - Because $\bar{X}_n$ is unbiased, we can write $E[(\bar{X}_n - \mu_X)^2]$
  - Sometimes, $\bar{x}_n > \mu_X$, sometimes $\bar{x}_n < \mu_X$, but the ***squared differences*** are all positive and do not cancel out.
- Standard deviation is the square root of the variance.

Confidence intervals via Central Limit theorem

Central Limit Theorem

- Informally: For *many different distributions* of X, the sampling distribution of $\overline{X}_n$ is approximately normal if n is big enough.
- More formally, for X with finite variance:

$$\bar{x}_n \sim N\left(\mu, \sigma^2_{\bar{X}_n}\right)$$

where

$$\sigma_{\overline{X}_n} = \frac{\sigma_X}{\sqrt{n}}$$

is called the ***standard error of*** $\overline{X}_n$ and σ^2_X is the variance of X.

CLT Consequence

- 95% of the mass of a normal with mean μ and standard deviation σ is between $\mu - 1.96\sigma$ and $\mu + 1.96\sigma$
- 95% of the time, the sample mean $\overline{X}_n$ will be between
  - $\mu - 1.96\frac{\sigma_X}{\sqrt{n}}$ and $\mu + 1.96\frac{\sigma_X}{\sqrt{n}}$
- 95% of the time, the true mean μ will be between
  - $\bar{x}_n - 1.96\frac{\sigma_X}{\sqrt{n}}$ and $\bar{x}_n + 1.96\frac{\sigma_X}{\sqrt{n}}$

Confidence Intervals

- Typically, we specify confidence given by 1- α
- Use the sampling distribution to get an interval that traps the parameter (estimand) with probability 1- α
- We almost always use 1- α = 0.95

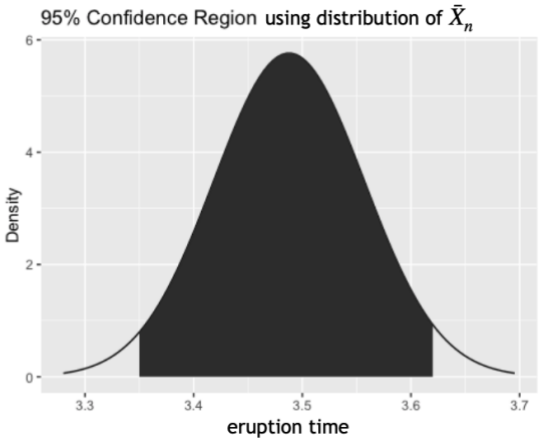
Quantifying Precision

- Eruptions dataset has n = 272 observations
- Our estimate of the mean of eruption times is $\overline{x}_{272}$ = 3.4877831

Mean = 3.49
StdDev = 1.14
n = 272

SE = $1.14/\sqrt{272}$
= 0.07

CI = 3.49 +- 1.96*0.07
CI = 3.49 +- 0.14



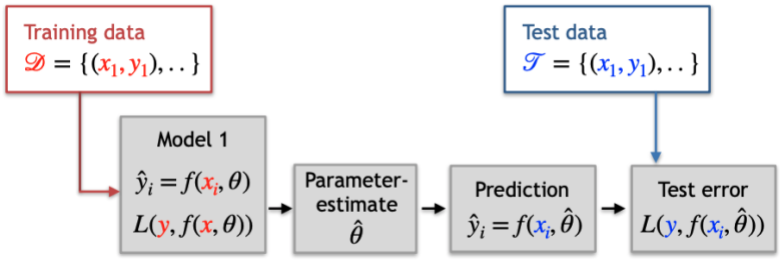
Aside: t versus z

- To make the CI, we had to estimate σ_X (or σ_l when talking about loss)
- This adds additional uncertainty
- Correct for this by using the t-distribution with n -1 degrees of freedom instead of
- normal.

- Doesn't really matter for bigger than about 30

Lab: Uncertainty of the test error

Prediction performance



- Ultimately, we want our model to work well on new data
- To test this, we can collect a new data or reserve a part of the data as test a test dataset

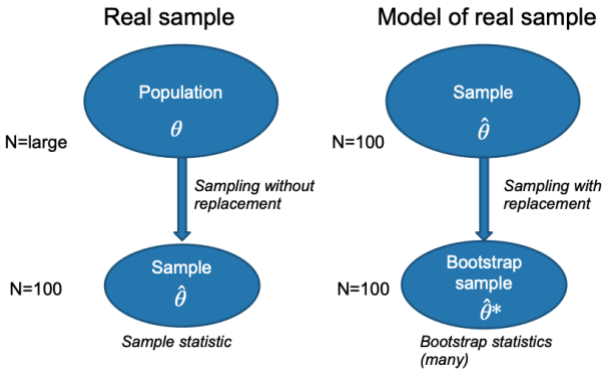
Test Error

Given a ***dataset*** (collection of realizations) $(x_1, y_1), \ldots, (x_n, y_n)$ of (X, Y), ***that were not used to find f*** the **test error** is: $l_{\mathcal{T}} = \frac{1}{n} \sum_i L(y_i, f(x_i, \hat{\theta}))$

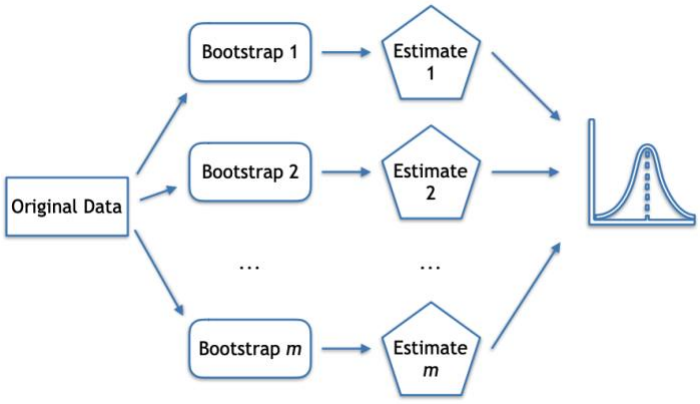
The Bootstrap

Summary

- Universal technique to obtain confidence intervals
- Can be applied to any statistics
- Confidence intervals are asymptotically correct
- Does not make assumptions about the underlying distribution
- Disadvantages
  - Extra programming
  - Can become very unstable with small N
  - Takes computation time
- For Linear models and data with approximately gaussian noise, the CLT provides excellent closed-form expressions for the distribution of parameter estimates
- However, bootstrap is a universal technique applicable to all data and models / sample statistics.
- Idea: Use the data we have to "pretend" we can sample more data



-



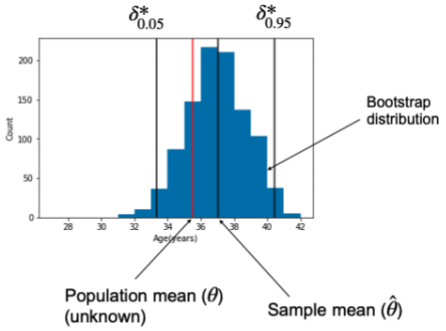
- Code

```
Your Sample S has N observations
For b in 1:numBootstrap:
     resample N from S with replacement -> S*
     Fit model to S* -> θ̂*(bootstrap statistics)
     Record your bootstrap statistics

     Return the distribution of bootstrap statistics
```

- Once you have your bootstrap distribution, you can construct confidence interval:

$\delta^* = \hat{\theta}^* - \hat{\theta}$

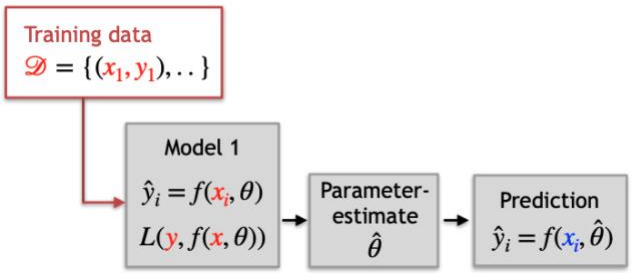
$\delta^*_{0.05}$: 5% percentile of bootstrap distribution



90% Confidence interval: $\hat{\theta}$ | $\delta^*_{0.95}$ | $\delta^*_{0.05}$ |

$$CI = \left[\hat{\theta} - \delta^*_{0.95}, \hat{\theta}^* - \delta^*_{0.05}\right]$$

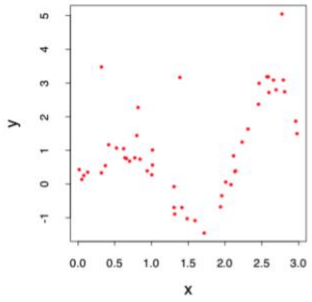
Prediction uncertainty



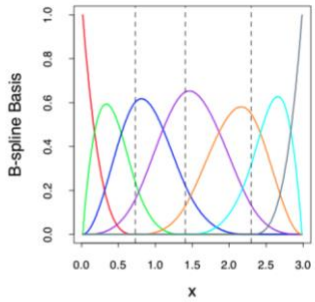
How does the uncertainty in the parameter estimates influence the uncertainty of the prediction?

How much would our prediction change, if we had drawn a different set of training data?

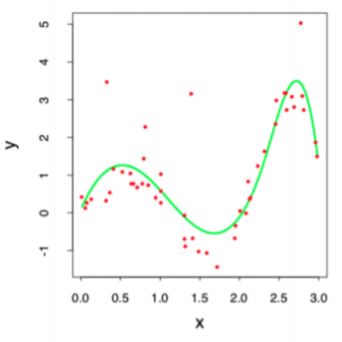
Prediction uncertainty



Say we have a data set with a clearly non-linear dependence between x and y

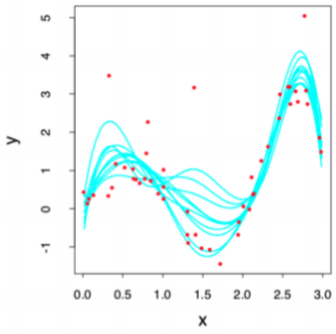


- We can define 7 nonlinear *basis functions* $b_i(x)$
- Here we use B-splines, but it could be any other (polynomials, gaussians, fourier-basis)
- We concatenate them into a design matrix X fit the model $\hat{y} = f(x) = X\theta$.

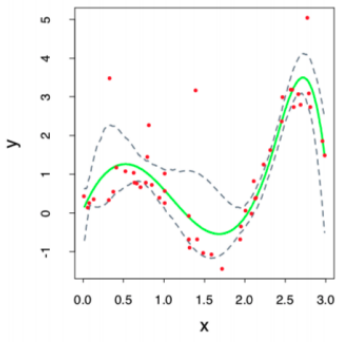


Our best fit on the training data $\hat{y} = \hat{f}(x) = X\hat{\theta}$ looks pretty good, but how certain can we be?

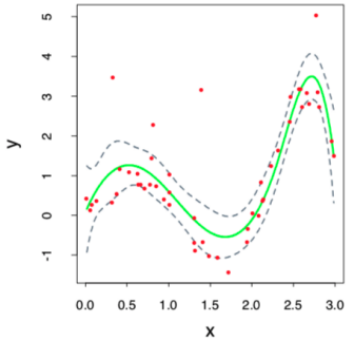
We do a bootstrap, each time getting a new sample, and each time getting a new parameter estimate $\hat{\theta}_b^*$



For each parameter vector, we can now plot a new prediction $\hat{y}_b^* = \hat{f}_b^*(x) = X\hat{\theta}_b^*$



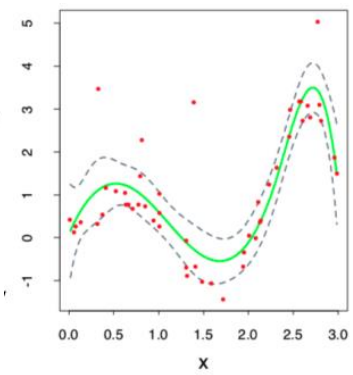
For each x we can then compute a 95% confidence interval based on $\hat{y}_b^*$



In the case of linear regression, we can also use the central limit theorem to get the CIs - as you can see they look fairly similar.

Prediction uncertainty

- Note that the confidence interval tells us that the true value ($f(x)$) is with 95% probability in this interval
- It does NOT mean that for a new new observation (x_n, y_n) is within the interval with 95% chance
- This is because y_n will differ from $f(x_n)$ by some random variability (σ^2)
- A confidence interval for new data (the *predictive density)* needs to take into account uncertainty of the predicted value, $var(\hat{f}(x_n))$, and the random variability (σ^2)



W5

Mean errors – loss function

Mean-squared error: $\mathrm{MSE} = n^{-1} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$

Root-mean-squared error: $\mathrm{RMSE} = \sqrt{n^{-1} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2}$

Mean absolute error: $\mathrm{MAE} = n^{-1} \sum_{i=1}^{n} |\hat{y}_i - y_i|$

Mean relative error: $\mathrm{MRE} = n^{-1} \sum_{i=1}^{n} \frac{|\hat{y}_i - y_i|}{|y_i|}$

Test Error

- Given a dataset(collection of realizations) (x1,y1), …, (xn,yn) of (X,Y), that were not used to find h the test error is

$$L_{\mathcal{D},\mathcal{T}} = \frac{1}{n} \sum_{i} L(y_i, f(x_i, \hat{\theta}))$$

-
- Given f and a set dataset, the test error is a fixed number
- From dataset to dataset, the test error will change and is a random variable
- We are interested in the expected test error
  - $E[L(y, f(x))]$

Conditional vs expected test error

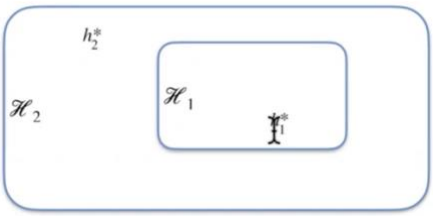
- Conditional test error: calculate the expectation over different test sets, given a specific training dataset
  - the average error that the model you fitted on a particular training set τ would incur when applied to examples drawn from the distribution of (X,Y) pairs.
  - Ex. If you lose money each time the fitted model makes an error (or proportional to the error if you're talking about regression), it's the average amount of money you lose each time you use the classifier.

    - The expected test error for the current estimated model
    - $Err_{\mathcal{D}} = E_{\mathcal{T}}(L(y_i, f(x_i, \hat{\theta})) \mid \mathcal{D})$
- Expected test error: the expectation is being taken over both training and test set
    - An average over all sorts of training sets that you're typically never going to get to use
    - $Err = E_{\mathcal{D},\mathcal{T}}(L(y_i, f(x_i, \hat{\theta})))$
    - The expected test error for an estimated model of this type

Bias-variance decomposition

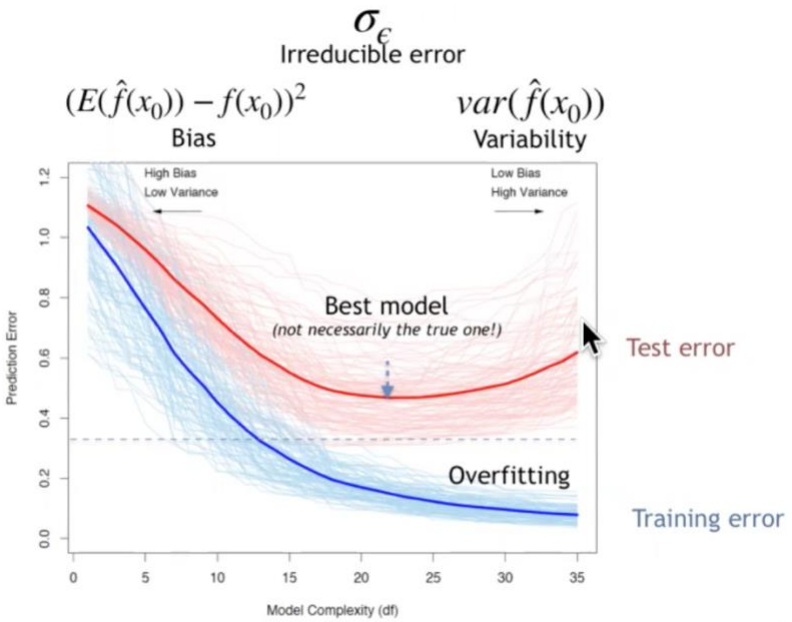
- Irreducible error: variability in the data that around the true relationship between x and y
    - $y_i = f(x_i) + \epsilon \longleftarrow \sigma_\epsilon^2$
    - can't be reduced by creating good models.
    - This is the test error we would have if we had the correct model class + infinite data to estimate – i.e. the test error if we knew f(x)
    - This is the best test error we could possibly expect
- Bias: systematic difference of the of the best fitted model from the true relationship
    - $E(\hat{f}(x_i)) - f(x_i)$
    - If the model is complex enough, bias starts going to zero
    - Underfitting the data → high bias
- Variance of the fit around the average fit
    - $E(\hat{f}(x_i) - E(\hat{f}(x_i)))^2$
    - Overfitting → high variance

Bias-variance trade-off

- Larger model spaces lead to lower training loss
- Suppose $\mathcal{H}_1$ is the space of all linear functions, $\mathcal{H}_2$ is the space of all quadratic functions. Note $\mathcal{H}_1 \subset \mathcal{H}_2$. (Why?)
    - Set x**2 to 0
- H2 can never be higher than h1
- It must be the case that $\frac{1}{n}\sum_{i=1}^{n} L(h_2^*(\mathbf{x}_i, y_i)) \leq \frac{1}{n}\sum_{i=1}^{n} L(h_1^*(\mathbf{x}_i, y_i))$
- 

Training vs test error

- Overfitting: while training error decreass, test error increases

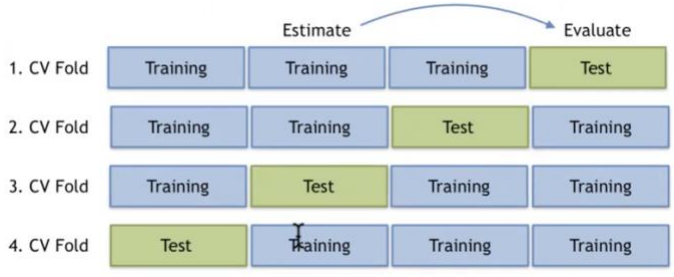


Size of test set

- The test error depends on the specific test set
- We would like to give confidence intervals for conditional test error (expected test error on a new test set)
- The confidence in your test error will depend on the size of test set
- $\bar{L}_n \pm 1.96\frac{\sigma_l}{\sqrt{n}}$ is an approximate 95% confidence interval for the expected loss.
    - σ_l: Standard deviation of test loss over observations in test set

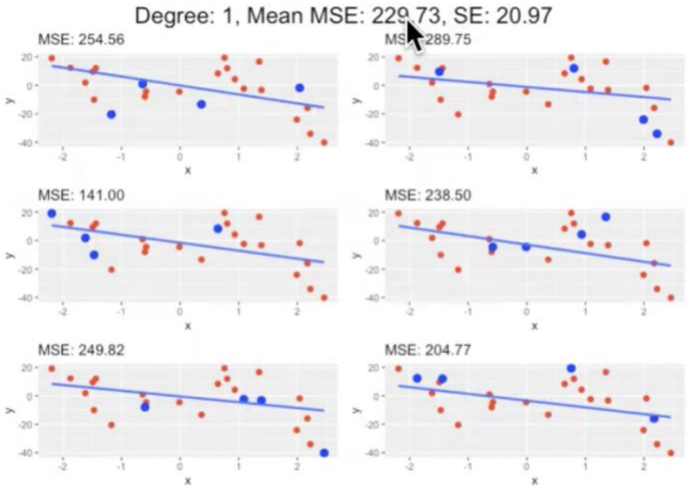
Cross validation

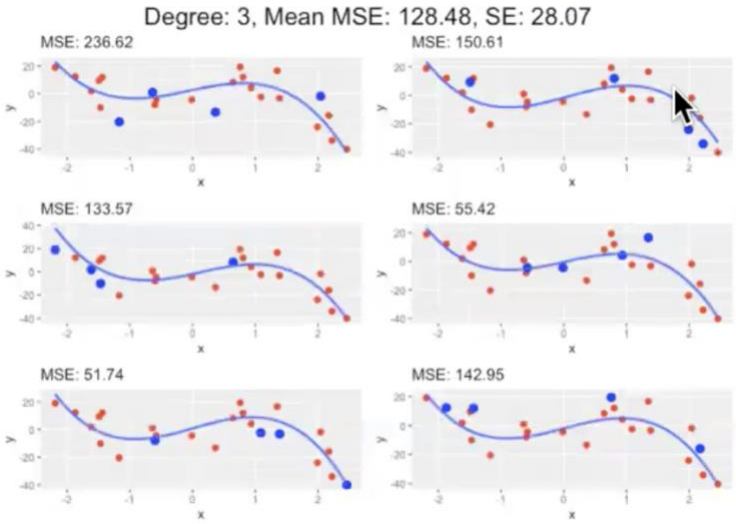
k-fold cross-validation

- Divide the instances into k disjoint partitions or folds of size n/k
- Loop through the partitions i= 1 … k:
    - Partition I is for evaluation
        - Estimating the performance of the algorithm after learning is done
    - The rest are used for training
        - Choosing the specific model within the space
- "cross-validation error" is the average error on the evaluation partitions. Has lower variance than error on one partition
- 4-fold cross-validation model selection
    - 

- Evaluated model fit is slightly different between folds
- In the end, you have an Li for each data point
- Error is averaged across all folds

- Ex. Cross-validation for different models



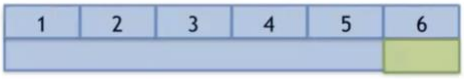


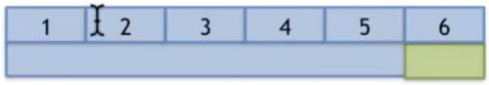
How many cross validation folds

	2-Fold CV (split-half)	K-Fold CV (often K=5-10)	N-Fold CV Leave-One-out
Overestimation bias of prediction error:	bad	present	nearly unbiased
Computational cost:	low	favourable	high
Variance of estimate:	low	low	high
Training sets are:	independent	similar	nearly identical

Other considerations

- Balanced classes (in y)
- Dependence of data

- Time-series data
  - 
- Within vs out-off sample (in x)
  - 

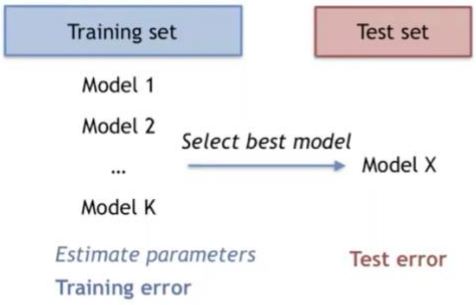
- Interpolation vs extrapolation
  - 

Model selection

- Select most parsimonious model whose error is no more than one standard error above the error of the best model

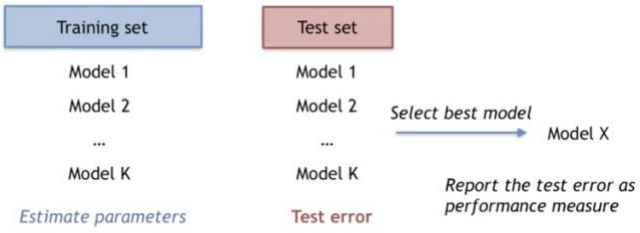
Model selection strategies

1. Choosing "best fitting" model



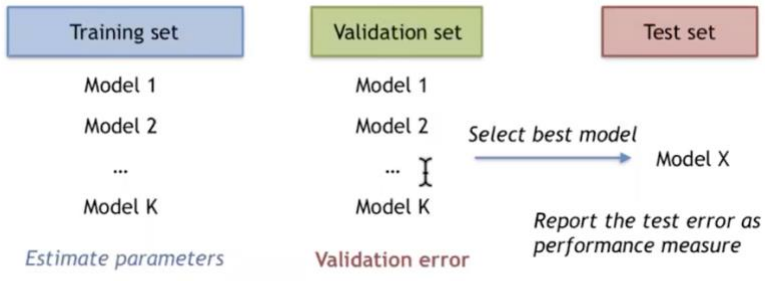
This strategy is bad, as it will select the most complex model -> overfitting

- Model selection based on penalized training error
  - Training error is biased downward
  - For simple models, including linear ones, we can get a less-biased estimate of generalization error by adjusting the training error upwards
  - These adjusted training error estimators includes: AIC, BIC, and Adjusted R-squared
  - If you have a limited amount of data, and you want to do model selection, you may want to use one of these instead of a validation set.
2. Choosing model with lowest test error



The test error for the best model is underestimated.
The underestimation can be quite dramatic if you have many models.
This is another form of "overfitting"

- Choosing the model with the lowest test error
    - Experimental scenario
        - Generate new data set
        - Split into training and test
        - Choose best model using performance on test set
        - Report performance of best model as a predictor for it's expected test error
    - Feature selection: find the best combination of features
    - If you have 100 features, you select from 2**100 models
    - Mortal sin 1: select the best features on all the data – then calculate CV or test error on that model – and report that error as performance metric
    - Mortal sin2: select the best features on the test (or CV) error – and report that error as performance metric
3. Doing model selection on the validation error



This is the standard strategy in machine learning.

- Training, model selection, and performance evaluation
    - The data are randomly partitioned into three disjoint subsets
        - A training set used only to find the parameters θ
        - A validation set used to find the right model space (e.g., the degree of the polynomial) – you can think of this decision as another set of parameters η
        - A test set used to estimate the generalization error of the selected model $M(\eta, \theta)$

W6

Feature Construction

- Construct new good features that are relevant and ready to be used in our machine learning model.

move beyond linearity

- replace/augment the vector of input features X with additional features, which are transformation of X, and then use linear models in this new space of derived input features
- $h_m(X) : \mathbb{R}^p \mapsto \mathbb{R}$ the m^{th} transformation X, m=1, …, M.

- $h_m(X)$: are sometimes called basis functions or feature functions. They define new features
    - The Feature space is typically more high-dimensional than the data space
    - $\hat{y} = \sum_{m=1}^{M} \beta_m h_m(\boldsymbol{X})$
- Some simple and widely used examples of the h_m
    - $h_m(X) = x_m$, m = 1, . . . , p recovers the original linear model.
    - $h_m(X) = x_m^2$ or $h_m(X) = x_m x_{m+1}$ → Allow us to augment the inputs with polynomial terms
    - $h_m(X) = \log(x_m)$, $\sqrt{x_m}$, …. → permits nonlinear transformations of single inputs

Polynomial Expansion

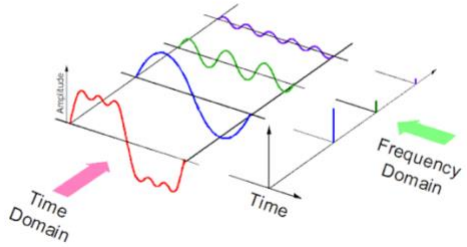
- **Single input:**
  Input: x
  Transformation:
  $h_0(x) = 1, \quad h_1(x) = x, \quad h_2(x) = x^2, \ldots\ldots, h_d(x) = x^d$
- **Multiple inputs:**
  Input: $\mathbf{X} = [x_1, x_2, x_3]$

  1st order - 3 features: x_1 x_2 x_3
  2nd order - 6 features: x_1^2 x_1x_2 x_2^2 x_2x_3 x_3^2 x_1x_3
  3rd order - 10 polynomial features: x_1^3 $x_1^2x_2$ $x_1x_2^2$ ..... $x_1x_2x_3$
  .....
- **Observations:**
    - Helps to capture non-linearities in regression models
    - May introduce high variance, especially near the boundaries of the data

Fourier Series

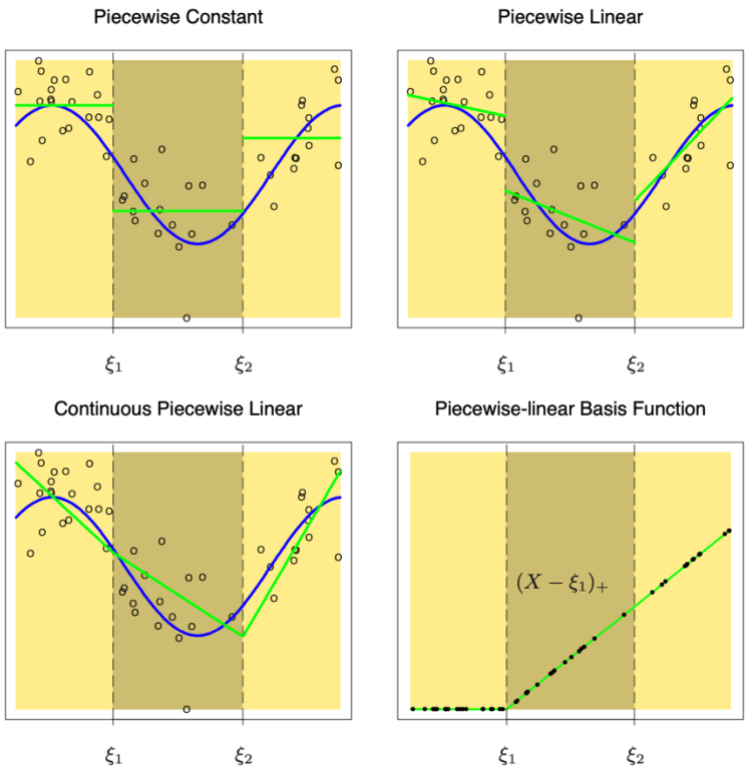
- Fourier Basis: Reasonable option for periodic data or data with know boundaries
- Example:

$$h_0(x) = 1 \qquad h_j(x) = \cos(\omega_j x + \psi_j) \qquad \text{for } j > 0$$



Piecewise Polynomials

- F(x) is obtained by dividing the domain of X into contiguous intervals, and representing f by a separate polynomial in each interval.



-

Why might you want to perform feature selection?

- Some or many of the features used in a multiple regression model are in fact not associated with the response.
- Irrelevant features leads to unnecessary complexity in the resulting model.
    - 10 M features: each prediction is expensive

Subset Selection:

- This approach involves identifying a subset of the k predictors that we believe to be related to the response.

Shrinkage (also known as regularization):

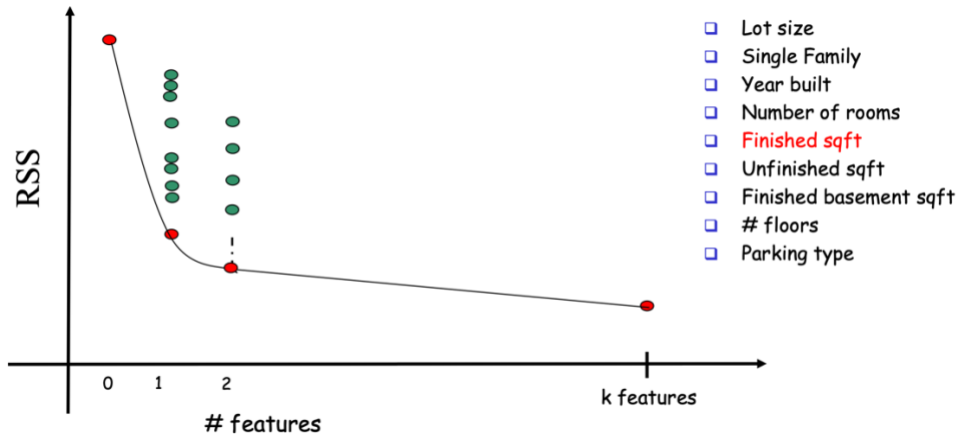
- This approach involves fitting a model involving all p predictors.
- However, the estimated coefficients are shrunken towards zero relative to the least squares estimates.
- This shrinkage has the effect of reducing variance.

Dimension Reduction:

- This approach involves projecting the k predictors into a M-dimensional subspace, where M < k.

All subsets

- Search over every possible combination of features we might want to include in our model and look at the performance of each of those models



-
- Complexity of "all subsets"
- How many models did we have to evaluate?
    - each indexed by features included (to indicate a feature is included or not)

      $\text{Complexity} \rightarrow 2^k$

      $k = 8 \rightarrow 256$ We have to search over 256 models

      $k = 30 \rightarrow 1{,}073{,}741{,}824$

      $k = 100 \rightarrow > 10^{30}$
    -

Best Subset Selection (Exhaustive Search over all the subsets)

- To perform best subset selection, we fit a separate least squares regression best subset for each possible combination of the k predictors.
    - we fit all k models selection that contain exactly one predictor Models that contain exactly two predictors,

      $\binom{k}{2} = \frac{k!}{2!\,(k-2)!} = \frac{k(k-1)}{2}$
    -

$\binom{k}{n} = \frac{k!}{n!\,(k-n)!}$

-
    - The set of all n-combinations of a set of size k
- K features → we are 2**k possibilities of combining them
    - $1 + k + \frac{k(k-1)}{2} + \binom{k}{2} + \binom{k}{2} + \ldots + 1 \approx 2^k$
- We then look at all of the resulting models, with the goal of identifying the one that is best.

Stepwise Selections

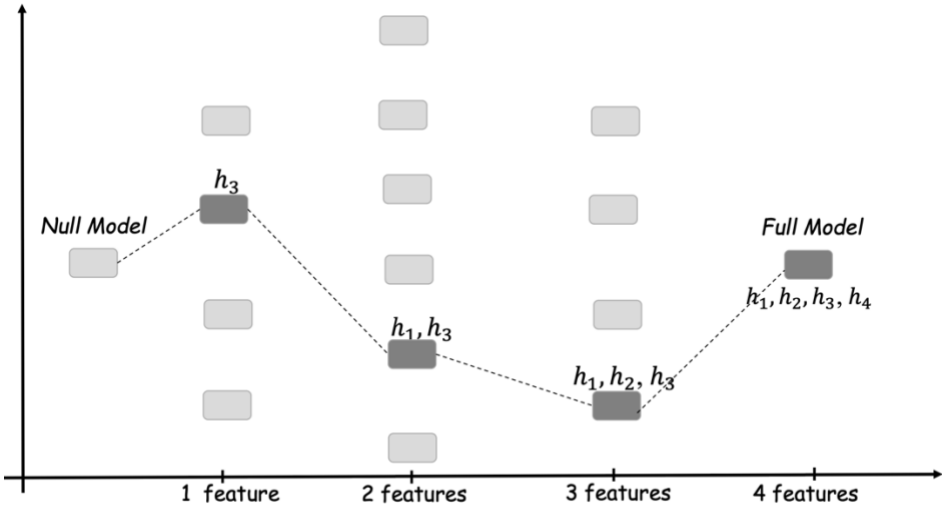
- Stepwise methods explore a far more restricted set of models: attractive alternatives to best subset selection
    - Forward Stepwise Selection:
    - Backward Stepwise Selection
- Both consider a much smaller set of models compared to the best subset selection
- Forward stepwise selection
    - Forward stepwise selection begins with a model containing no features, and then adds features to the model, one-at-a-time, until all of the features are in the model.

- At each step the feature that gives the greatest additional improvement to the fit is added to the model

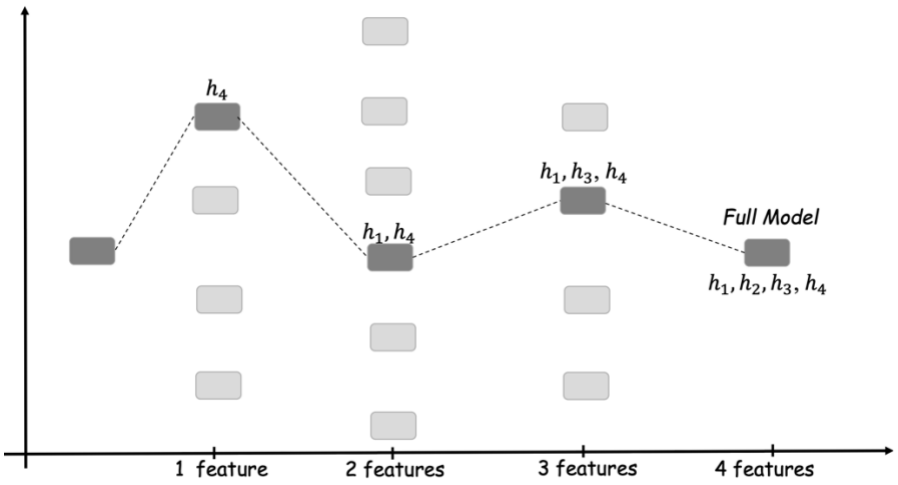
  Considers: $(k - j)$ models in the j^{th} iteration

  Number of possibilities:

  $$1 + \sum_{j=0}^{k} (k - j) = 1 + k + (k - 1) + .. + 1 = 1 + \frac{k(k+1)}{2}$$

- 

  - ❑ Select the best model (best CV error)

- Backward stepwise selection
  - Consider the Full Model first, which contains all features
  - Then iteratively removes the least useful feature, one-at-a-time.
  - Keep the best among the Models
  - 

Regularization

- Keep all features – Consider all features
- Constrains (regularizes) the coefficient estimates of the features – or may shrink the coefficient estimates towards zero

- It turns out that shrinking the coefficient estimates can significantly reduce their variance.
- Two best-known regularization techniques (shrinking the regression coefficients):
  - Ridge regression
  - Lasso regression
- Change the optimization criterion where we balance:
  - How well the function fits data
  - Magnitude of estimated coefficients
- Total cost = measure of fit + measure of magnitude of coefficients
  - This is a new measure of the quality of the mode

Ridge Regression

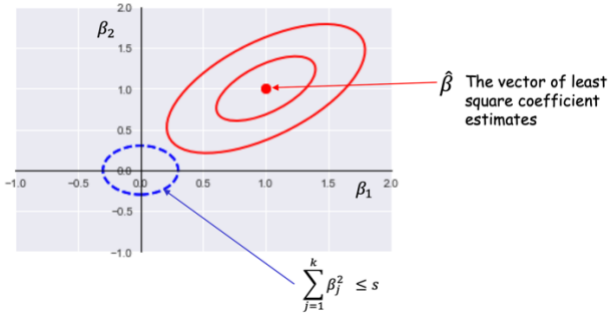
β_j are selected to minimize

$$J_{ridge} = \underbrace{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij}\right)^2}_{\text{RSS}} + \underbrace{\lambda\sum_{j=1}^{k}\beta_j^2}_{\text{Regularization Term}}$$

-
- λ is the regularization coefficient
  - If λ =0: no regularization
  - If λ= ∞ : $\beta_j = 0 \quad ,\forall\ j$
- Another Formulation for Ridge Regression

$$\min_{\beta}\ \left\{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij}\right)^2\right\}$$

  - $\text{subject to}: \sum_{j=1}^{k}\beta_j^2 \leq s$
  - For every value of λ there is some s such the equation above will give the ridge regression coefficient estimates
- Ridge regression penalty: results in a line with smaller slope
  - λ increase → slope becomes smother



  -

Scaling

- Un-regularized Regression:
  - Multiplying a feature by a constant c simply leads to a scaling of the least square coefficient estimates by a factor of 1/c.
  - Regardless of what jth feature is called $\sum_{i=1}^{n} x_{ij}\beta_j$ will remain the same

    - Scale equivariant
- Regularization:
    - Ridge regression coefficient estimates can change substantially when multiplying a given feature by a constant
    - $J_{ridge} = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{k}\beta_j^2$
- Since scaling changes the result, one strategy is to z-standardize features before submitting them to the model
    - $x = \frac{x - mean(x)}{std(x)}$
    - Another possibility is to have different regularization coefficients (λ) for different features (sometimes discussed as Tikhonov regularization)

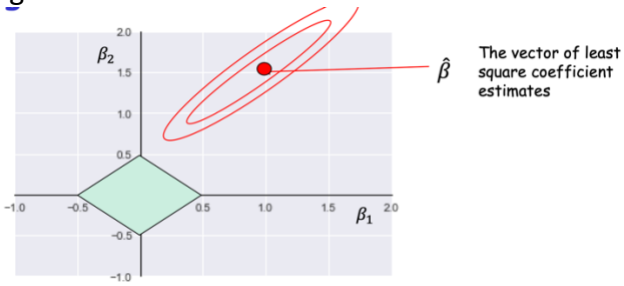
L1 Regularization: Lasso

- Lasso penalizes with the L1-norm:

$$J_{Lasso} = \underbrace{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij}\right)^2}_{\text{RSS}} + \underbrace{\lambda \sum_{j=1}^{k}|\beta_j|}_{\text{Regularization Term } (L_1 \text{ penalty})}$$

- Ridge Regression will always generate a model involving all features → increasing the value of λ will tend to reduce the magnitudes of the coefficients, but will not result in exclusion of any of the variables
- Lasso: the L1 penalty has the effect of forcing some of the coefficients estimates to be exactly equal to zero when the
- tuning parameter is sufficiently large.
- Another Formulation for Lasso
    - $\min_{\beta} \left\{\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij})^2\right\}$

      $\text{subject to} : \sum_{j=1}^{k}|\beta_j| \leq s$
- For every value of λ there is some s such the equation above will give the ridge regression coefficient estimates



    - The lasso coefficients estimates are given by the first point an ellipse contacts the constraint region
    - The green diamond represents the lasso constraints.
    - The lasso constraint has corners at each of the axes → the ellipse will often intersect the constraint region at the an axis → one of the coefficient will be zero

Lasso vs. Ridge Regression

- Often neither one is overall better.
- Lasso can set some coefficients to zero → performing feature selection, while ridge regression cannot.
- Both methods: as λ increases, the variance decreases and the bias increases.
- Lasso tends to do well if there are a small number of significant parameters and the others are close to zero.
- Ridge works well if there are many large parameters of about the same value (ergo: when most predictors impact the response).
- lasso performs variable selection, and hence results in models that are easier to interpret
- Note: in general we regularize: $\|\boldsymbol{\beta}\|^q$

Elastic Net

- Combine the penalties of ridge regression and lasso to get the best of both worlds.
- $$J_{Elastic\ Net} = \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij}\right)^2 + \lambda\left(\alpha\sum_{j=1}^{k}|\beta_j| + (1-\alpha)\sum_{j=1}^{k}\beta_j^2\right)$$
- α is the mixing parameter between ridge (α = 0) and lasso (α = 1).
- Two parameters to tune

How do we select λ?

- Implementing ridge regression and the lasso requires a method for selecting a value for the tuning parameter λ
- Cross-validation: A simple method to tackle this problem. We choose a grid of λ values, and compute the cross-validation error for each value of λ.
    - The tuning parameter with the smallest cross-validation error is chosen.

Ridge Regression vs. Least Squares

- Ridge regression's advantage: the bias-variance trade-off.
- λ increases → the flexibility of the ridge regression fit decreases, leading to decreased variance but increased bias
- In general, if the relationship between the response and the features is close to linear → the least squares estimates will have low bias but may have high variance
- Ridge regression works best in situations where the least squares estimates have high variance
- Ridge regression also has substantial computational advantages over best subset selection, which requires searching through 2**k models.
- Ridge regression (for any fixed value of λ) only fits a single model, and the model-fitting procedure can be performed quite quickly.

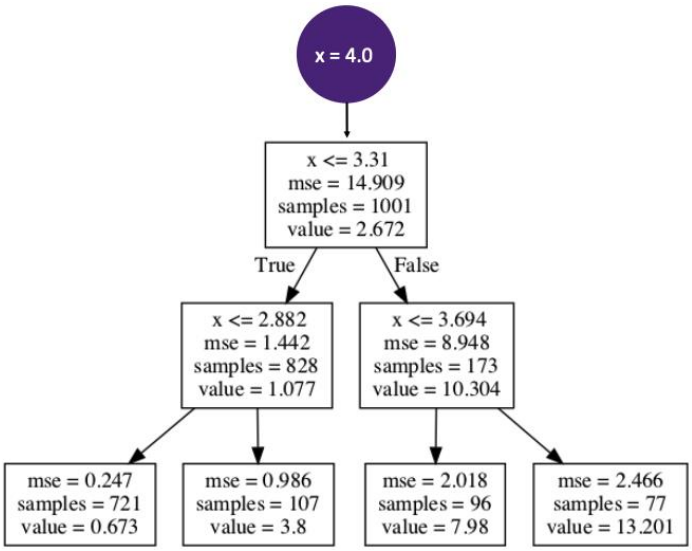
W7

Decision tree

- Trees recursively split the feature space into (hyper)-rectangles and fit a constant function to each (hyper)-rectangle

- Greedily decide where to split by determining which split leads to largest gain in accuracy/reduction in loss
- Terminate when
    - Made enough splits, or
    - Rectangles have some minimum number of observations

Visualizing The Process



More Dimensions

- Cut into rectangle

Trees Are High Variance

- Trees are an unstable model

Trees Lack Smoothness

- Recursive splits mean the resulting estimates are discontinuous.
- Performance in regression can be degraded in underlying processes is smooth
- MARS might be a better option in this case

Summary

- PROS
    - Trees are highly flexible!
    - Trees are non-parametric!
    - Trees are invariant to scale and can handle
    - categorical predictors naturally!
    - Trees are INTERPRETABLE and easy to
    - described to managers, marketers, etc.
- CONS
    - Tree starts to fit the noise (overfit) and not the signal when we go too deep. (Important to cross validate!)
    - Trees lack smoothness for regression
    - Sklearn has a great list of pros and cons should you need to look them up.

Ways to combat the overfitting

- Bagging
- Boosting

- Creating a Random Forest

Bagging

- Bootstrap Aggregation

Bootstrap B datasets

- For b =1…B, fit a deep tree (high variance)
- Combine trees and average B predictions
- Can be done for other estimators too!
- Calculating the output

  Now, take an element $x_{new} = (x_1, x_2, x_3, \dots, x_n)$.

  To get an estimate for the case:

  1. Take the bag of estimators $\{t_j(x_{new})\}_{j=1}^{B}$, with B the total number of trees you just calculated.
  2. For each tree t_j, calculate its estimate $y_j = t_j(x_{new})$.
  3. The final output (prediction!) is the average of all trees

  $$y_{new} = \sum_{j}^{B} \frac{t_j(x_{new})}{B}$$

  -
  - Regression → average
  - Classification → Vote
- Why Does This Work?

  When you average random variables each with variance σ^2 ,then the variance of the average looks like σ^2/B

  - As $B \longrightarrow \infty$, variance of the average becomes smaller

Bagging For Classification

- Suppose problem is to predict one of K classes
- Trees in the bag can vote for which class
- Result is a vector of proportions [p1, p2, p3, …] where pi is the proportion of the B trees voting for class i
- Important: You might be tempted to treat these as probabilities. They are not.
- If bagged classifier has smooth decision function, just average the decision function values. Usually leads to lower variance
- There is an elegant mathematical argument that bagging will reduce MSE in regression (pg. 285)
- In classification, not always true; bagging good classifier will can make it better, while bagging a bad one can make it worse.
- Furthermore, interpretation is lost as bagged trees are not trees anymore.

Summary of Bagging

- Reduce variance by adding adding more trees and bootstrapping
- Result is B trees, each grown on a bootstrapped version of the training data
- Predictions from each of the B trees is then averaged and used as a final prediction

Problem

- Correlation
  - Suppose that we fit a bagged estimator onto data with lots of features

    - Suppose further that just a handful of those features are important
    - All B trees will make splits based on that handful of features
    - All B trees are correlated, thus benefits of averaging are limited, and variance of average is $\rho\sigma^2 + (1-\rho)\sigma^2/B$

Random Forest

- Solve correlation problem from bagging

a bagged estimator

- Instead of showing the estimator all the features, I only show it a random subset

Bootstrap B datasets.

- Grow B trees
- At each split, allow the tree to decide where to split based on a random sample of the features
- Average predictions from the B trees
- The main idea is to reduce variance in bagging by reducing correlation of the trees. Achieved by
- showing subset of features

Why Does RF Further Reduce Correlation

- Simply, pairs of tree predictions tend to be less correlated if they do not use the same splitting variables.
- If we show each tree fewer variables, then there is less overlap in the features used to split on between trees.

Applied Preference

- Random forests are popular among data scientists because:
    - They are easily parallelized
    - Low bias, and lower variance than something like a tree or bagging
    - We can examine how important a feature was to predicting the data

Feature Importance

- The improvement in the split criterion is the importance measure attributed to the feature.
- Aggregate the improvement across all features. Once model is trained, make a plot like this
- More sophisticated approaches can calculate the average loss reduction after adding an element.
- You can even do this with profit or costs.

Out of Bag Error

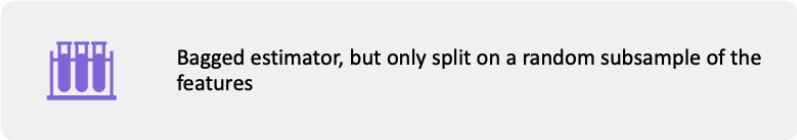
- Not every observation (x,y) appears in each bootstrapped dataset.
- If we use the trees that did not see (x,y) to make predictions on x, we can essentially validate
- how well our model is performing.
- Call this the “Out of Bag” (OOB) error.
- An OOB error estimate is almost identical to that obtained by N-fold cross- validation. Unlike many other nonlinear estimators, random forests can be fit in one sequence, with crossvalidation being per- formed along the way.

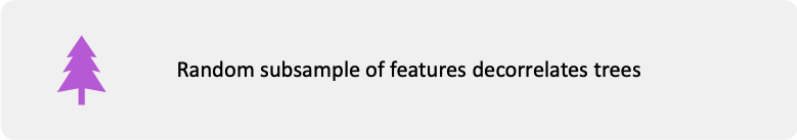
- Once the OOB error stabilizes, the training can be terminated.

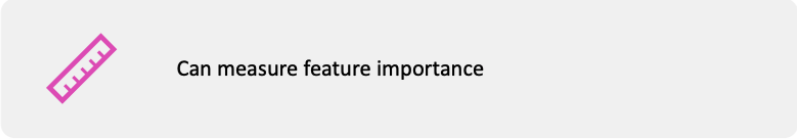
Can a RF Overfit?

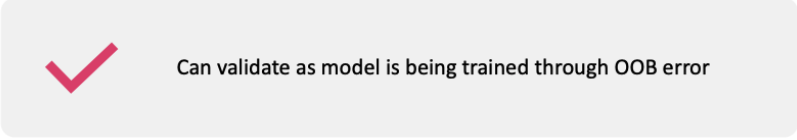
- Yes (although in a perfect world it should not)
- The average of fully grown trees can result in too rich a model, and incur unnecessary variance
- Small gains in performance by controlling the depths of the individual trees grown in random forests.
- Using full-grown trees seldom costs much, and results in one less tuning parameter.

Summary of Random Forests









Stochastic Gradient Boosting

Boosting

- going too deep in trees leads to overfitting
- not going far enough means we can't learn complex structure
- Boosting applies a weak learner (usually a tree with one split, called a stump) in a very neat way in order to learn complex structure

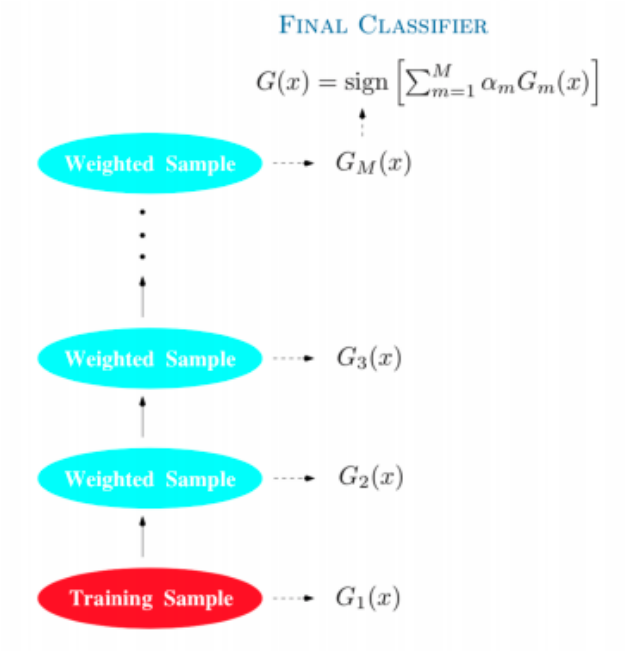
AdaBoost.M1

- Fit a weak learner to the sample
- Reweight the observations
- Fit a weak learner to the reweighed sample
- Weight the predictions of the learners

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

-



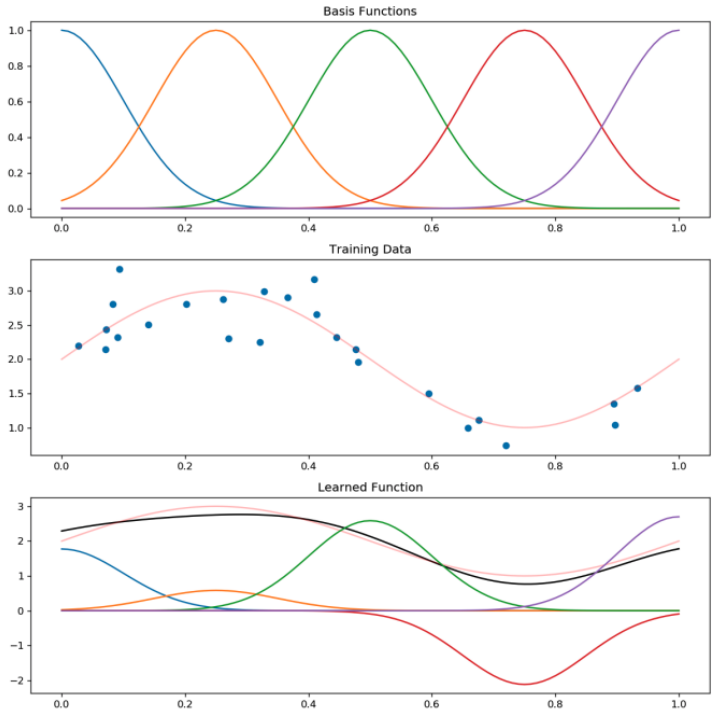
-

Boosting

- It works great for smallish sample sizes compared to Random Forests. It can be flexible and accurate with little work.
- Libraries like xgboost and lightgbm have fast implementations and APIs very similar to sklearn; no need to learn anything new

Boosting Fits an Additive Model

- An additive model is one which is expressed as
  $$g(x) = \beta_0 + \sum_i f_i(\mathbf{x})$$
  ○
- Basis functions are simple functions which can be combined together to fit complex structure.
- You are already familiar with the radial basis function.
- Taking linear combinations of basis functions allows us to get flexible fits



- In boosting, the basis functions are the weak learners
- The basis function expansion looks like
  - $$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m),$$
- To fit this model, we need to solve
  - $$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)\right).$$

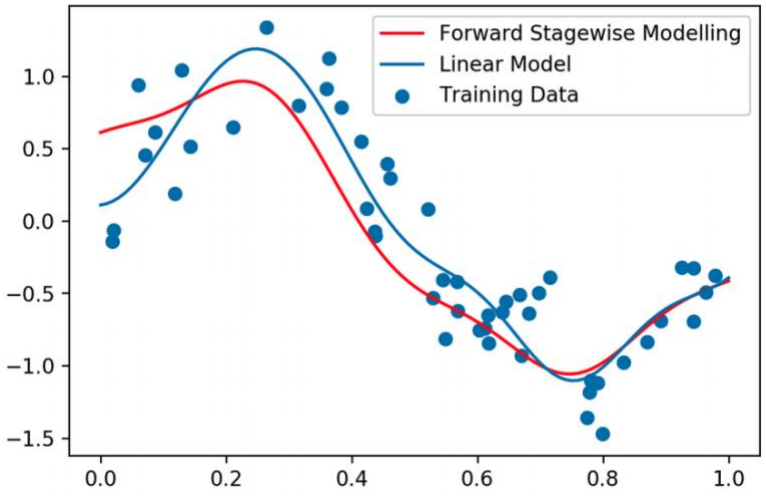
Forward Stagewise Additive Modelling

- Minimizing the loss can be computationally intensive, so we use “forward stagewise modelling”.
- Approximate the solution by sequentially adding new basis functions to the expansion without adjusting parameters and coefficients of basis functions already fit
  - **Algorithm 10.2** *Forward Stagewise Additive Modeling.*
    1. Initialize $f_0(x) = 0$.
    2. For $m = 1$ to M:
       (a) Compute
       $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$
       (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.
- When using squared error loss, some nice math happens
  - $$L(y, f(x)) = (y - f(x))^2,$$

$$
\begin{aligned}
L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i;\gamma))^2 \\
&= (r_{im} - \beta b(x_i;\gamma))^2,
\end{aligned}
$$

- So for squared error, we predict with our existing model, get the residuals, and then fit the a tree to the residuals. Then, that tree becomes part of our model
- Not generally the procedure when loss is different from squared error

```
def rbf(x, a, gamma):
    y =  np.exp(-(x-a)**2/gamma)
    return y

# Generate data
np.random.seed(0)
x = np.random.uniform(0,1,50)
X = np.linspace(0,1,101) #Going to predict with this later
y = np.sin(2*np.pi*x) + np.random.normal(0, 0.3, size = x.size)

f = 0
M = 5 #Number of basis functions

#Set up my basis functions
alphas = np.linspace(0,1,M)
g = 0.025
b = [rbf(x,a,g).reshape(-1,1) for a in alphas]

model = LinearRegression(fit_intercept=False)
ypred = 0 #Will update my predictions as we go

for i in range(M):

    model.fit(b[i], y-f)
    f += model.predict(b[i])
    c = model.coef_[0]
    ypred += c*rbf(X,alphas[i], g)
```



Boosted Trees

- For classification setting with exponential loss, forward stagewise additive modelling yields the adaboost algorithm
- For regression using squared error loss, the next tree we add is the tree which best fits our residuals (we will see this is also the case with gradient boosting).
- Forward stagewise modelling is a greedy processes, so solutions we obtain are a greedy approximation to the true minimizer.

Boosting Summary

Deep trees can overfit, but stumps have high bias.

If we fit an additive model through Forward Stagewise Additive Modelling, we can ensemble weak learners to make a flexible learner

If we are using MSE as our loss, Forward Stagewise Additive Modelling boils down to fitting a tree to the residuals at each step.

Summary of Methods

- Boosting > Random Forest > Bagging > Decision Trees as per Nested Spheres experiment.
- There is one more method, Gradient Boosting, which is fast and can be very accurate.

Gradient Descent

- Gradient descent seeks a local optima by performing
  - $\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha\nabla f(\mathbf{x}_n) = (\mathbf{x}_{n-1} - \alpha\nabla f(\mathbf{x}_{n-1})) - \alpha\nabla f(\mathbf{x}_n) = \mathbf{x_0} - \sum_i \alpha\nabla f(\mathbf{x}_i)$
- Here, f is a loss function, alpha is the learning rate
- Gradient descent is a “greedy” strategy, as it moves in the direction of steepest descent given its
- current position. Other movements could lead to smaller loss, but we don’t take those.

Boosting is Like Gradient Descent

- Forward stagewise additive modelling is also very greedy. At each step, the solution tree is the one that maximally reduces the loss.
- Thus, tree predictions in each boosting iterations are kind of like the components of the gradient; they move is in the direction where loss is reduced maximally.
- Gradient boosting fits a tree to the negative gradient values of the loss using least squares and a weak learner (e.g. a stump).
- When we use squared error loss, the gradients are the residuals (remember lecture 1?)

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

Gradient Boosting Summary

- Boosting fits weak learners through Forward Stage Additive Modelling
- Gradient boosting fits weak learners to the gradient of the loss
- This emulates something like gradient descent.