Linear Regression

Machine Learning – CSE546
Sham Kakade
University of Washington

Oct 4, 2016

Announcements:

Write your own code.

- TA office hours posted on website
- Recitation this week: Python
- HW1 posted
- Addcodes: decided by tomorrow. You will be contacted by email.

- Today:
  - MLE continued
  - Regression

What about continuous variables?

- Billionaire says: If I am measuring a continuous variable, what can you do for me?
- **You say: Let me tell you about Gaussians…**

$$P(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Some properties of Gaussians

- affine transformation (multiplying by scalar and adding a constant)
  - $X \sim N(\mu,\sigma^2)$
  - $Y = aX + b \rightarrow Y \sim N(a\mu+b, a^2\sigma^2)$

- Sum of Gaussians
  - $X \sim N(\mu_X,\sigma^2_X)$
  - $Y \sim N(\mu_Y,\sigma^2_Y)$
  - $Z = X+Y \rightarrow Z \sim N(\mu_X+\mu_Y, \sigma^2_X+\sigma^2_Y)$

Learning a Gaussian

- Collect a bunch of data
  - Hopefully, i.i.d. samples
  - e.g., exam scores
- Learn parameters
  - Mean
  - Variance

$x_1 \quad 98$, $\vdots \quad 95$, $\vdots$, $x_N \quad 32$

also ← MLE

$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i$

(why?)

$$P(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

MLE for Gaussian

$\Pr(x_1 \ldots x_n \mid \mu, \sigma) = \Pr(x_1 \mid \mu\sigma) \cdots \Pr(x_n \mid \mu\sigma)$ by i.i.d.

- Prob. of i.i.d. samples $D=\{x_1,\ldots,x_N\}$:

$$P(\mathcal{D} \mid \mu, \sigma) = \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N \prod_{i=1}^{N} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}$$

$\hat{\mu}_{MLE}, \hat{\sigma}_{MLE} = \arg\max_{\mu,\sigma} \ln \Pr(D \mid \mu, \sigma)$

- Log-likelihood of data:

$$\begin{aligned}\ln P(\mathcal{D} \mid \mu, \sigma) &= \ln\left[\left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N \prod_{i=1}^{N} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}\right] \\ &= -N \ln \sigma\sqrt{2\pi} - \sum_{i=1}^{N} \frac{(x_i-\mu)^2}{2\sigma^2}\end{aligned}$$

Your second learning algorithm: MLE for mean of a Gaussian

- What's MLE for mean?

$$\frac{d}{d\mu} \ln P(\mathcal{D} \mid \mu, \sigma) = \frac{d}{d\mu}\left[-N \ln \cancel{\sigma\sqrt{2\pi}} - \sum_{i=1}^{N} \frac{(x_i-\mu)^2}{2\sigma^2}\right]$$

$$-\sum \frac{d}{d\mu} \frac{(x_i-\mu)^2}{2\sigma^2} = -\sum_{i=1}^{N} \frac{-x_i+\mu}{\sigma^2} = 0$$

$$\hat{\mu} = \frac{\sum_{i=1}^{N} x_i}{N} \quad \checkmark$$

MLE for variance

- Again, set derivative to zero:

$$\begin{aligned}\frac{d}{d\sigma} \ln P(\mathcal{D} \mid \mu, \sigma) &= \frac{d}{d\sigma}\left[-N \ln \sigma\sqrt{2\pi} - \sum_{i=1}^{N} \frac{(x_i-\mu)^2}{2\sigma^2}\right] \\ &= \frac{d}{d\sigma}\left[-N \ln \sigma\sqrt{2\pi}\right] - \sum_{i=1}^{N} \frac{d}{d\sigma}\left[\frac{(x_i-\mu)^2}{2\sigma^2}\right]\end{aligned}$$

$-\frac{N}{\sigma} \quad - \quad \sigma^{-3}$

$$\hat{\sigma}^2_{MLE} = \frac{1}{N}\sum_{i=1}^{N} (x_i - \hat{\mu}_{MLE})^2$$

Learning Gaussian parameters

- MLE:

$$\hat{\mu}_{MLE} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\hat{\sigma}^2_{MLE} = \frac{1}{N}\sum_{i=1}^{N} (x_i - \hat{\mu})^2$$

- BTW. MLE for the variance of a Gaussian is **biased**
  - Expected result of estimation is **not** true parameter!
  - Unbiased variance estimator:

$$\hat{\sigma}^2_{unbiased} = \frac{1}{N-1}\sum_{i=1}^{N} (x_i - \hat{\mu}_{MLE})^2$$

Prediction of continuous variables

- Billionaire says: Wait, that's not what I meant!
- You say: Chill out, dude.
- She says: I want to predict a continuous variable for continuous inputs: I want to predict salaries from GPA.
- You say: **I can regress that…**

Linear The regression problem

input target/output

- **Instances:** <$\mathbf{x}_j$, t_j>
- **Learn:** Mapping from x to t(**x**)
- **Hypothesis space:**
  - Given, basis functions $H = \{h_1, \ldots, h_K\}$
  - Find coeffs $\mathbf{w}=\{w_1,\ldots,w_k\}$ $\underbrace{t(\mathbf{x})}_{\text{data}} \approx \hat{f}(\mathbf{x}) = \sum_i w_i h_i(\mathbf{x})$

goal is to approx/estimate $t(x)$

  - Why is this called linear regression???
    - model is linear in the parameters

$\mathcal{H} = \{ g(x) \mid g(x) = \sum w_i h_i(x) \text{ for } w \in \mathbb{R}^k \}$

- Precisely, minimize the residual squared error:

large estimate

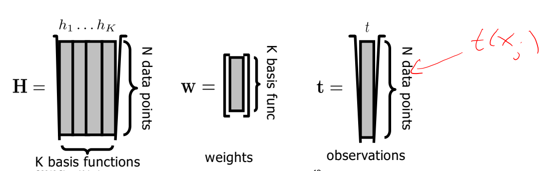
$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{j=1}^{N} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 \quad [Hw]_j$$

The regression problem in matrix notation

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

$\|$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \underbrace{(\mathbf{H}\mathbf{w} - \mathbf{t})^T(\mathbf{H}\mathbf{w} - \mathbf{t})}_{\text{residual error}}$$



Minimizing the Residual

scalar case $\frac{d}{dw}(xw-t)(xw-t)$

$= 2x(xw-t)$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \underbrace{(\mathbf{Hw}-\mathbf{t})^T(\mathbf{Hw}-\mathbf{t})}_{\text{residual error}}$$

$\nabla_w F(w) = 0$

$= 2H^T(Hw-t) = 0$

$H^T H w - H^T t = 0$

$\boxed{w^* = (H^T H)^{-1} H^T t}$

in matrix case

$\nabla_w (Hw-t)^T(Hw-t)$

$= 2H^T(Hw-t)$

Regression solution = simple matrix operations

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \underbrace{(\mathbf{Hw}-\mathbf{t})^T(\mathbf{Hw}-\mathbf{t})}_{\text{residual error}}$$

$$\text{solution: } \mathbf{w}^* = \underbrace{(\mathbf{H}^T\mathbf{H})^{-1}}_{\mathbf{A}^{-1}} \underbrace{\mathbf{H}^T\mathbf{t}}_{\mathbf{b}} = \mathbf{A}^{-1}\mathbf{b}$$

where $\mathbf{A} = \mathbf{H}^T\mathbf{H} =$ [k×k matrix grid] $\mathbf{b} = \mathbf{H}^T\mathbf{t} =$ [k×1 vector]

k×k matrix
for k basis functions

k×1 vector

But, why?

- Billionaire again, she says: Why sum squared error???
- You say: Gaussians, Gaussians…

  assume $\varepsilon_x \sim N(0, \sigma^2)$

- Model: prediction is linear function plus Gaussian noise
  - $t(\mathbf{x}) = \sum_i w_i h_i(\mathbf{x}) + \varepsilon_{\mathbf{x}}$

  · $Pr(t(x) \mid x) = N(\sum_i w_i h_i(x), \sigma^2)$

  · $t(x)$ i.i.d.

  one point

- Learn **w** using MLE

$$P(t \mid \mathbf{x}, \mathbf{w}, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-[t-\sum_i w_i h_i(\mathbf{x})]^2}{2\sigma^2}}$$

Maximizing log-likelihood

$\rightarrow Pr((x_1, t(x_1)) \mid w, \sigma) \cdots Pr((x_n, t(x_n)) \mid w, \sigma)$

Maximize:

$$\ln P(\mathcal{D} \mid \mathbf{w}, \sigma) = \ln\left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N \prod_{j=1}^N e^{\frac{-[t_j-\sum_i w_i h_i(\mathbf{x}_j)]^2}{2\sigma^2}}$$

$$\arg\max_w \left[\underbrace{\ln(\ )}_{\text{const}} - \frac{\sum_{j=1}^N (t(x_j) - \sum_i w_i h_i(x_j))^2}{2\sigma^2} \right]$$

$$= \arg\min_w \sum_{j=1}^N (t(x_j) - \sum_i w_i h_i(x_j))^2$$

Least-squares Linear Regression is MLE for Gaussians!!!

Bias-Variance Tradeoff

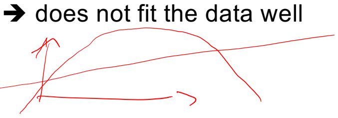
Machine Learning – CSE546
Sham Kakade
University of Washington

Oct 4, 2016

Bias-Variance tradeoff – Intuition

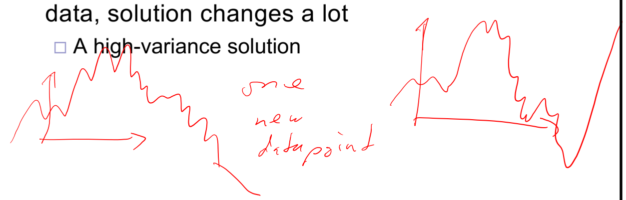
- Model too "simple" ➔ does not fit the data well
  - A biased solution

best fit line



- Model too complex ➔ small changes to the data, solution changes a lot
  - A high-variance solution

one new datapoint



(Squared) Bias of learner

$D \rightarrow$ learn h_D

- Given dataset *D* with *N* samples, learn function $h_D(x)$
- If you sample a different dataset *D'* with *N* samples, you will learn different $h_D'(x)$
- **Expected hypothesis**: $E_D[h_D(x)|x]$ $E_D[h_D(x)|X]$

  $\bar{h}_N =$
- **Bias:** difference between what you expect to learn and truth
  - Measures how well you expect to represent true solution
  - Decreases with more complex model $(f(x) - \bar{h}_N(x))^2$
  - Bias² at one point *x*:
  - Average Bias²: $E_x[(f(x) - \bar{h}_N(x))^2]$

Variance of learner

$D \rightarrow h_D.$

- Given dataset *D* with *N* samples, learn function $h_D(x)$
- If you sample a different dataset *D'* with *N* samples, you will learn different $h_D'(x)$
- **Variance:** difference between what you expect to learn and what you learn from a particular dataset
  - Measures how sensitive learner is to specific dataset
  - Decreases with simpler model
  - Variance at one point *x*: $\rightarrow E_D[(h_D(x) - \bar{h}_N(x))^2|X]$
  - Average variance: $E_X E_D[(h_D(x) - \bar{h}_N(x))^2].$

Bias-Variance Tradeoff

- Choice of hypothesis class introduces learning bias
  - More complex class → less bias
  - More complex class → more variance

Bias-Variance Decomposition of Error

$\bar{h}_N(x) = E_D[h_D(x)]$

- Expected mean squared error: $\text{MSE} = E_D\left[E_x\left[(t(x) - h_D(x))^2\right]\right]$
- To simplify derivation, drop x:
- Expanding the square:

Moral of the Story: Bias-Variance Tradeoff Key in ML

- Error can be decomposed:

$$\text{MSE} = E_D\left[E_x\left[(t(x) - h_D(x))^2\right]\right]$$
$$= E_x\left[(t(x) - \bar{h}_N(x))^2\right] + E_D\left[E_x\left[(\bar{h}(x) - h_D(x))^2\right]\right]$$

- Choice of hypothesis class introduces learning bias
  - More complex class → less bias
  - More complex class → more variance

What you need to know

- Regression
  - Basis function = features
  - Optimizing sum squared error
  - Relationship between regression and Gaussians
- Bias-variance trade-off
- Play with Applet

Overfitting

Machine Learning – CSE546
Sham Kakade
University of Washington

Oct 4, 2016

Bias-Variance Tradeoff

- Choice of hypothesis class introduces learning bias
  - More complex class → less bias
  - More complex class → more variance

Training set error

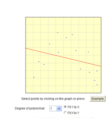
$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

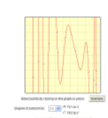
- Given a dataset (Training data)
- Choose a loss function
  - e.g., squared error (L_2) for regression
- **Training set error:** For a particular set of parameters, loss function on training data:

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

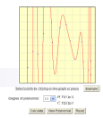
Training set error as a function of model complexity

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$





Prediction error



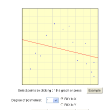
- Training set error can be poor measure of "quality" of solution
- **Prediction error:** We really care about error over all possible input points, not just training data:

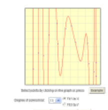
$$error_{true}(\mathbf{w}) = E_{\mathbf{x}}\left[\left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2\right]$$
$$= \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2 p(\mathbf{x})d\mathbf{x}$$

Prediction error as a function of model complexity

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2 p(\mathbf{x})d\mathbf{x}$$





Computing prediction error

- Computing prediction
  - Hard integral
  - May not know t(**x**) for every **x**

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2 p(\mathbf{x})d\mathbf{x}$$

- Monte Carlo integration (sampling approximation)
  - Sample a set of i.i.d. points $\{\mathbf{x}_1,\ldots,\mathbf{x}_M\}$ from p(**x**)
  - Approximate integral with sample average

$$error_{true}(\mathbf{w}) \approx \frac{1}{M}\sum_{j=1}^{M} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

Why training set error doesn't approximate prediction error?

- Sampling approximation of prediction error:

$$error_{true}(\mathbf{w}) \approx \frac{1}{M}\sum_{j=1}^{M} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

- Training error :

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

- Very similar equations!!!
  - Why is training set a bad measure of prediction error???

Why training set error doesn't approximate prediction error?

Because you cheated!!!

Training error good estimate for a single **w,**
But you optimized **w** with respect to the training error, and found **w** that is good for this set of samples

Training error is a (optimistically) biased estimate of prediction error

- Very similar equations!!!
  - Why is training set a bad measure of prediction error???

Test set error

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

- Given a dataset, **randomly** split it into two parts:
  - Training data – {$\mathbf{x}_1, \ldots, \mathbf{x}_{Ntrain}$}
  - Test data – {$\mathbf{x}_1, \ldots, \mathbf{x}_{Ntest}$}
- Use training data to optimize parameters **w**
- **Test set error:** For the ***final output*** $\hat{\mathbf{w}}$, evaluate the error using:

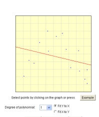
$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

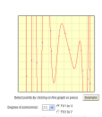
Test set error as a function of model complexity

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \right)^2 p(\mathbf{x}) d\mathbf{x}$$

$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$





Overfitting

- **Overfitting:** a learning algorithm overfits the training data if it outputs a solution **w** when there exists another solution **w'** such that:

$$[error_{train}(\mathbf{w}) < error_{train}(\mathbf{w}')] \wedge [error_{true}(\mathbf{w}') < error_{true}(\mathbf{w})]$$

How many points to I use for training/testing?

- Very hard question to answer!
  - Too few training points, learned **w** is bad
  - Too few test points, you never know if you reached a good solution
- Bounds, such as Hoeffding's inequality can help:

$$P(|\,\hat{\theta} - \theta^* \,| \geq \epsilon) \;\leq\; 2e^{-2N\epsilon^2}$$

- More on this later this quarter, but still hard to answer
- Typically:
  - If you have a reasonable amount of data, pick test set "large enough" for a "reasonable" estimate of error, and use the rest for learning
  - If you have little data, then you need to pull out the big guns…
    - e.g., bootstrapping

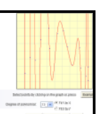
Error estimators

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2 p(\mathbf{x})d\mathbf{x}$$

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

Error as a function of number of training examples for a fixed model complexity



$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

little data → infinite data

Error estimators

Be careful!!!

Test set only unbiased if you never never ever ever do any any any any learning on the test data

For example, if you use the test set to select the degree of the polynomial… no longer unbiased!!! (We will address this problem later in the quarter)

$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

What you need to know

- True error, training error, test error
  - Never learn on the test data
  - Never learn on the test data
  - Never learn on the test data
  - Never learn on the test data
  - Never learn on the test data

- Overfitting

Bayesian Methods

Machine Learning – CSE546
Sham Kakade
University of Washington

Oct 4, 2016

What about prior

- Billionaire says: Wait, I know that the thumbtack is "close" to 50-50. What can you do for me now?
- **You say: I can learn it the Bayesian way…**
- Rather than estimating a single θ, we obtain a distribution over possible values of θ

Bayesian Learning

- Use Bayes rule:

$$P(\theta \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \theta) P(\theta)}{P(\mathcal{D})}$$

- Or equivalently:

$$P(\theta \mid \mathcal{D}) \propto P(\mathcal{D} \mid \theta) P(\theta)$$

Bayesian Learning for Thumbtack

$$P(\theta \mid \mathcal{D}) \propto P(\mathcal{D} \mid \theta) P(\theta)$$

- Likelihood function is simply Binomial:

$$P(\mathcal{D} \mid \theta) = \theta^{\alpha_H}(1-\theta)^{\alpha_T}$$

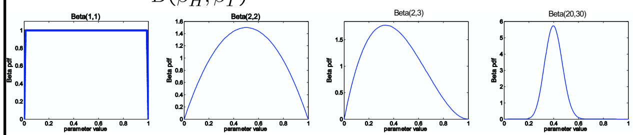
- What about prior?
  - Represent expert knowledge
  - Simple posterior form
- Conjugate priors:
  - Closed-form representation of posterior
  - **For Binomial, conjugate prior is Beta distribution**

Beta prior distribution – P(θ)

$$P(\theta) = \frac{\theta^{\beta_H-1}(1-\theta)^{\beta_T-1}}{B(\beta_H,\beta_T)} \sim Beta(\beta_H,\beta_T)$$

Mean:

Mode:

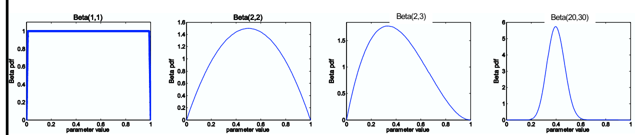


- Likelihood function: $P(\mathcal{D} \mid \theta) = \theta^{\alpha_H}(1-\theta)^{\alpha_T}$
- Posterior: $P(\theta \mid \mathcal{D}) \;\propto\; P(\mathcal{D} \mid \theta)P(\theta)$

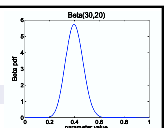
Posterior distribution

- Prior: $Beta(\beta_H,\beta_T)$
- Data: α_H heads and α_T tails
- Posterior distribution:

$$P(\theta \mid \mathcal{D}) \sim Beta(\beta_H+\alpha_H, \beta_T+\alpha_T)$$



Using Bayesian posterior



- Posterior distribution:

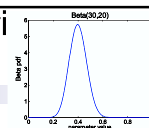
$$P(\theta \mid \mathcal{D}) \sim Beta(\beta_H+\alpha_H, \beta_T+\alpha_T)$$

- Bayesian inference:
  - No longer single parameter:

$$E[f(\theta)] = \int_0^1 f(\theta)P(\theta \mid \mathcal{D})d\theta$$

  - Integral is often hard to compute

MAP: Maximum a posteriori approximation



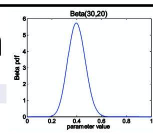
$$P(\theta \mid \mathcal{D}) \sim Beta(\beta_H+\alpha_H, \beta_T+\alpha_T)$$

$$E[f(\theta)] = \int_0^1 f(\theta)P(\theta \mid \mathcal{D})d\theta$$

- As more data is observed, Beta is more certain
- MAP: use most likely parameter:

$$\widehat{\theta} = \arg\max_\theta P(\theta \mid \mathcal{D}) \qquad E[f(\theta)] \approx f(\widehat{\theta})$$

MAP for Beta distribution



$$P(\theta \mid \mathcal{D}) = \frac{\theta^{\beta_H+\alpha_H-1}(1-\theta)^{\beta_T+\alpha_T-1}}{B(\beta_H+\alpha_H, \beta_T+\alpha_T)} \sim Beta(\beta_H{+}\alpha_H, \beta_T{+}\alpha_T)$$

- MAP: use most likely parameter:

$$\widehat{\theta} = \arg\max_{\theta} P(\theta \mid \mathcal{D}) =$$

- Beta prior equivalent to extra thumbtack flips
- As $N \rightarrow 1$, prior is "forgotten"
- **But, for small sample size, prior is important!**