# Lecture 3: Updating SEPs

(supervised learning vs. posterior updating)

Instructor: **Aaron Schein**

**Modern Methods in Applied Statistics**

STAT 34800 (Spring 2023)

THE UNIVERSITY OF CHICAGO

# Recap

- USS *Scorpion* is lost in the Atlantic
- "Search effectiveness probability" (SEP)

$$q = P(\tilde{Z} = 1 \mid Z = 1, a = 1)$$

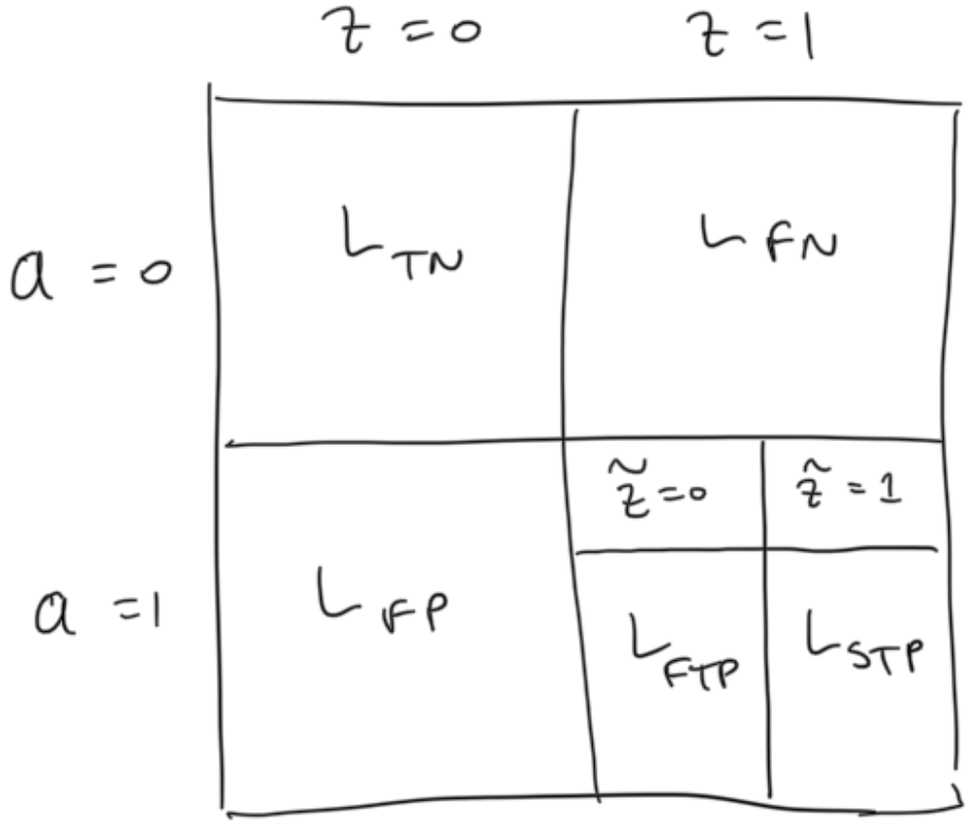
- $a$ = whether we search the cell
- $Z$ = whether it is in the cell
- $\tilde{Z}$ = whether we actually find it
- $\pi^*$ = posterior over $Z$

- Choose action that minimizes *Bayesian expected loss*

$$\rho(\pi^*, a) = E_{Z \sim \pi^*}[\ell(Z, a)]$$

$$\rho(\pi^*, 1) = \pi^* L_{TP} + (1 - \pi^*) L_{FP}$$

where $L_{TP} = q L_{STP} + (1 - q) L_{FTP}$



# Binomial trials

- SEPs play pivotal role in our decision
- In the real search, they tried estimating SEPs with **trials**
- e.g., drop $N$ objects, see how many ($b$) divers can find

$$b \sim \text{Binom}(N, q)$$

- PMF is proportional (in $q$) to

$$P(b \mid N, q) \propto_q q^b (1-q)^{N-b}$$

- Sum this over # of all $N$-sequences with $b$ successes

$$P(b \mid N, q) = \frac{N!}{b!(N-b)!} q^b (1-q)^{N-b}$$

often aliased as ***N*-choose-*b*** $\binom{N}{b}$

# Binomial trials

- How should we update $q$ based on trials?
- Maximum likelihood estimate (MLE) is

$$\hat{q}^{MLE} = \text{argmax}_q \; q^b(1-q)^{N-b}$$

$$= \frac{b}{N}$$

- ``Black swan effect'' (e.g., $N = 2,\ b = 0 \rightarrow \hat{q}^{MLE} = 0$)

# Binomial trials

- Recall there are *K* cells
- Want to search cell with lowest expected loss
- Involves *K* different SEPs

$$q_k = P(\tilde{Z} = k \mid Z = k,\, a = k)$$

- We can run trials in ***only some*** cells; for most $N_k = 0$

$$b_k \sim \text{Binom}(N_k, q_k)$$

- Desideratum: learn structure that **generalizes**
- i.e., estimating $q_k$ with trial should tell us about $q_{k'}$

# Outline

- Part I: Supervised learning
- Part II: Bayesian posterior updating

# Outline

- Part I: Supervised learning
- Part II: Bayesian posterior updating

# **Supervised learning**: Basic setup

- Each cell *i* has **covariates** $\boldsymbol{x}_i \in \mathbb{R}^p$
- e.g., $x_{i1} \in \{0,1\}$ is *depth* (shallow vs deep)
- e.g., $x_{i2} \in \{0,1\}$ is *temperature* (warm vs cold)
- Share information across cells by assuming SEPs are

$$q_i = q(\boldsymbol{x}_i)$$

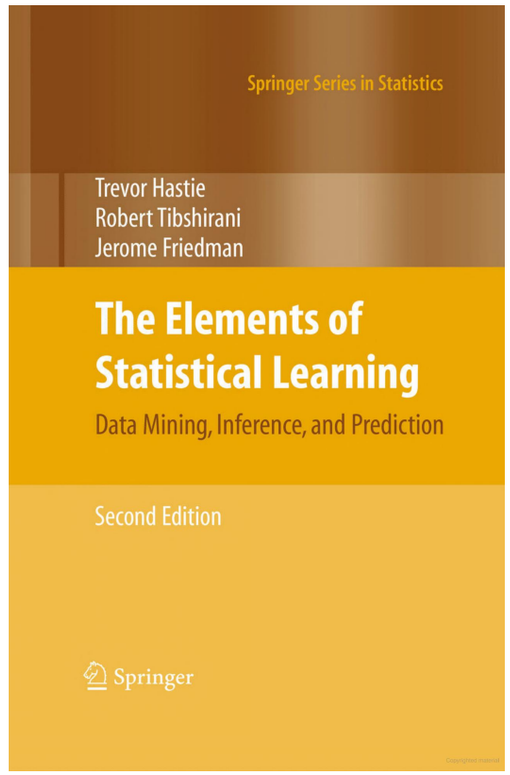
- Learn $\hat{q}(\,\cdot\,)$ using the few cells where we did do trials
- Then evaluate $\hat{q}(\,\cdot\,)$ on cells where we didn't do trials

# **Supervised learning**: Basic setup

- Each cell *i* has **covariates** $\boldsymbol{x}_i \in \mathbb{R}^p$
- e.g., $x_{i1} \in \{0,1\}$ is *depth* (shallow vs deep)
- e.g., $x_{i2} \in \{0,1\}$ is *temperature* (warm vs cold)

- **Disclaimer:** We are going temporarily depart from some of our notation and setup to match Chapter 1 of ESL

- It will all connect back though…



# **Supervised learning**: Basic setup

- Each cell *i* has **covariates** $\boldsymbol{x}_i \in \mathbb{R}^p$
- e.g., $x_{i1} \in \{0,1\}$ is *depth* (shallow vs deep)
- e.g., $x_{i2} \in \{0,1\}$ is *temperature* (warm vs cold)
- Binary **outcome** $y_i \in \{0,1\}$ (success vs failed)
- Want to learn decision rule $y_i \approx f(\boldsymbol{x}_i)$
- Run trials in $n$ cells to obtain **supervised data**

$$\mathscr{T} = (\boldsymbol{x}_i, y_i)_{i=1}^n$$

- This is our **training set**; use it to learn $\hat{f}$
- Then we evaluate on **test data** $\hat{y} = \hat{f}(\boldsymbol{x})$

# **Supervised learning**: Example (*k*NNs)

- Define the ***k*-nearest neighbors (kNNs)** decision rule:

$$f(\boldsymbol{x}) = \frac{1}{k} \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} y_i$$

- $N_k(\boldsymbol{x})$ = the $k$ nearest $\boldsymbol{x}_i$ in the training data (`neighbors") to $\boldsymbol{x}$

- For any point $\boldsymbol{x}$, average the outcomes $y_i$ of the nearest neighbors in the training set

# **Supervised learning**: Example (*k*NNs)

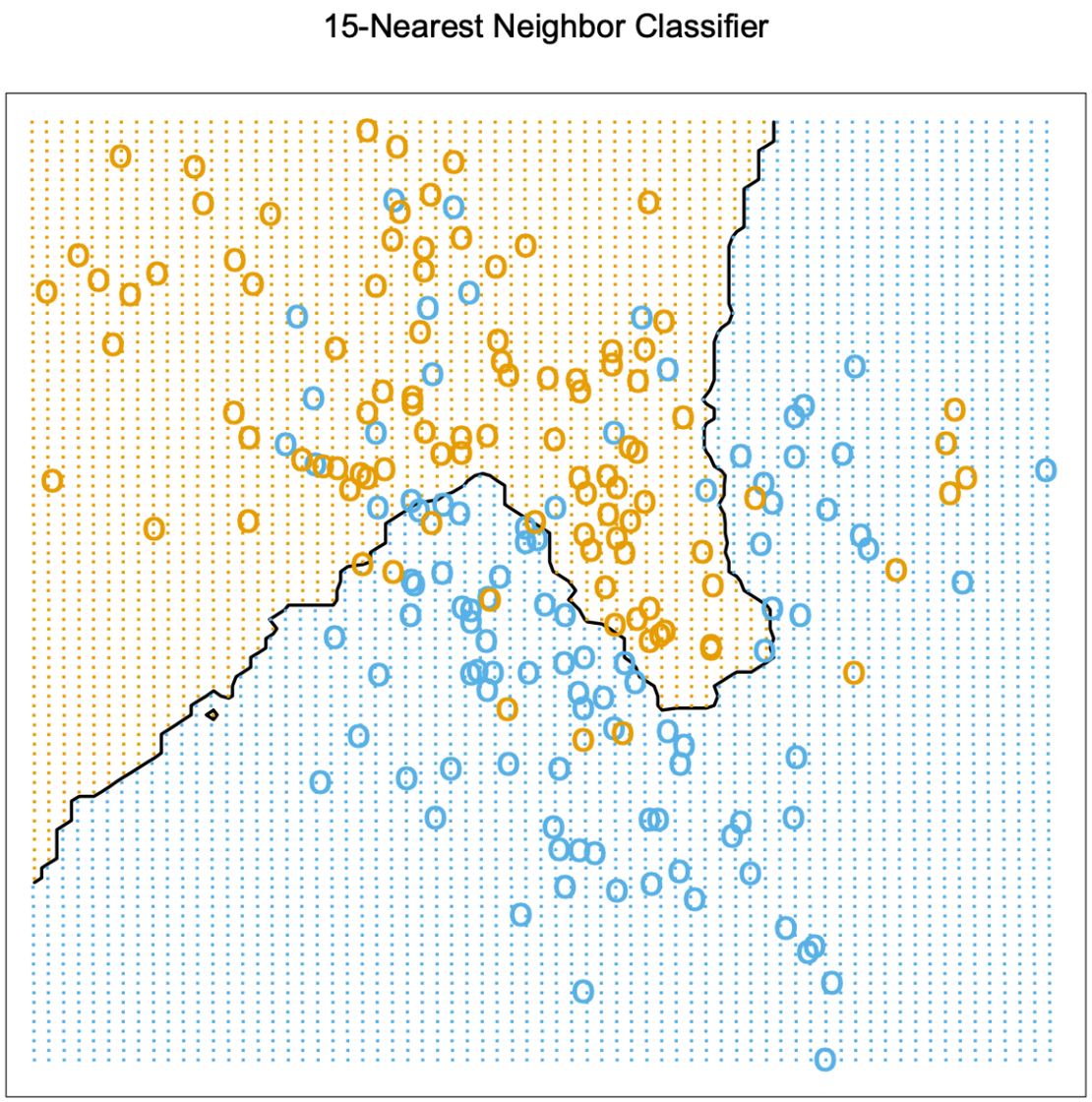
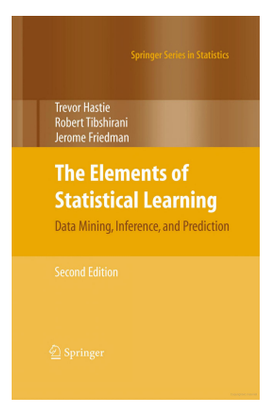



**FIGURE 2.2.** *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (BLUE $= 0$, ORANGE $= 1$) *and then fit by* 15*-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the* 15*-nearest neighbors.*



# **Supervised learning**: Risk

- Define optimal rule to be one that minimizes **risk**

$$\hat{f}^{opt} = \text{argmin}_{f(\cdot)} \, E_{(\boldsymbol{X},Y)\sim P(\boldsymbol{X},Y)}[\ell(Y, f(\boldsymbol{X}))]$$

- Optimal rule minimizes **conditional** expectation

$$\hat{f}^{opt}(\boldsymbol{x}) = \text{argmin}_{a} \, E_{Y\sim P(Y|\boldsymbol{X}=\boldsymbol{x})}[\ell(Y, a) \mid \boldsymbol{X} = \boldsymbol{x}]$$

- Depends on loss function. For **squared loss**:

$$\begin{aligned}\hat{f}^{opt}(\boldsymbol{x}) &= \text{argmin}_{a} \, E_{Y\sim P(Y|\boldsymbol{X}=\boldsymbol{x})}[(Y-a)^2 \mid \boldsymbol{X} = \boldsymbol{x}] \\ &= E[Y \mid \boldsymbol{X} = \boldsymbol{x}]\end{aligned}$$

# **Supervised learning**: Risk

- Optimal rule minimizes **conditional** expectation

$$\hat{f}^{opt}(\boldsymbol{x}) = \text{argmin}_a \; E_{Y \sim P(Y|X=\boldsymbol{x})}[\ell(Y, a) \mid \boldsymbol{X} = \boldsymbol{x}]$$

- Generally, optimal rule will be **some functional** $g(\,\cdot\,)$ of the conditional distribution:

$$\hat{f}^{opt}(\boldsymbol{x}) = g\Big(P(Y \mid \boldsymbol{X} = \boldsymbol{x})\Big)$$

- This suggests two distinct approaches:
  - **generative**: model $P(Y \mid \boldsymbol{X} = \boldsymbol{x})$, then apply $g(\,\cdot\,)$
  - **discriminative**: learn $f(\,\cdot\,)$ directly

# **Supervised learning**: Generative approach

- The **naive Bayes classifier** is **generative**
- It infers $P(Y \mid \boldsymbol{x})$ by **modeling** $P(\boldsymbol{x})$ and $P(\boldsymbol{x} \mid Y = y)$
- **Main assumption:** covariates are mutually **independent**

$$P(\boldsymbol{x} \mid Y = y) = \prod_{j=1}^{p} P(x_j \mid Y = y) \qquad P(\boldsymbol{x}) = \prod_{j=1}^{p} P(x_j)$$

$$P(x_1 = \text{shallow} \mid y = \text{success}) = 0.6 \qquad P(x_1 = \text{shallow}) = 0.1$$

$$P(x_1 = \text{shallow} \mid y = \text{ failure }) = 0.1$$

$$P(x_2 = \text{cold} \mid y = \text{ failure }) = 0.8$$

# **Supervised learning**: Generative approach

- The **naive Bayes classifier** is **generative**
- It infers $P(Y \mid \boldsymbol{x})$ by **modeling** $P(\boldsymbol{x})$ and $P(\boldsymbol{x} \mid Y = y)$
- **Main assumption:** covariates are mutually **independent**

$$P(\boldsymbol{x} \mid Y = y) = \prod_{j=1}^{p} P(x_j \mid Y = y) \qquad P(\boldsymbol{x}) = \prod_{j=1}^{p} P(x_j)$$

- By Bayes rule, we can then calculate **tractably**:

$$P(Y = y \mid \boldsymbol{x}) = \frac{\prod_{j=1}^{p} P(x_j \mid Y = y)\, P(x_j)}{\sum_{y'} \prod_{j=1}^{p} P(x_j \mid Y = y')\, P(x_j)}$$

# **Supervised learning**: Discriminative approach

- The generative approach relies on modeling assumptions to make full posterior inference tractable

- **Discriminative** approach:
  - We *don't need* the full posterior $P(Y \mid \boldsymbol{X} = \boldsymbol{x})$
  - We *just need* a functional of it

$$\hat{f}^{opt}(\boldsymbol{x}) = g\Big(P(Y \mid \boldsymbol{X} = \boldsymbol{x})\Big)$$

  - So, estimate it directly!
  - Example: the **regression function**

$$f(\boldsymbol{x}) = E[Y \mid \boldsymbol{X} = \boldsymbol{x}]$$

# **Supervised learning**: Discriminative approach

- Discriminative approaches **also** rely on modeling assumptions
- Instead of $P(\boldsymbol{X}, Y)$, assumptions are about $f$
- e.g., **linear regression** assumes that

$$f(\boldsymbol{x}) = E[Y \mid \boldsymbol{X} = \boldsymbol{x}] \approx \beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}$$

- e.g., ***k*-nearest neighbors** assumes that

$$f(\boldsymbol{x}) = E[Y \mid \boldsymbol{X} = \boldsymbol{x}] \approx \frac{1}{k} \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} y_i$$

- These assumptions are about the **complexity** of $f$

# **Supervised learning**: Example (*k*NNs)

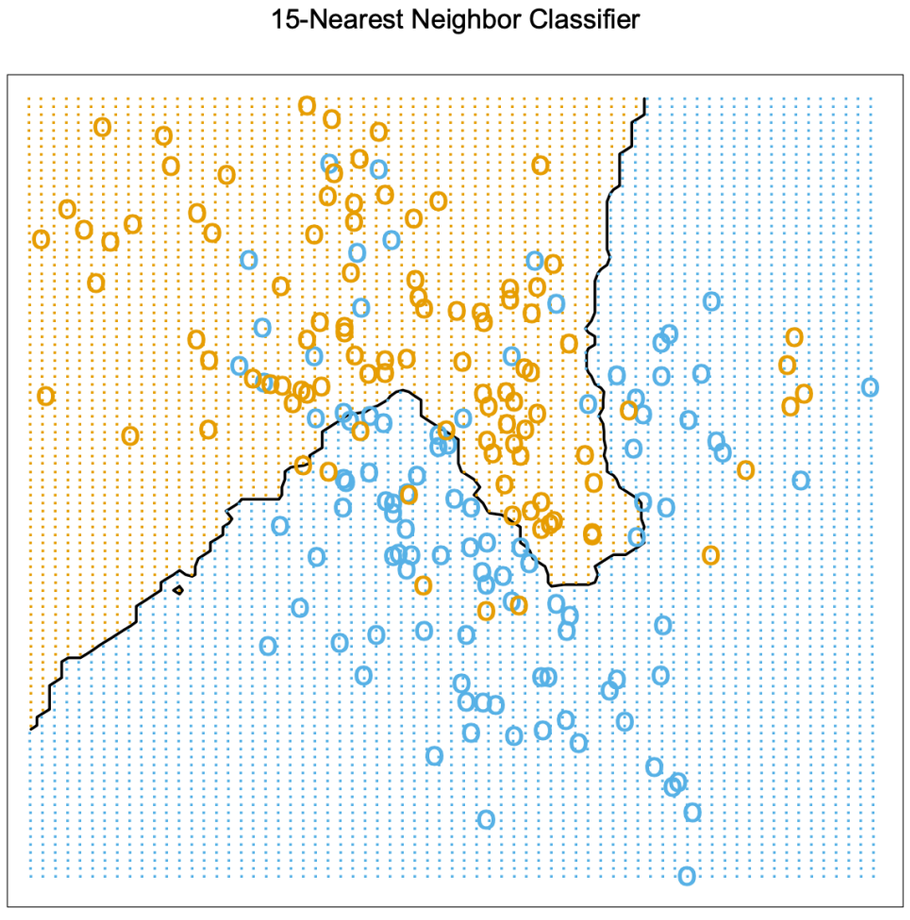



**FIGURE 2.2.** *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (BLUE $= 0$, ORANGE $= 1$) *and then fit by* 15*-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the* 15*-nearest neighbors.*

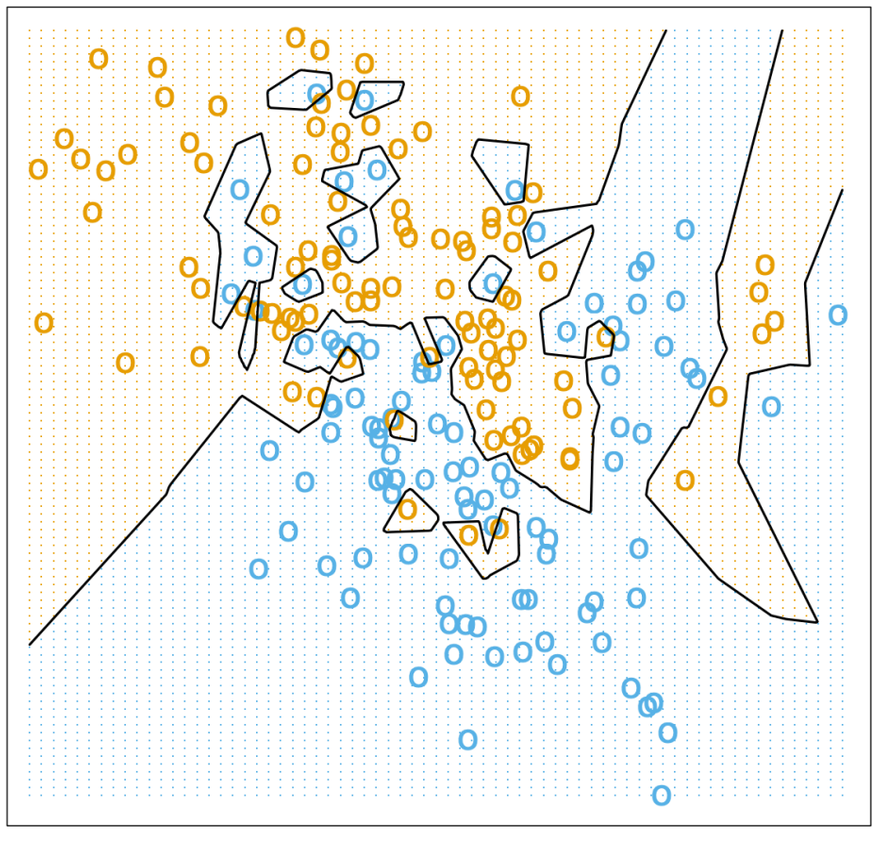
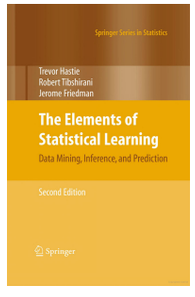



**FIGURE 2.3.** *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (BLUE $= 0$, ORANGE $= 1$)*, and then predicted by* 1*-nearest-neighbor classification.*

- Effective number of parameters $p \approx N/k$



# **Supervised learning**: Overfitting

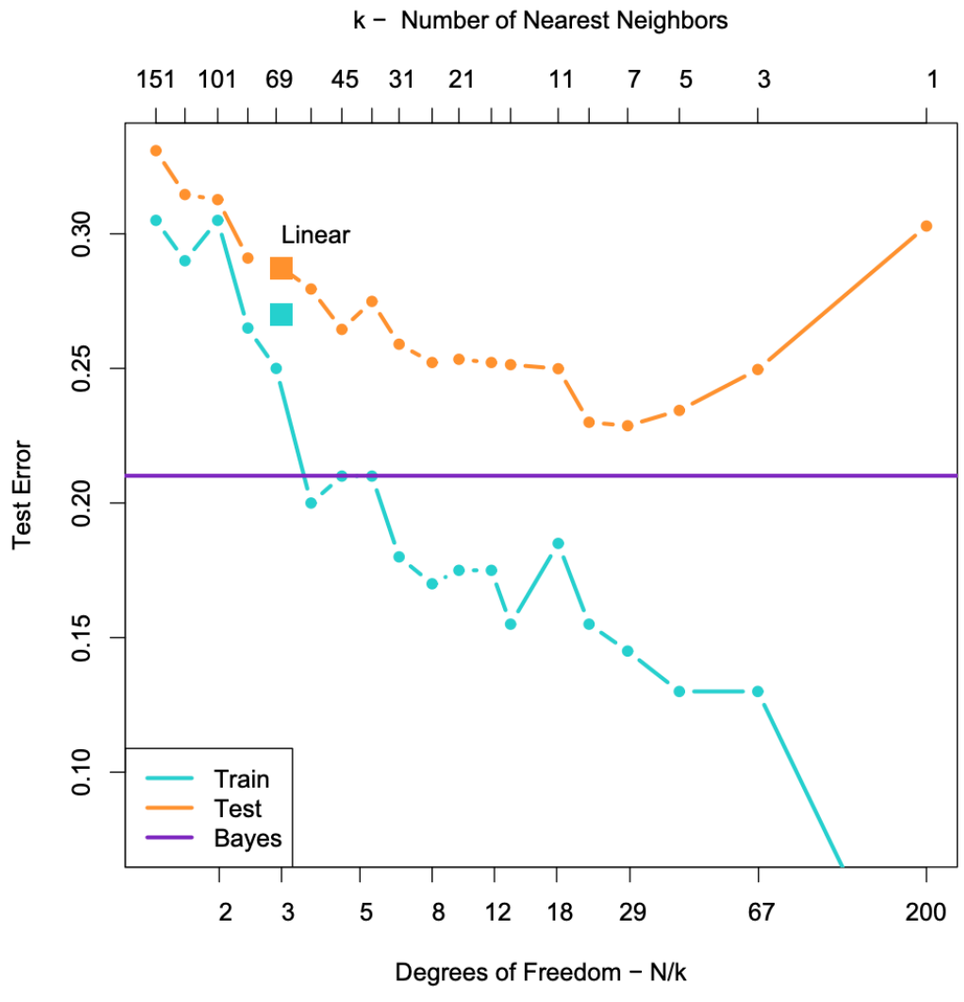
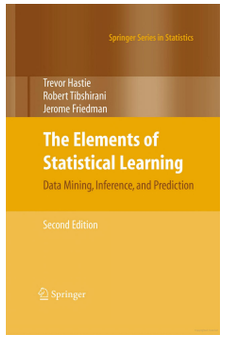



**FIGURE 2.4.** *Misclassification curves for the simulation example used in Figures 2.1, 2.2 and 2.3. A single training sample of size* 200 *was used, and a test sample of size* 10,000*. The orange curves are test and the blue are training error for k-nearest-neighbor classification. The results for linear regression are the bigger orange and blue squares at three degrees of freedom. The purple line is the optimal Bayes error rate.*

- As the complexity of $f$ increases, it begins to **memorize** training data
- This threatens its ability to **generalize**
- **Overfitting**: train loss ↓, test loss ↑



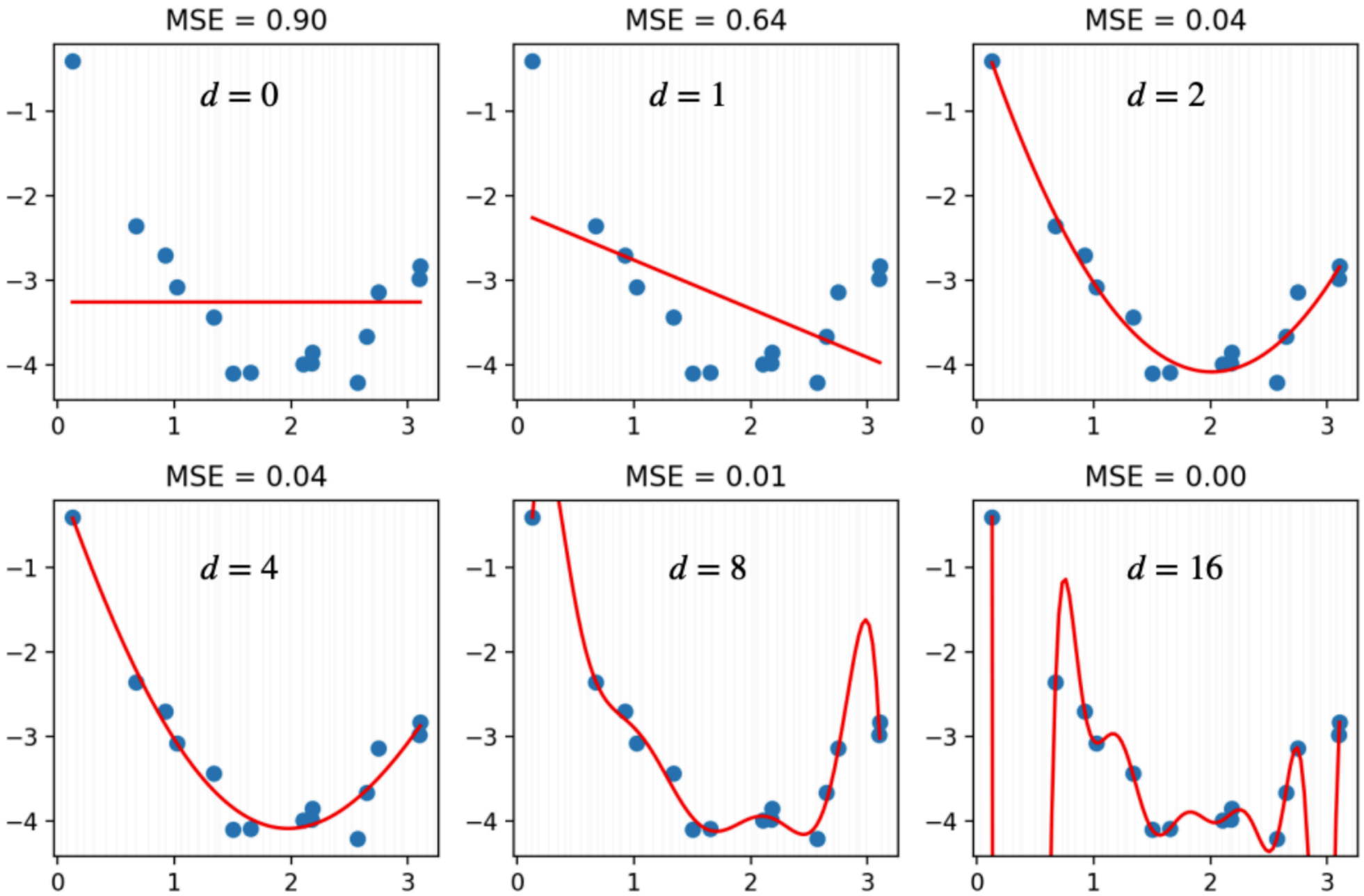
# **Supervised learning**: Overfitting

- As another example: **linear regression**
- Introduce **linear basis expansions**

$$f(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{p^*} h_j(\boldsymbol{x})\beta_j$$

- e.g., polynomial interactions $h_j(\boldsymbol{x}) = x_1^2 x_2^3$
- Still linear, but **number of parameters** is bigger ($p^* \gg p$)
- $f$ can fit complex functions (of original covariates)
- Risk of overfitting…

# **Supervised learning**: Overfitting



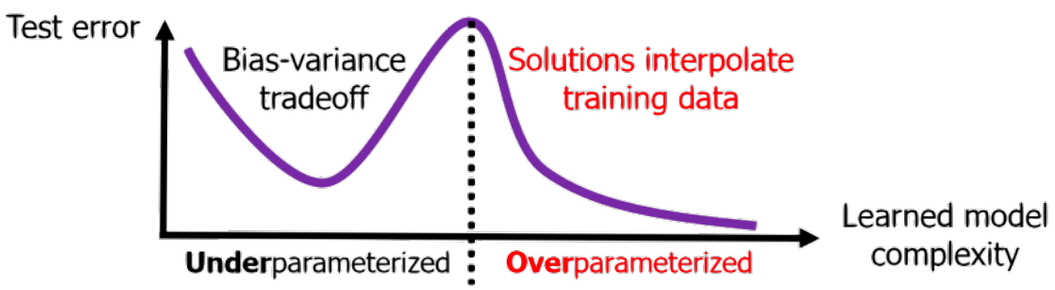

https://www.davidinouye.com/course/ece57000-fall-2021/lectures/loss-functions-and-regularization.pdf

# **Supervised learning**: Overfitting

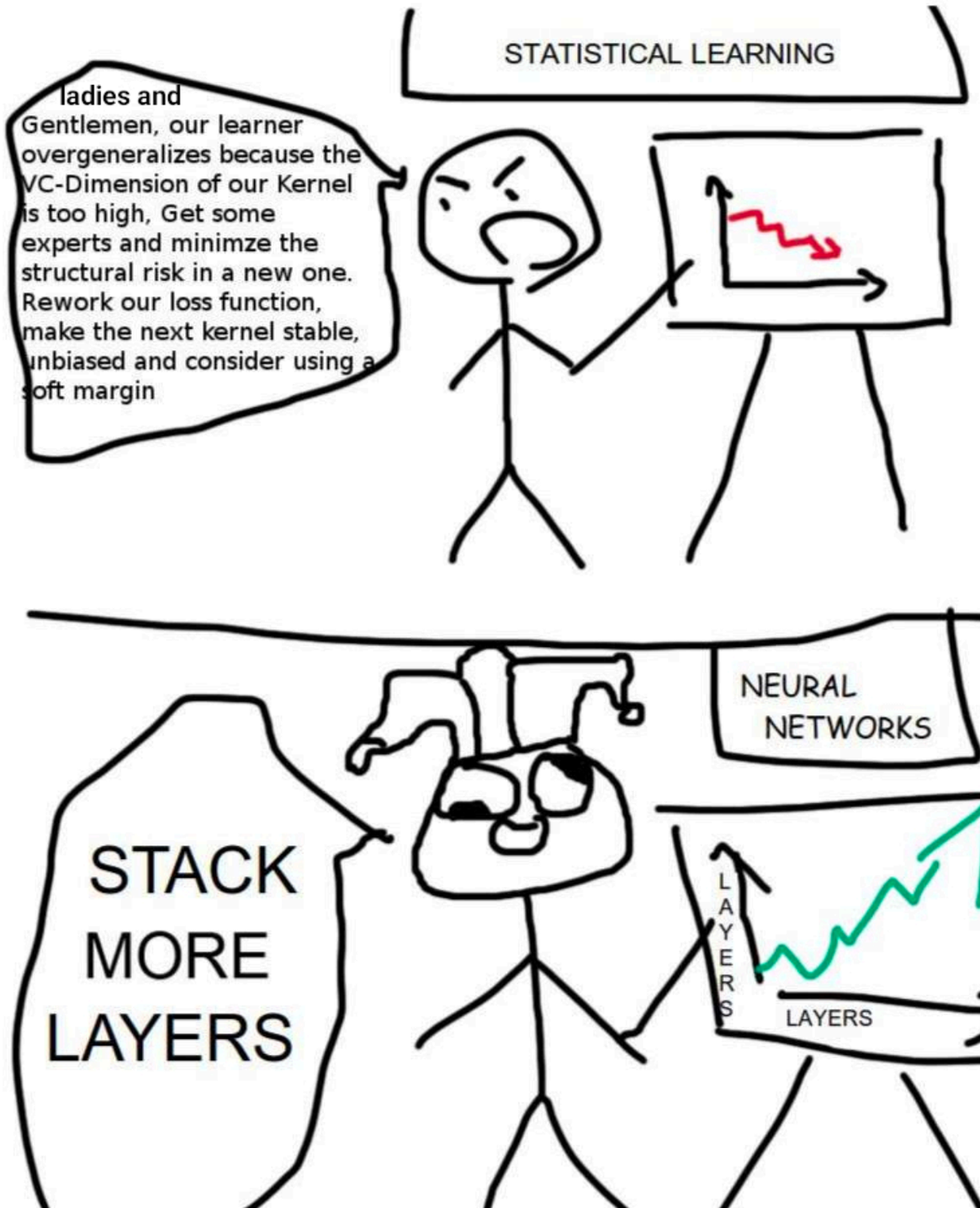
- **Caveat**: everything I've said is true for data sets and models that aren't **BIG**…

- When training sets and models get **BIG**, things get more complicated…

- See recent papers on ``**double descent"**




https://arxiv.org/pdf/2109.02355.pdf



# **Supervised learning**: Regularization

- One way to constrain $f$ is to limit the number of parameters
- Another way is to **restrict values** of the parameters
- Example: linear regression with basis expansion ($p^*$ large)

$$f_\beta(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{p^*} h_j(\boldsymbol{x})\beta_j$$

- Minimize **penalized** or **regularized** loss function
- $R(\,\cdot\,)$ is the regularizer and $\lambda$ is the strength of regularization

$$\hat{\boldsymbol{\beta}} = \text{argmin} \left[ \sum_{i=1}^{n} \ell(y_i, f_\beta(\boldsymbol{x}_i)) + \lambda R(\boldsymbol{\beta}) \right]$$

# **Supervised learning**: Regularization

- One way to constrain $f$ is to limit the number of parameters
- Another way is to **restrict values** of the parameters
- Example: linear regression with basis expansion ($p^*$ large)

$$f_\beta(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{p^*} h_j(\boldsymbol{x})\beta_j$$

- Minimize **penalized** or **regularized** loss function
- e.g., **L2** or **ridge penalty** (encourages small squared values)

$$\hat{\boldsymbol{\beta}} = \text{argmin} \left[ \sum_{i=1}^{n} \ell(y_i, f_\beta(\boldsymbol{x}_i)) + \lambda \sum_{j=1}^{p^*} (\beta_j)^2 \right]$$

# **Supervised learning**: Regularization

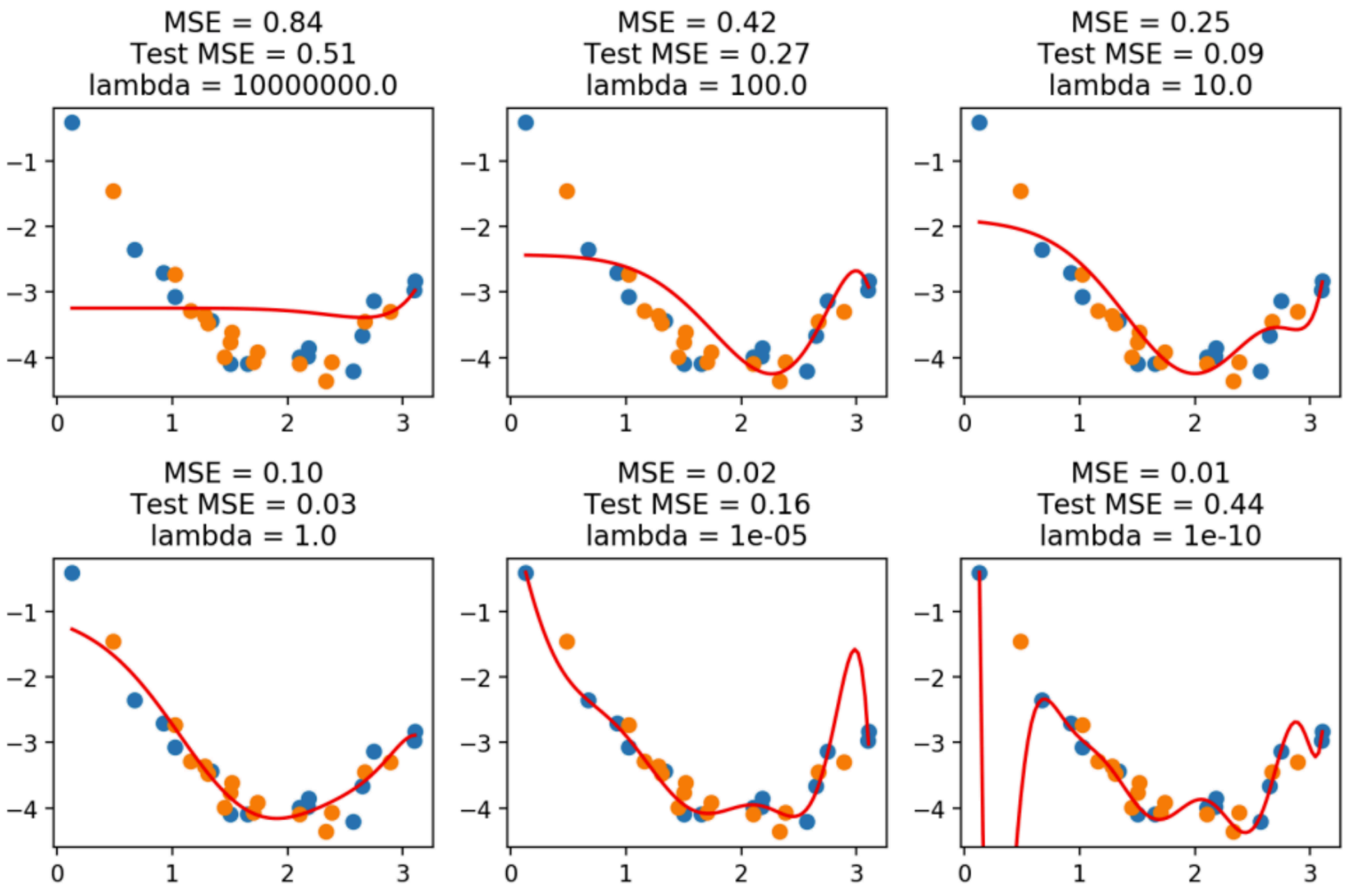
- One way to constrain $f$ is to limit the number of parameters
- Another way is to **restrict values** of the parameters
- Example: linear regression with basis expansion ($p^*$ large)

$$f_\beta(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{p^*} h_j(\boldsymbol{x})\beta_j$$

- Minimize **penalized** or **regularized** loss function
- e.g., **L1** or **LASSO penalty** (encourages small absolute values)

$$\hat{\boldsymbol{\beta}} = \text{argmin} \left[ \sum_{i=1}^{n} \ell(y_i, f_\beta(\boldsymbol{x}_i)) + \lambda \sum_{j=1}^{p^*} |\beta_j| \right]$$

# **Supervised learning**: Regularizers as priors




https://www.davidinouye.com/course/ece57000-fall-2021/lectures/loss-functions-and-regularization.pdf

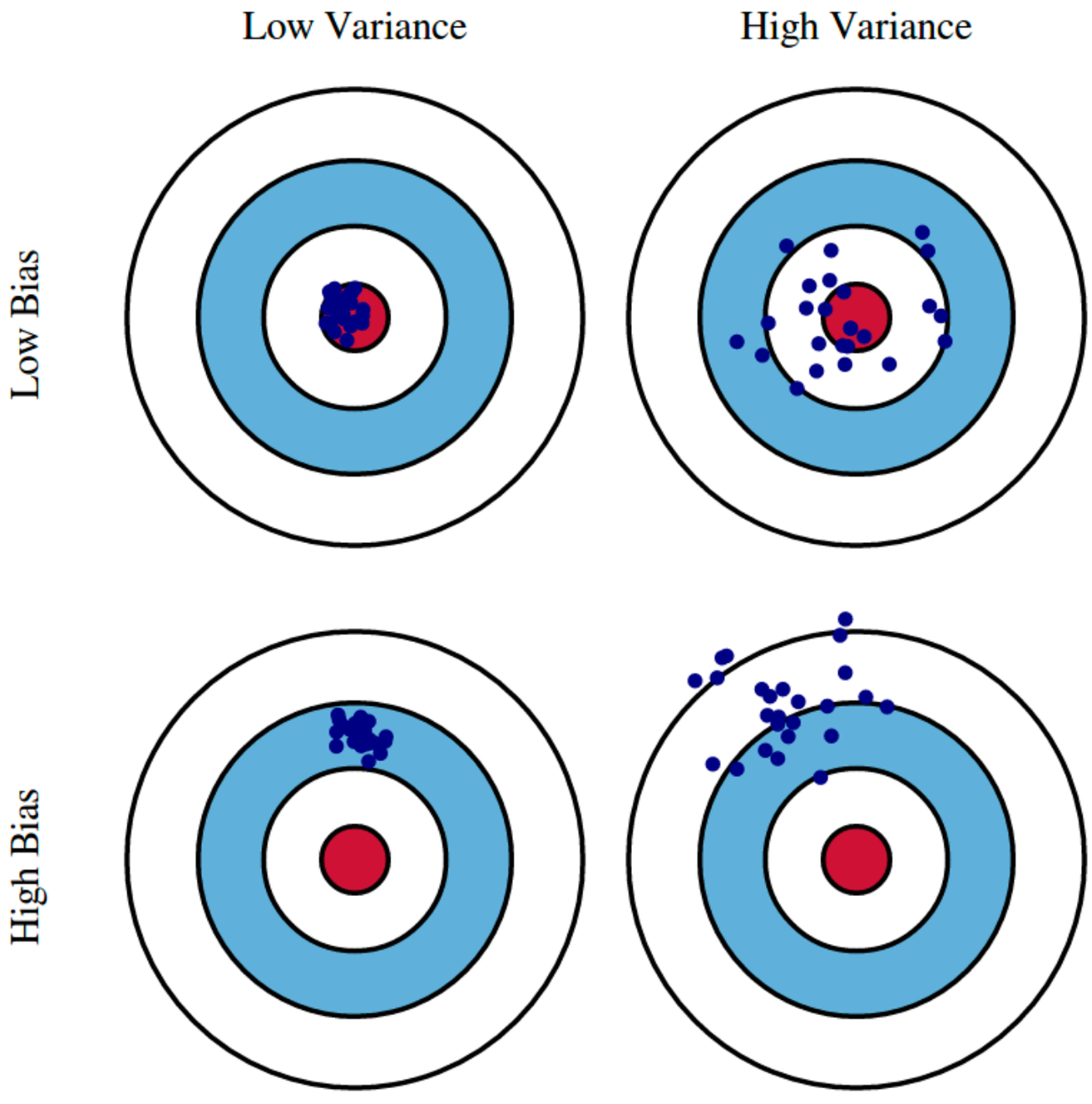
# **Supervised learning**: Regularizers as inductive biases

- Imagine **true** function $f$ vs **estimated** function $\hat{f}$
- Consider (squared) prediction error at a given point $x$
- How does $\hat{f}$ do in expectation over training sets $\mathcal{T}$?

$$\begin{aligned}\text{Err}(x) &= E_{\mathcal{T}}[(\hat{f}(x) - f(x))^2] \\ &= \text{Var}_{\mathcal{T}}(\hat{f}(x) - f(x)) + E_{\mathcal{T}}[\hat{f}(x) - f(x)]^2 \\ &= \text{Var}_{\mathcal{T}}(\hat{f}(x)) + \text{Bias}(\hat{f}(x))^2\end{aligned}$$

- **Bias-variance decomposition**: we can **bias** our estimator to decrease its **variance** and decrease expected error

# **Supervised learning**: Regularizers as inductive biases




https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html

## **Supervised learning**: Regularizers as inductive biases

- Regularizers **bias** our estimator in a good way
- ``Inductive bias'': decrease variance, improve **generalization**

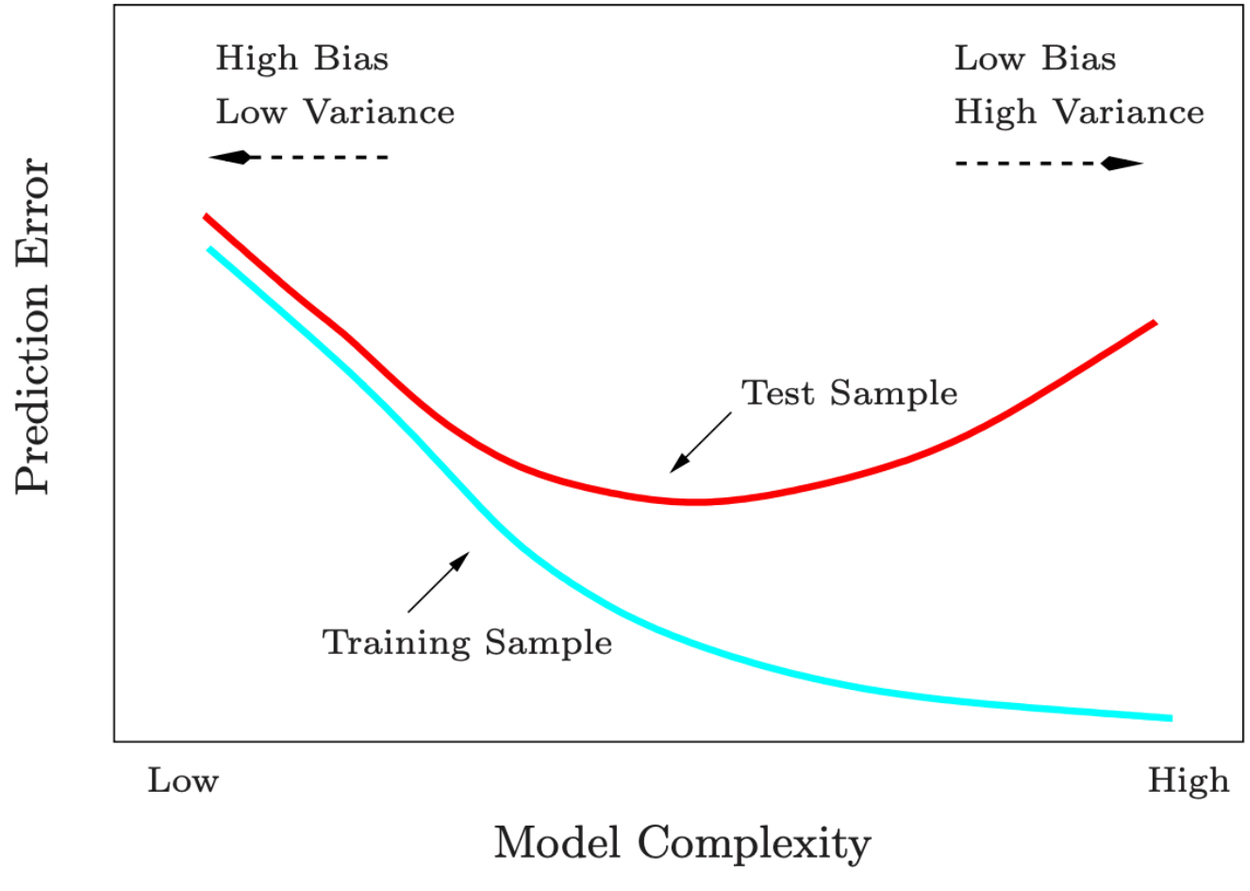
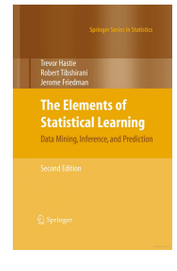



**FIGURE 2.11.** *Test and training error as a function of model complexity.*



# **Supervised learning**: Regularizers as priors

- Let's revisit minimizing squared error (no regularizer)

$$\hat{\boldsymbol{\beta}} = \text{argmin} \left[ \sum_{i=1}^{n} (y_i - f_\beta(\boldsymbol{x}_i))^2 \right]$$

  - Instead think **probabilistically**… assume $y_i$ are Gaussian

$$\log P(\boldsymbol{y} \mid \boldsymbol{x}) = \log \prod_{i=1}^{n} \mathcal{N}(y_i;\ f_\beta(\boldsymbol{x}_i), \sigma^2)$$

$$= \sum_{i=1}^{n} \log \left[ \frac{1}{\sigma\sqrt{2\pi}} \exp\left( -\frac{1}{2} \left( \frac{y_i - f_\beta(\boldsymbol{x})_i}{\sigma} \right)^2 \right) \right]$$

- The terms in this **likelihood** proportional to parameters are

$$\propto_\beta -\frac{1}{\sigma^2} \sum_{i=1}^{n} \left( y_i - f_\beta(\boldsymbol{x})_i \right)^2$$

- **Minimizing squared loss = maximizing Gaussian likelihood**

# **Supervised learning**: Regularizers as priors

- Now let's revisit **L2-regularized (``ridge'') regression**

$$\hat{\boldsymbol{\beta}} = \text{argmin}_{\beta} \left[ \sum_{i=1}^{n} (y_i - f_{\beta}(\boldsymbol{x}_i))^2 + \lambda \sum_{j=1}^{p^*} (\beta_j)^2 \right]$$

- Assume $\beta_j$ are independent zero-mean Gaussian…

$$\log P(\boldsymbol{\beta}) = \sum_{j=1}^{p^*} \log \mathcal{N}(\beta_i;\, 0, \tau^2) = \sum_{j=1}^{p^*} \log \left[ \frac{1}{\tau\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{\beta_j}{\tau}\right)^2\right) \right]$$

$$\propto_{\beta} -\frac{1}{\tau^2} \sum_{j=1}^{p^*} (\beta_j)^2$$

- **Minimizing L2 penalty = maximizing Gaussian prior**

# **Supervised learning**: Regularizers as priors

- Now let's revisit **L2-regularized (``ridge'') regression**

$$\hat{\boldsymbol{\beta}} = \text{argmin}_{\beta} \left[ \sum_{i=1}^{n} (y_i - f_{\beta}(\boldsymbol{x}_i))^2 + \lambda \sum_{j=1}^{p^*} (\beta_j)^2 \right]$$

- So, then maximizing posterior $P(\boldsymbol{\beta} \mid \boldsymbol{y}, \boldsymbol{x}) \propto P(\boldsymbol{\beta}, \boldsymbol{y} \mid \boldsymbol{x})$…

$$\hat{\boldsymbol{\beta}}^{MAP} = \text{argmax}_{\beta} \left[ -\frac{1}{\sigma^2} \sum_{i=1}^{n} \left( y_i - f_{\beta}(\boldsymbol{x})_i \right)^2 - \frac{1}{\tau^2} \sum_{j=1}^{p^*} (\beta_j)^2 \right]$$

- …is minimizing the **ridge regression** loss:

$$= \text{argmin}_{\beta} \left[ \sum_{i=1}^{n} \left( y_i - f_{\beta}(\boldsymbol{x})_i \right)^2 + \frac{\sigma^2}{\tau^2} \sum_{j=1}^{p^*} (\beta_j)^2 \right]$$

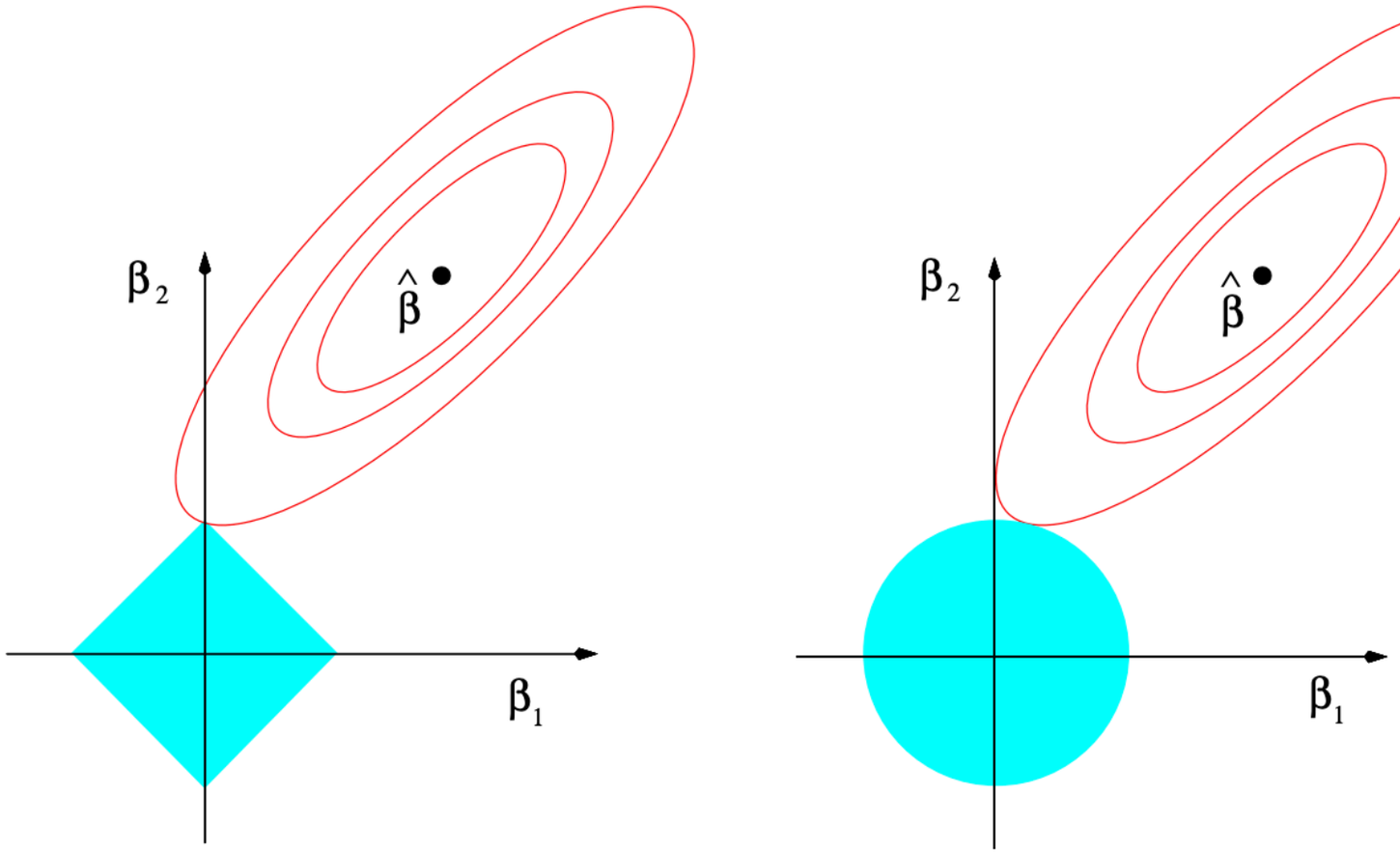
- where regularization determined by prior variance $\lambda = \frac{\sigma^2}{\tau^2}$

## **Supervised learning**: Regularizers as priors

- Regularizers are *morally* like **priors** on parameters
- Not all have strict probabilistic interpretations…
- …but the most common ones do: e.g.,
    - L2 / "ridge" = Gaussian priors
    - L1 / "LASSO" = Laplace priors
    - "Group LASSO" = spike-and-slab priors
- Generally, these are **shrinkage priors**

# **Supervised learning**: Regularizers as priors

- L1 selects **sparse** coefficients, L2 selects **shrunk** coefficients



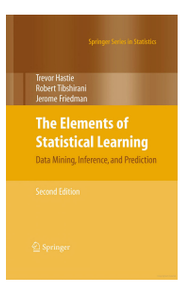




**FIGURE 3.11.** *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

# **Supervised learning**: Regularizers as priors

- L1 selects **sparse** coefficients, L2 selects **shrunk** coefficients

**The geometric view:**

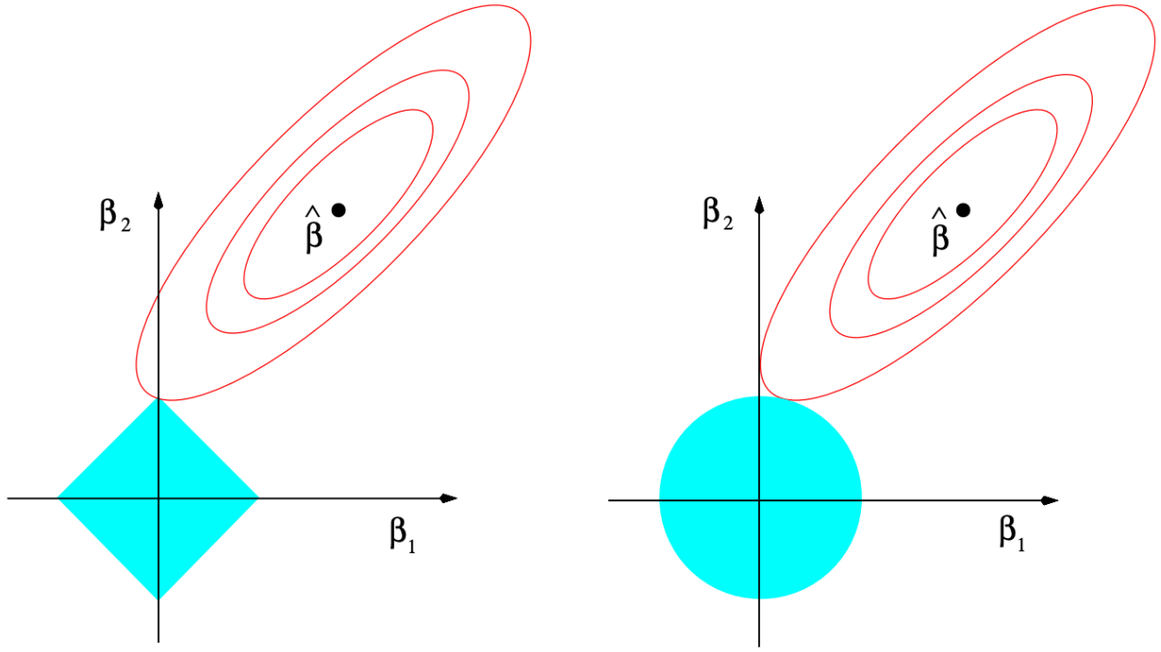
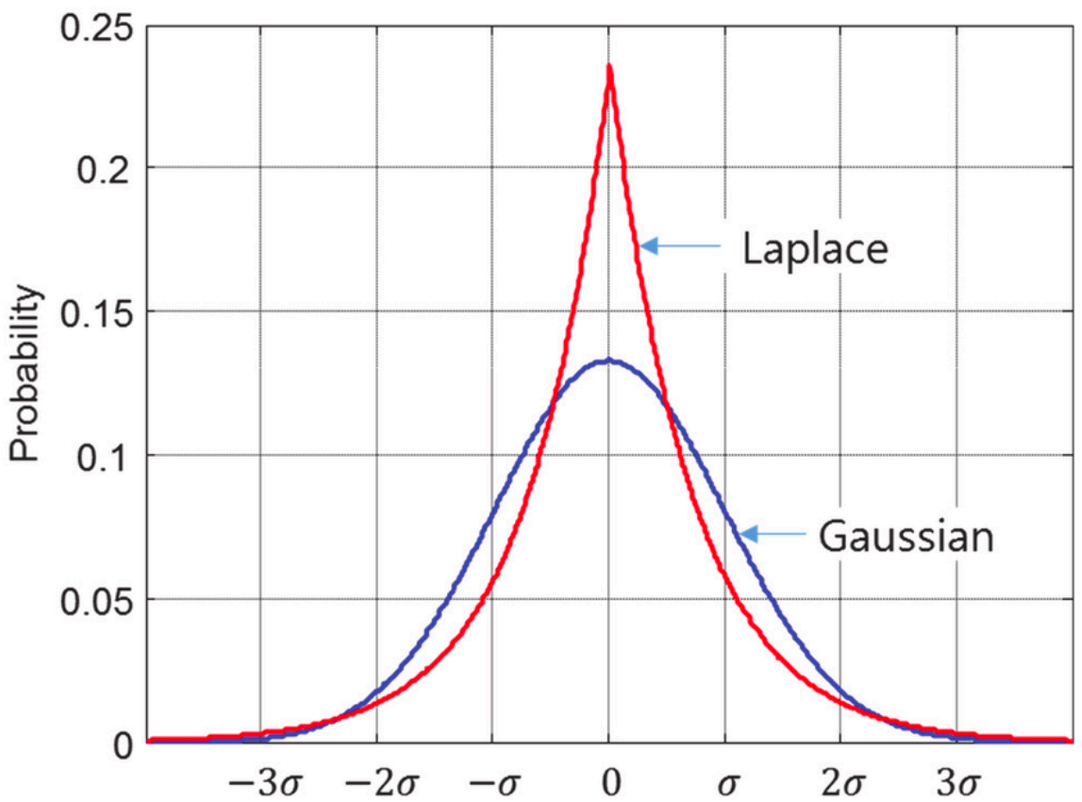



**FIGURE 3.11.** *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

**The probabilistic view:**




https://www.researchgate.net/figure/Gaussian-distribution-and-Laplace-distribution_fig7_321825093

# **Learning SEPs** with penalized logistic regression

- Let's map this all back to our setting…
- For cell *k* our SEP $q_k$ parameterizes a binomial likelihood

$$P(b_k \mid q_k, N_k) = \text{Binom}(b_k;\, N_k, q_k)$$

- Define $q_k = q_\beta(\boldsymbol{x}_k)$ to be a function of covariates

$$q_\beta(\boldsymbol{x}_k) = \frac{\exp(\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_k)}{1 + \exp(\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_k)}$$

- Equivalently, define the **log odds** or **logit** to be linear

$$\text{logit}(q_\beta(\boldsymbol{x}_k)) = \log \frac{q_\beta(\boldsymbol{x}_k)}{1 - q_\beta(\boldsymbol{x}_k)} = \beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_k$$

- This is (binomial) **logistic regression**

# **Learning SEPs** with penalized logistic regression

- Our loss function is the **negative log likelihood**

$$\ell(\boldsymbol{\beta}) = -\log \sum_{k=1}^{K} \text{Binom}\left(b_k;\, N_k, q_\beta(\boldsymbol{x}_k)\right)$$

- Could also introduce **linear basis expansions** (for a more complex function) with a **penalized** loss function (e.g., L1)

$$\ell(\boldsymbol{\beta}) = -\left[\log \sum_{k=1}^{K} \text{Binom}\left(b_k;\, N_k, q_\beta(\boldsymbol{x}_k)\right) - \lambda \sum_{j=1}^{p^*} |\beta_j|\right]$$

- Minimize the loss (maximize the likelihood)

$$\hat{\boldsymbol{\beta}} = \text{argmin}_\beta\, \ell(\boldsymbol{\beta})$$

- e.g., can use Newton-Raphson (loss is convex)

## **Learning SEPs** with penalized logistic regression

- We can then apply $\hat{\boldsymbol{\beta}}$ to cells $k'$ where we didn't do trials

$$\hat{q}_{k'} = q_{\hat{\boldsymbol{\beta}}}(\boldsymbol{x}_{k'}) = \frac{\exp(\hat{\beta}_0 + \hat{\boldsymbol{\beta}}^\top \boldsymbol{x}_{k'})}{1 + \exp(\hat{\beta}_0 + \hat{\boldsymbol{\beta}}^\top \boldsymbol{x}_{k'})}$$

- Voila! Data-informed SEPs for all search cells.

# **Supervised learning:** Takeaways

- Good idea if you have **good covariates**

  ``Features are king" -famous phrase (forgot who)

- If not, may need to more **complex functions** (e.g., basis expansions)
- For more complex functions we then risk **overfitting**
- We can combat this with **regularizers** ≈ **inductive biases** ≈ **priors**

**Uncertainty**

- The usual discriminative approaches return **point estimates** $\hat{q}_k$
- If we think error is high, we'd rather **quantify that uncertainty** $P(q_k)$
- We could then **propagate uncertainty** into our decision problem
- This leads us to the next section…

# Outline

- Part I: Supervised learning
- Part II: Bayesian posterior updating

# **Bayesian updating** of SEPs

- We can propagate uncertainty $P(q_k \mid -)$ into decision

- Define $\boldsymbol{q} = (q_1, \ldots, q_K)$ to be the full vector SEPs
- Re-define our **latent state** to be $\theta = (Z, \boldsymbol{q})$

- Our posterior is now $\pi^*(\theta) = P(q_k \mid -)\, P(Z \mid -)$
- Then the Bayesian expected loss is

$$\rho(\pi^*, a) = E_{\theta \sim \pi^*}[\ell(\theta, a)]$$

- The principle: treat $\boldsymbol{q}$ like a **latent variable** and integrate

# **Bayesian updating** of SEPs

- Like all latent variables, we begin with a prior

$$P(q_k)$$

- If we have a (binomial) trial in cell *k*, our posterior would be

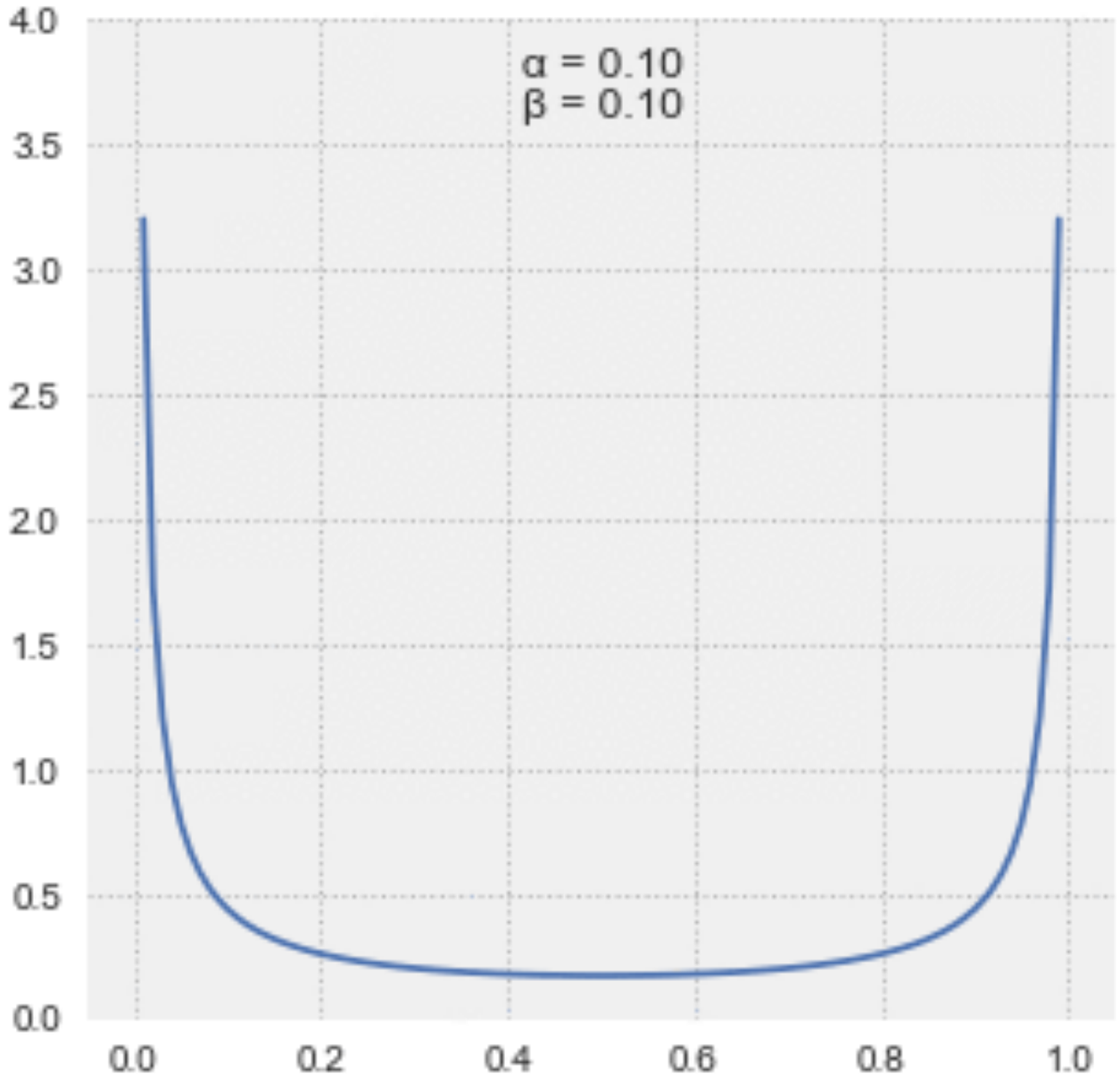
$$P(q_k \mid b_k, N_k)$$

- A convenient prior in this case is the **Beta distribution**

$$q_k \sim \text{Beta}(\alpha_0, \beta_0)$$

# **Bayesian updating:** Beta distribution

- The **Beta distribution** has support on (0,1)




https://en.wikipedia.org/wiki/Beta_distribution

# **Bayesian updating:** Beta distribution

- The PDF is proportional (in $q$) to

$$\text{Beta}(q;\, \alpha_0, \beta_0) \propto_q q^{\alpha_0 - 1}(1-q)^{\beta_0 - 1}$$

- This is called the **kernel** of the distribution; it will be useful

- The **normalizing constant** is then

$$\int_0^1 q^{\alpha_0 - 1}(1-q)^{\beta_0 - 1}\, dq = \frac{\Gamma(\alpha_0)\,\Gamma(\beta_0)}{\Gamma(\alpha_0 + \beta_0)}$$

often aliased as the **Beta function**: $\text{B}(\alpha_0, \beta_0)$

- The PDF is then equal to

$$\text{Beta}(q;\, \alpha_0, \beta_0) = \frac{\Gamma(\alpha_0 + \beta_0)}{\Gamma(\alpha_0)\Gamma(\beta_0)}\, q^{\alpha_0 - 1}(1-q)^{\beta_0 - 1}$$

# **Bayesian updating:** Beta-binomial conjugacy

- Again the **kernel** of the Beta distribution is:

$$\text{Beta}(q;\, \alpha_0, \beta_0) \propto_q q^{\alpha_0 - 1}(1-q)^{\beta_0 - 1}$$

- Recall the terms in the binomial PMF proportional to $q$

$$\text{Binom}(b; N, q) \propto_q q^b (1-q)^{N-b}$$

- If prior is $q \sim \text{Beta}(\alpha_0, \beta_0)$, and likelihood is $b \sim \text{Binom}(N, q)$…

$$\begin{aligned} P(q \mid b, N, \alpha_0, \beta_0) &\propto_q \text{Binom}(b; N, q)\, \text{Beta}(q;\, \alpha_0, \beta_0) \\ &\propto_q q^{b+\alpha_0-1}(1-q)^{N-b+\beta_0-1} \end{aligned}$$

- The **kernel of a (different) Beta** distribution!

# **Bayesian updating:** Beta-binomial conjugacy

- Again the **kernel** of the Beta distribution is:

$$\text{Beta}(q;\, \alpha_0, \beta_0) \propto_q q^{\alpha_0 - 1}(1-q)^{\beta_0 - 1}$$

- Recall the terms in the binomial PMF proportional to $q$

$$\text{Binom}(b; N, q) \propto_q q^{b}(1-q)^{N-b}$$

- If prior is $q \sim \text{Beta}(\alpha_0, \beta_0)$, and likelihood is $b \sim \text{Binom}(N, q)$…

$$P(q \mid b, N, \alpha_0, \beta_0) = \text{Beta}(q;\, \alpha_0 + b,\ \beta_0 + N - b)$$

- **Conjugacy:** the posterior is in the same family as the prior

# **Bayesian updating:** Beta-binomial conjugacy

- If prior is $q \sim \text{Beta}(\alpha_0, \beta_0)$, and likelihood is $b \sim \text{Binom}(N, q)$…

$$P(q \mid b, N, \alpha_0, \beta_0) = \text{Beta}(q;\ \alpha_0 + b,\ \beta_0 + N - b)$$

- e.g., say *N*=1 and b=1 (Bernoulli trial with outcome 1)

$$P(q \mid b, N, \alpha_0, \beta_0) = \text{Beta}(q;\ \alpha_0 + 1,\ \beta_0)$$

- e.g., say *N*=3 and b=2 (Bernoulli trial with outcome 1)

$$P(q \mid b, N, \alpha_0, \beta_0) = \text{Beta}(q;\ \alpha_0 + 2,\ \beta_0 + 1)$$

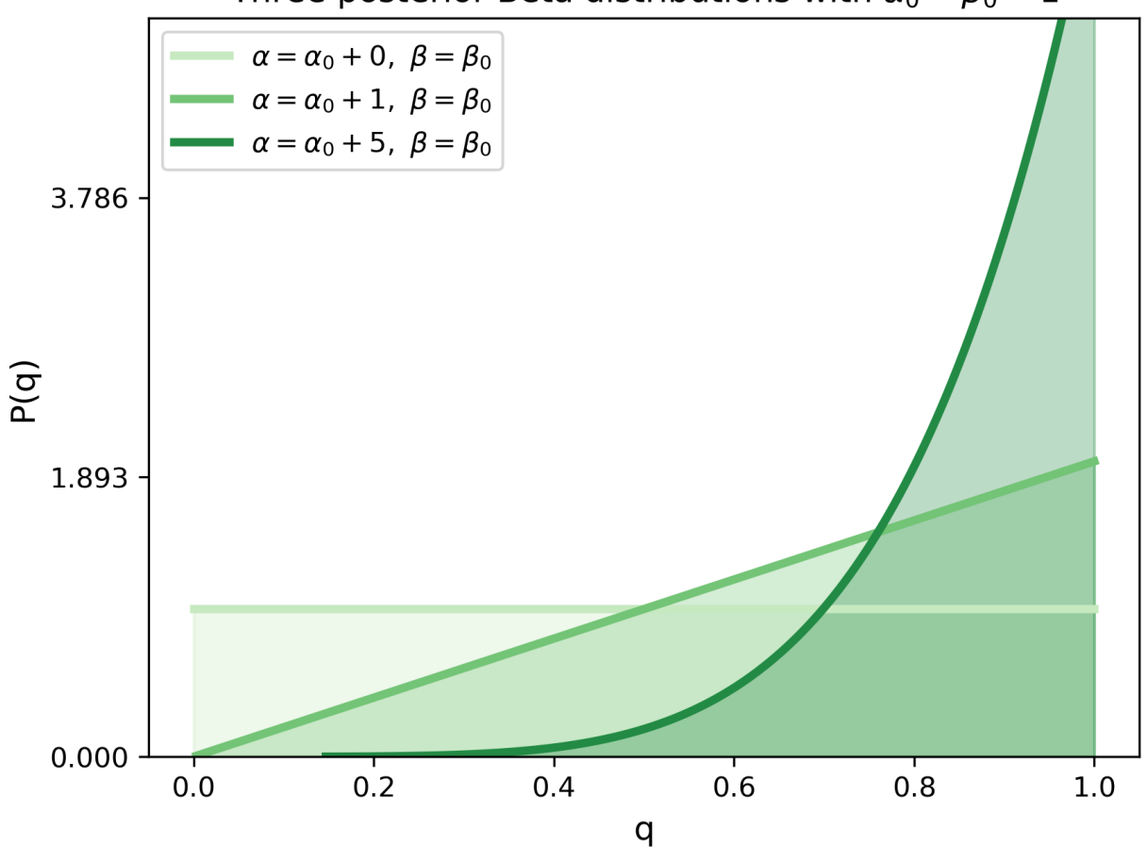
- Notice how the hyperparameters $\alpha_0, \beta_0$ act as **pseudocounts**

# **Bayesian updating:** Beta-binomial conjugacy

- Notice how the hyperparameters $\alpha_0, \beta_0$ act as **pseudocounts**

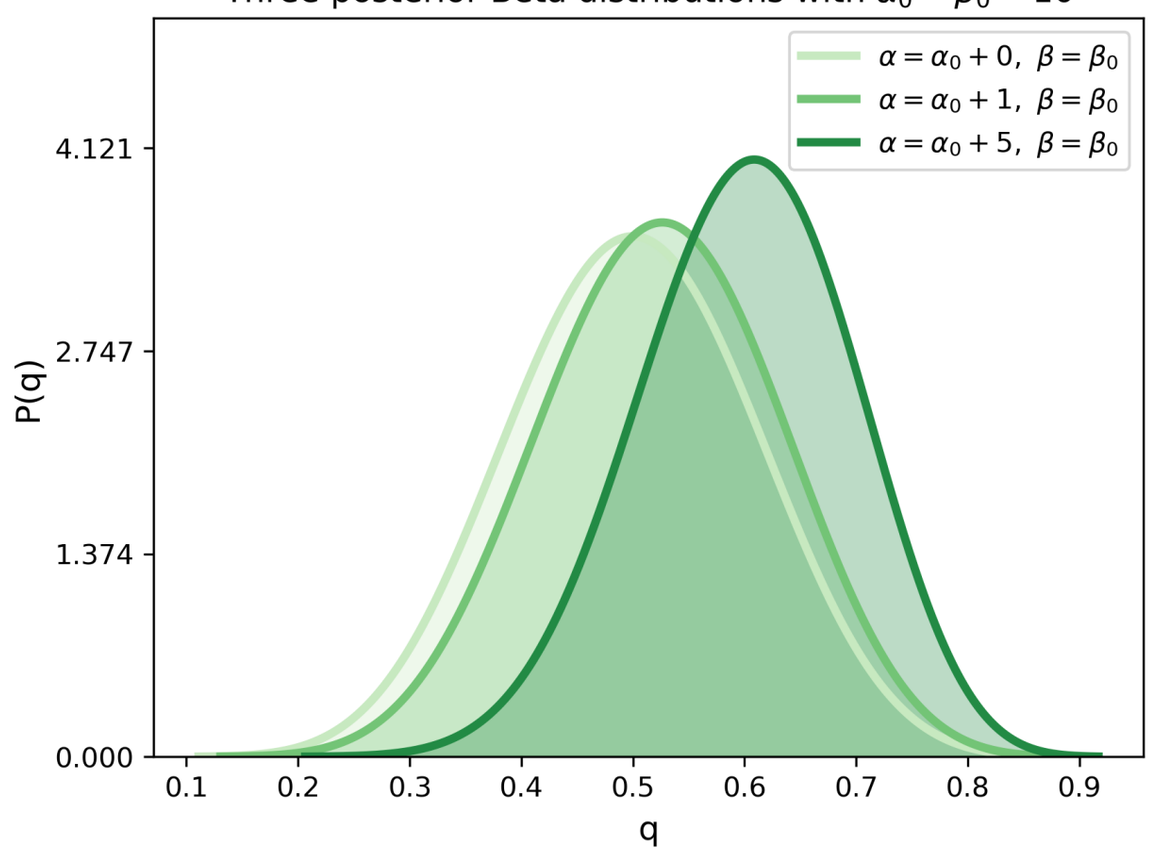
**Weak prior**

Three posterior Beta distributions with $\alpha_0 = \beta_0 = 1$



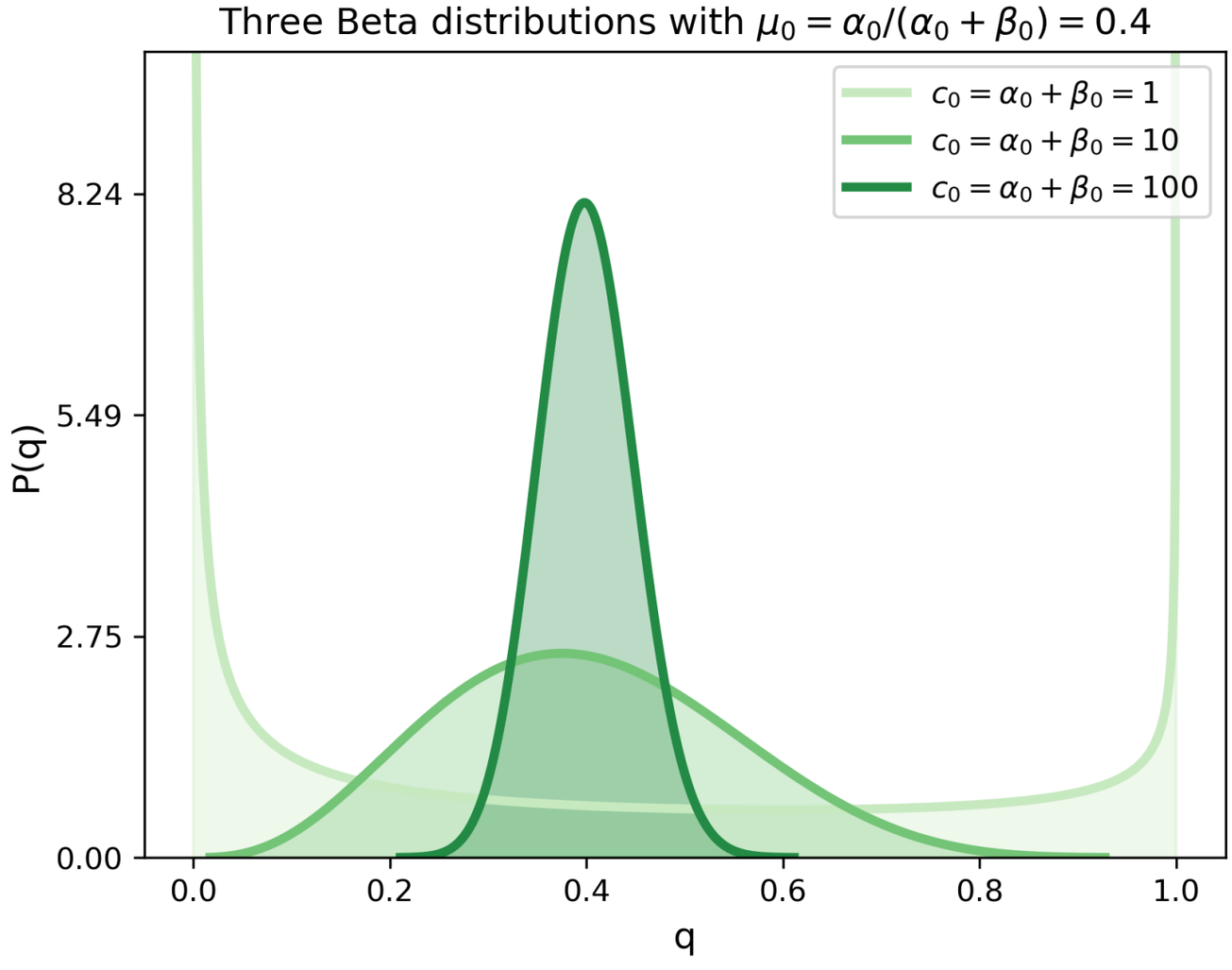

**Strong prior**

Three posterior Beta distributions with $\alpha_0 = \beta_0 = 10$



# **Bayesian updating:** Beta-binomial conjugacy

- Notice how the hyperparameters $\alpha_0, \beta_0$ act as **pseudocounts**
- A common reparameterization is **mean-concentration**




$$\mu_0 = \frac{\alpha_0}{\alpha_0 + \beta_0} \quad \textbf{mean}$$

$$c_0 = \alpha_0 + \beta_0 \quad \textbf{concentration}$$

# **Bayesian updating:** Beta-binomial conjugacy

- Posterior mean of the Beta distribution is

$$E[q \mid \alpha_0, \beta_0, b, N] = \frac{\alpha_0 + b}{\alpha_0 + \beta_0 + N}$$

$$= \frac{b}{N}\left(\frac{N}{\alpha_0 + \beta_0 + N}\right) + \left(\frac{\alpha_0}{\alpha_0 + \beta_0}\right)\left(\frac{\alpha_0 + \beta_0}{\alpha_0 + \beta_0 + N}\right)$$

$$= \frac{b}{N}\left(\frac{N}{c_0 + N}\right) + \mu_0\left(\frac{c_0}{c_0 + N}\right)$$

$$= \hat{q}^{MLE}\,\gamma + \mu_0(1 - \gamma)$$

- Linear combination of the **MLE** and **the prior mean**
- Combination given by **sample size *N*** versus **prior strength** $c_0$
- Smooths MLE away from ``Black Swan" values near 0,1

# **Empirical Bayes:** Learning the prior

- The current setup lets us posit a Beta prior

$$q_k \sim \text{Beta}(\alpha_0, \beta_0)$$

- Then get a posterior at each cell where we did a trial

$$P(q_k \mid b_k, N_k) = \text{Beta}(\alpha_0 + b_k, \beta_0 + N_k - b_k)$$

- How does this help us **generalize** to new cells?

- Answer: **empirical Bayes.** We will *learn the prior*.

# **Empirical Bayes:** Learning the prior

- Consider the **marginal likelihood** with all $q_k$ marginalized out:

$$P(\boldsymbol{b} \mid \boldsymbol{N}, \alpha_0, \beta_0) = \prod_k \int_0^1 \text{Binom}(b_k;\, N_k, q_k)\, \text{Beta}(q_k; \alpha_0, \beta_0)\, dq_k$$

$$= \prod_k \text{Beta-Binom}(b_k;\, N_k, \alpha_0, \beta_0)$$

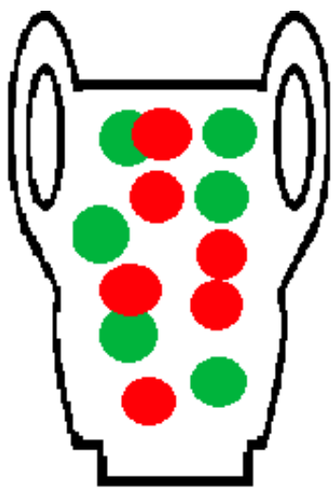
- It equals a product of **beta-binomial distributions**.

- It is not an ``accident'' that the marginal has analytic form (**conjugacy** implies this)

# **Empirical Bayes:** Beta-binomial distribution

$$\text{Beta-Binom}(b_k;\, N_k, \alpha_0, \beta_0) = \frac{\binom{N_k}{b_k} B(\alpha_0 + b_k, \beta_0 + N_k - b_k)}{B(\alpha_0, \beta_0)}$$

- This is a count distribution (same support as binomial)
- Represents **dependent** Bernoulli trials (unlike binomial)
- Often represented as the **Pólya urn sampling scheme**



- There are $\alpha_0$ red balls and $\beta_0$ green balls in an urn
- Pick one at random:
  - If red put **two** red back in
  - If green put **two** green back in
- Repeat $N_k$ times.
- $b_k$ = # of red balls sampled.

# **Empirical Bayes:** Learning the prior

- So the **marginal likelihood** is

$$P(\boldsymbol{b} \mid \boldsymbol{N}, \alpha_0, \beta_0) = \prod_k \text{Beta-Binom}(b_k;\, N_k, \alpha_0, \beta_0)$$

- We can fit the parameters using **type-II maximum likelihood**

$$\hat{\alpha}_0^{MLE}, \hat{\beta}_0^{MLE} = \underset{a,b}{\text{argmax}}\ P(\boldsymbol{b} \mid \boldsymbol{N}, a, b)$$

- We can do this in various ways, details not important for now

  e.g., Minka (2000)'s fixed-point algorithm

# **Empirical Bayes:** Learning the prior

- The parameters $\alpha_0, \beta_0$ parameterize our **marginal likelihood**
- They also parameterize the prior over **latent variables**
- Fitting them with maximum marginal (type-II) likelihood, coincides with obtaining an **empirical prior:**

$$q_k \sim \textbf{Beta}(\hat{\alpha}_0^{MLE}, \hat{\beta}_0^{MLE})$$

- This is a **data-driven** guess for SEP in cells with no trial data
- For a cells with trial data, we still get a posterior

$$P(q_k \mid - ) = \text{Beta}(\hat{\alpha}_0^{MLE} + b_k,\ \hat{\beta}_0^{MLE} + N_k - b_k)$$

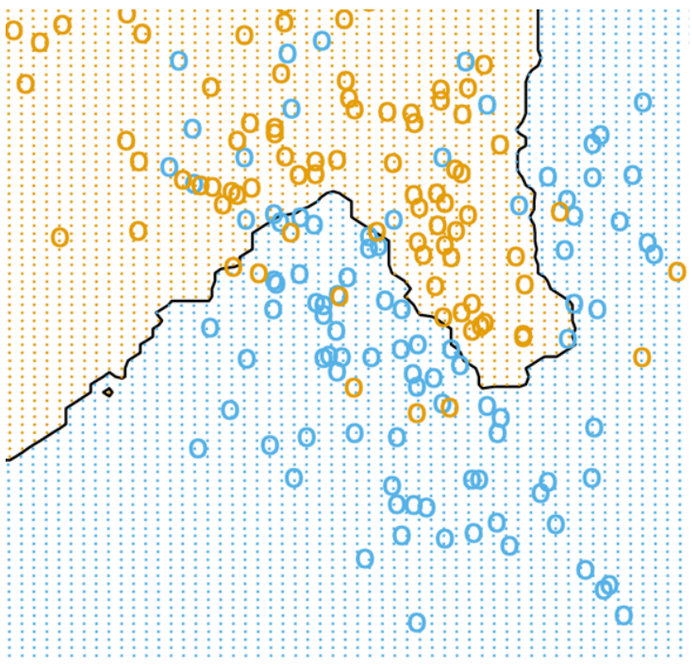
# Summary

## Part I: Supervised learning

- Featurize the problem
- Share information across cells by

$$\hat{q}_k = \hat{q}(\boldsymbol{x}_k)$$

- Learn $\hat{q}(\,\cdot\,)$ discriminatively
- Plug-in $\hat{q}_k$ point estimates



## Part II: Bayesian updating

- Treat $q_k$ as a latent variable
- Share information across cells by

$$q_k \sim \text{Beta}(\alpha_0, \beta_0)$$

- Learn the prior (empirical Bayes)
- Integrate over it