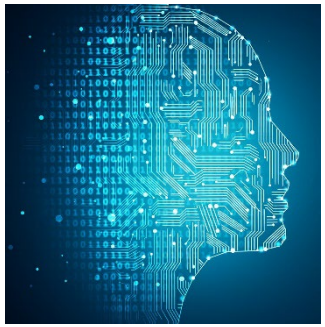
Machine Learning

Validation and Model Selection

Kevin Moon (kevin.moon@usu.edu)

STAT 6655



Training vs Test Error

- Challenge of machine learning: algorithm must perform well on previously unseen inputs
  - Not just training inputs
- In other words, we want the algorithm to ***generalize*** well to unseen data
  - That is, we want the expected error rate on previously unseen inputs to be low
  - This error rate is called the ***generalization error*** or the expected ***test error***
- We also need to estimate the generalization error
  - Done typically by dividing the total amount of data into two sets: ***training data*** and ***test data***
    - More on how this is done specifically later
  - Model is trained on the training data and then tested on the test data
- Expected test error $\geq$ expected training error

Group Exercise

1. For a given machine learning with training and test data, discuss the differences and relationships between training error, test error, and the expected test error.
2. Determine if any of the following can be better than (i.e. less than) the Bayes error: training error, test error, expected test error.

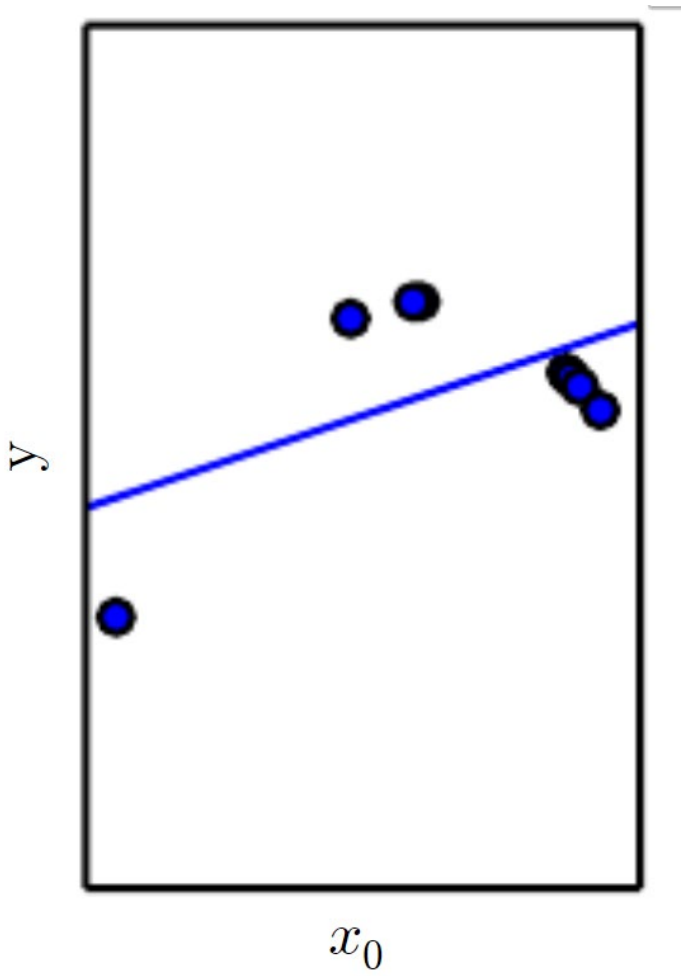
1. **Training error**: the empirical error achieved on the training data after training a machine learning model on the training data.
   Test error: the empirical error achieved on the test data after training a machine learning model on the training data.
   Expected test error: the expected value of the error after training a machine learning model on the training data. The training error and test error are estimators of the expected test error. Thus they may be biased and have nonzero variance.
2. The training error and test error may be less than the Bayes error for finite sample sets. But the expected test error cannot be less than the Bayes error.

Test Error and Hyperparameters

- Why do we care about the test error?
  - Provides an estimate of model performance (useful for comparing different models); referred to as *model assessment*
  - Helps us to choose *hyperparameters*
- Hyperparameters are parameters not determined by the learning algorithm
- Examples: k in k-nearest neighbors, λ in regularized ERM, others we'll see today
- Need some way to select these hyperparameters
- Hyperparameters typically affect *model complexity*
- The process of choosing appropriate hyperparameters is called *model selection*
  - Need to select a model with appropriate complexity

Example: Linear Regression

- Many processes are not well modeled by linear functions
  - Biological systems
  - Stock market prices
  - Periodic signals
- In these cases, linear functions can ***underfit*** the data
  - Training error is very high
  - Test error will also be high



Goodfellow et al., 2016

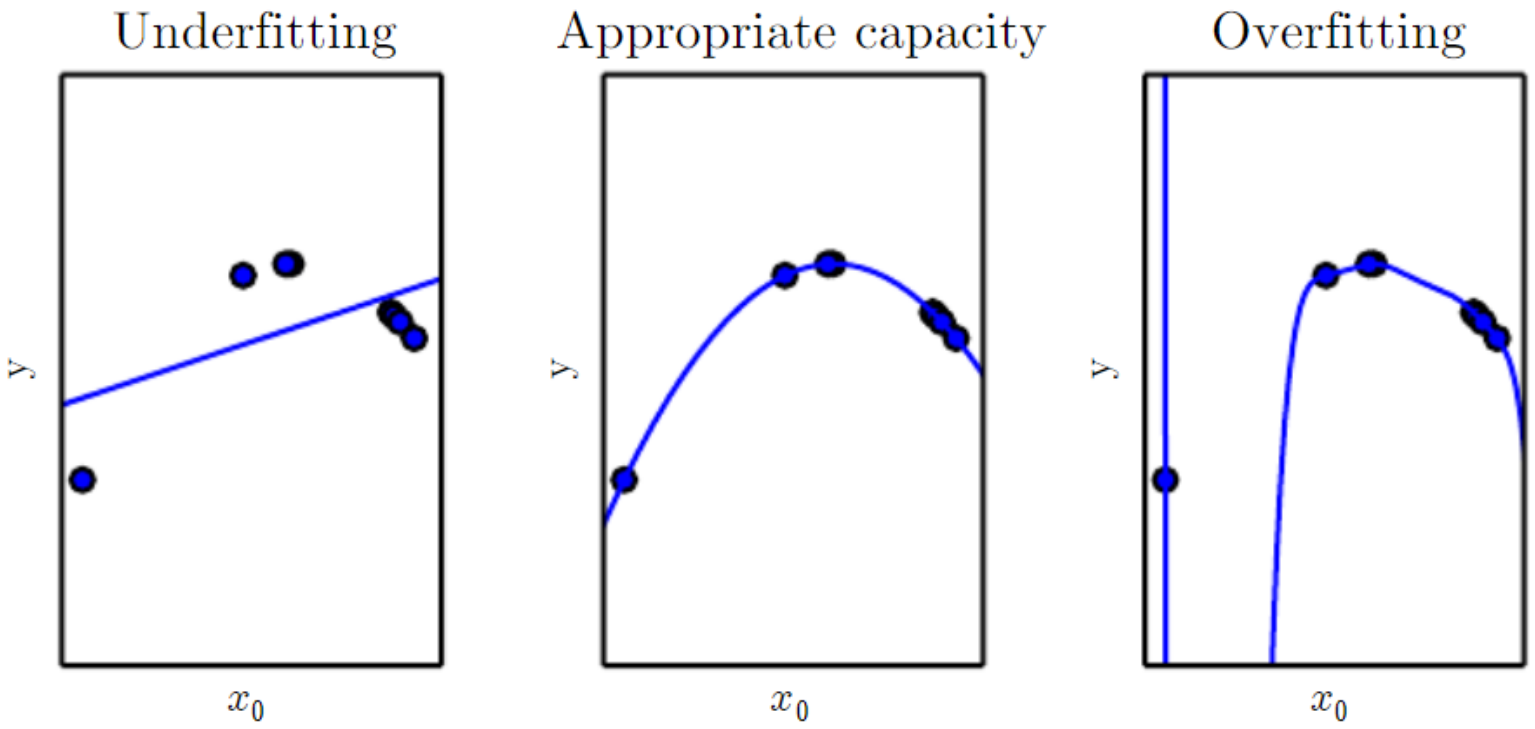
Model Complexity/Capacity

- Solution: increase ***complexity*** of the model
  - Model complexity is the # of independent parameters in the function model (i.e. "degrees of freedom")
- We can increase the complexity of our linear function by adding polynomial terms
  - Example: $d = 1$

$$f(x; \boldsymbol{w}) = \sum_{j=0}^{m} w_j x^j$$

  - Still linear in $\boldsymbol{w}$ ⇒ we can use linear regression!
- Can generalize further by using other nonlinear functions of x (e.g. log or exp)

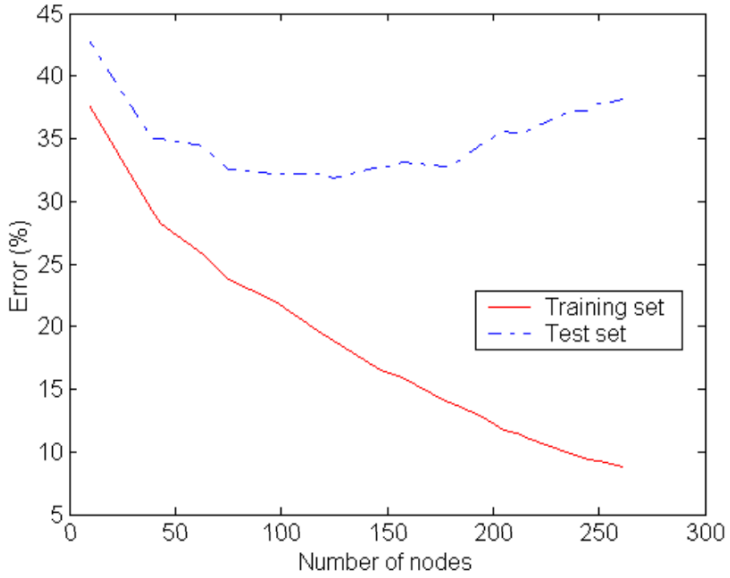
Nonlinear Regression



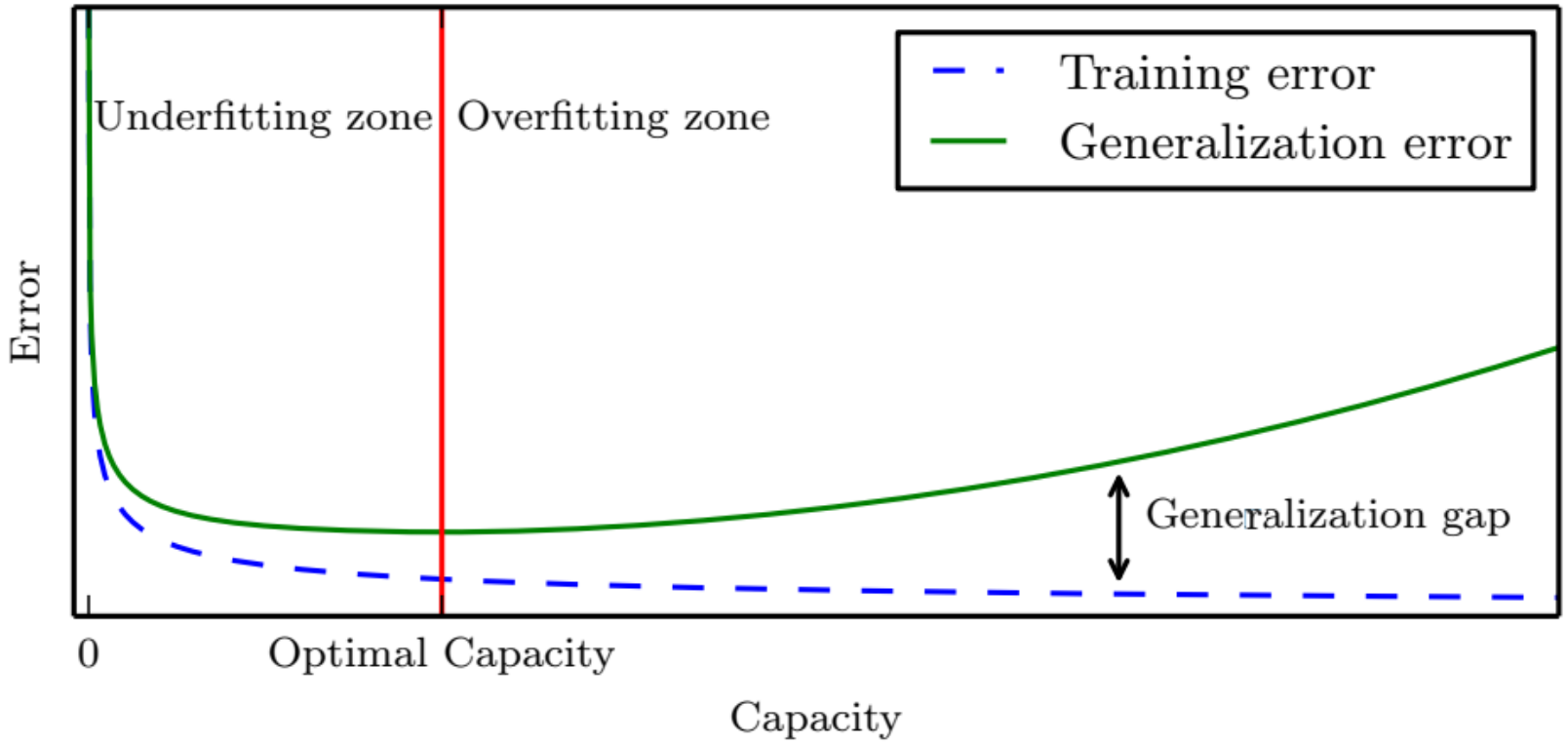
Goodfellow et al., 2016

Overfitting

- Overfitting occurs when the gap between training error and test error is too large
  - The model is capturing unique characteristics of the training set
  - The model does not generalize well to new samples
  - Occurs when the complexity is too large
- Avoiding overfitting
  - Choose model complexity based on test error
  - Generally requires the use of another test set for evaluation



Overfitting and Underfitting



Goodfellow et al., 2016

Penalizing Model Complexity

- Include a penalty on model complexity directly in the cost function:

$$\sum_i L(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i) + \#\text{parameters}$$

- The addition of the penalty is known as ***regularization***
- Regularization: "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error." (Goodfellow et al., 2016)
  - Lots of different penalties can be included in regularization
  - Regularization is useful/important in machine learning

Penalizing Model Complexity

- Ridge regression: penalize with L2 norm

$$\boldsymbol{w}^* = \arg\min \sum_i L(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i) + \lambda \sum_{j=1}^{m} w_j^2$$

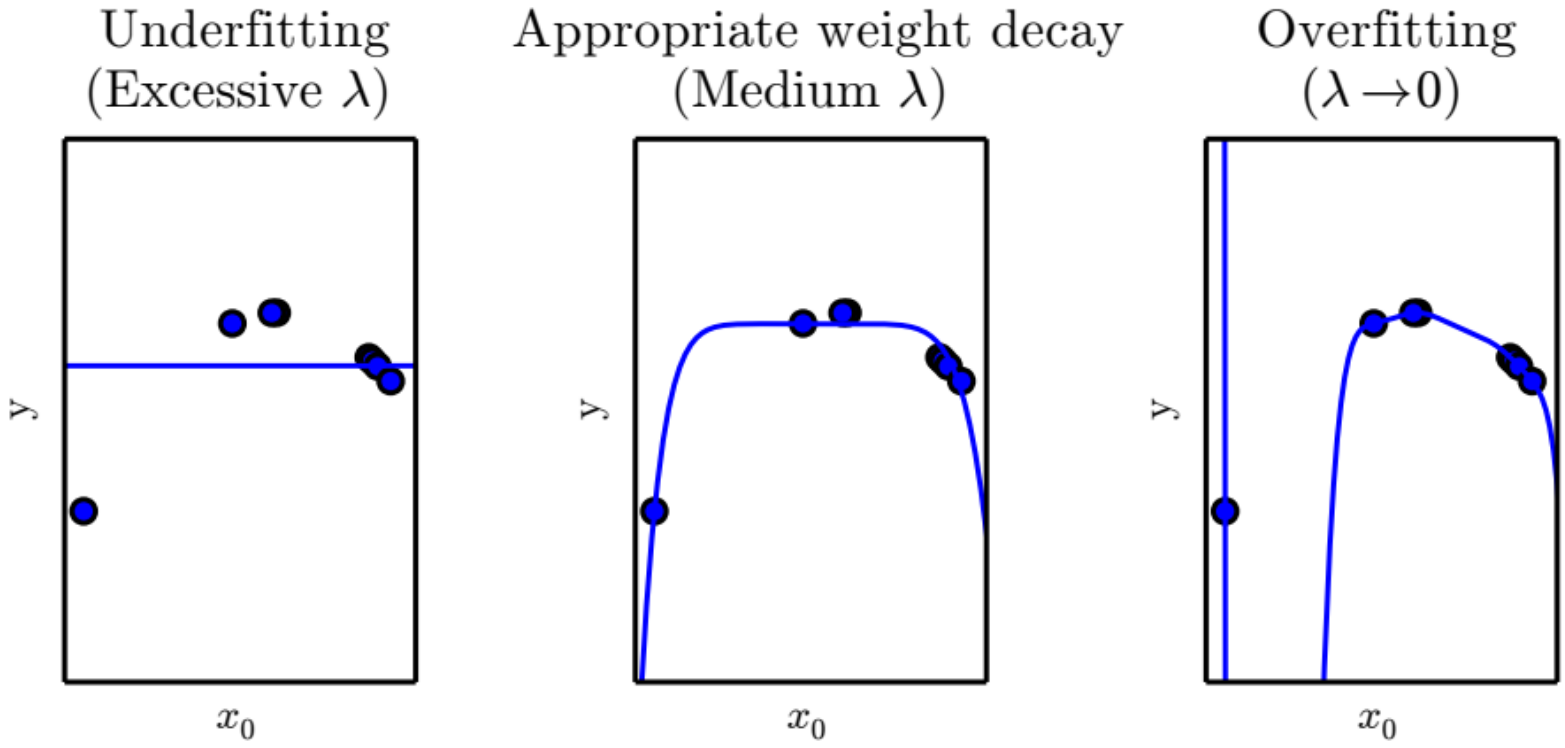
  - Least squares closed form solution exists: $\boldsymbol{w}^* = (\lambda I + X^T X)^{-1} X^T \boldsymbol{y}$

- LASSO regression: penalize with L1 norm

$$\boldsymbol{w}^* = \arg\min \sum_i L(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i) + \lambda \sum_{j=1}^{m} |w_j|$$

  - No closed form solution but still convex (optimal solution can be found)

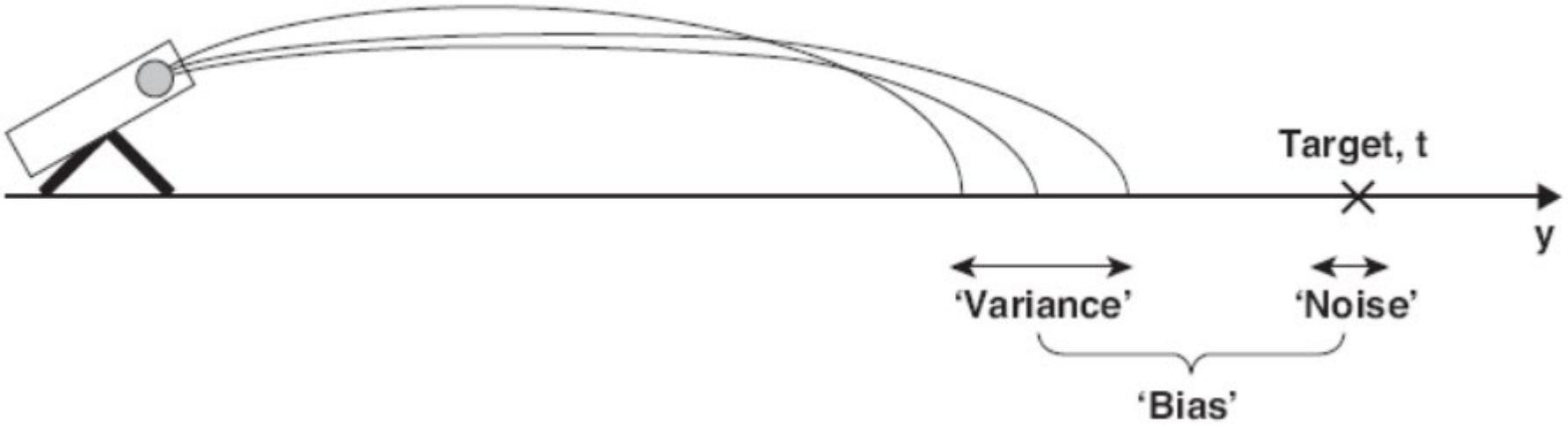
Effect of λ



Goodfellow et al., 2016

Bias-variance tradeoff

- Prediction errors can be formulated as a sum of variance, bias, and noise



- Typically, variance comes from differences between training sets, bias from model assumptions, and noise from other factors

Bias-variance tradeoff

- Consider the regression setting: minimize MSE
- Assume $Y = f(\boldsymbol{X}) + \epsilon$ where $\mathbb{E}[\epsilon] = 0$ and $\mathrm{Var}[\epsilon] = \sigma_\epsilon^2$
- Expected prediction error at input point $\boldsymbol{x}_0$:

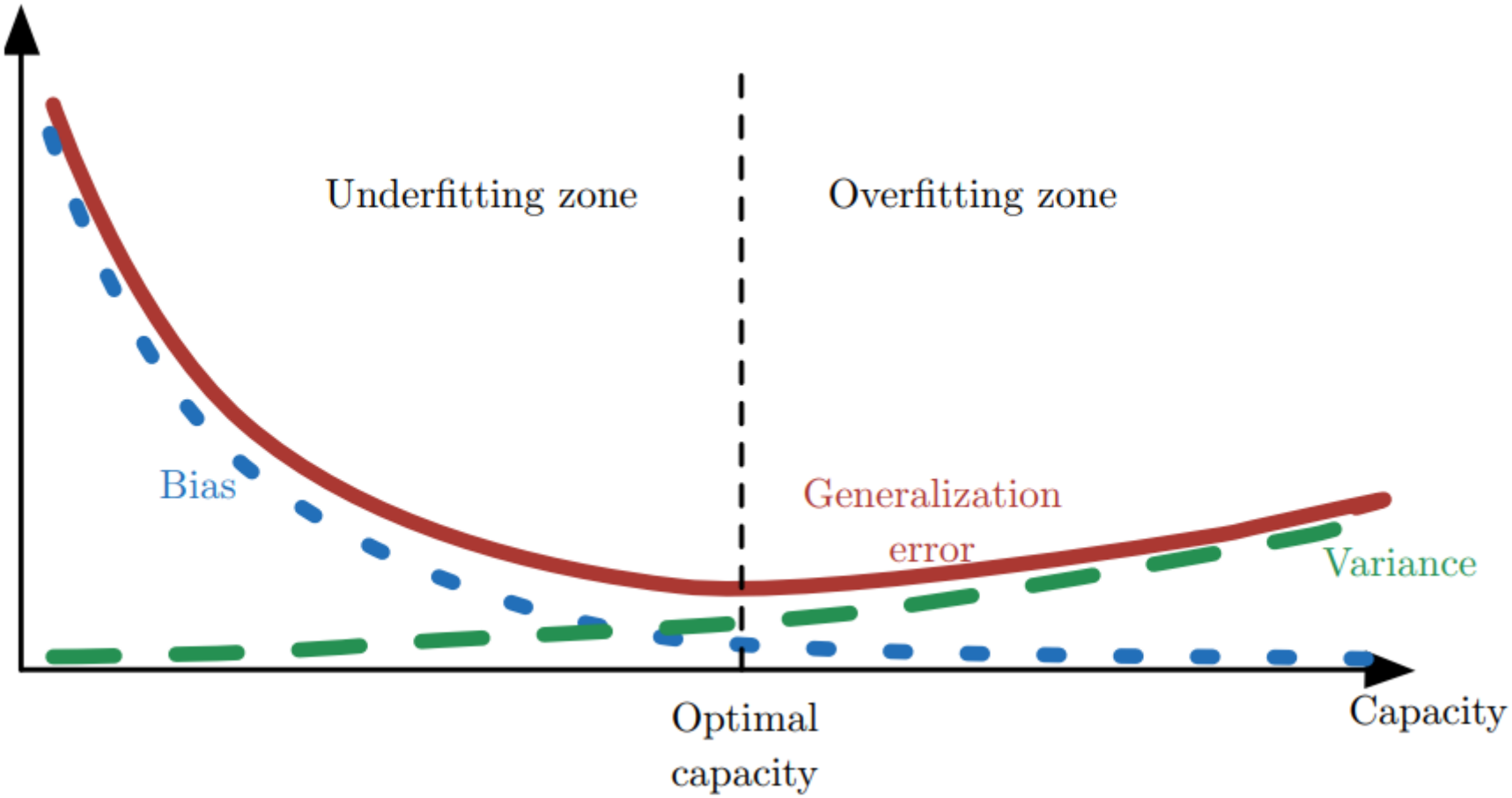
$$Err(\boldsymbol{x}_0) = \mathbb{E}\left[\left(Y - \hat{f}(\boldsymbol{x}_0)\right)^2 | X = \boldsymbol{x}_0\right]$$

$$= \sigma_\epsilon^2 + \left(\mathbb{E}\left[\hat{f}(\boldsymbol{x}_0)\right] - f(\boldsymbol{x}_0)\right)^2 + \mathbb{E}\left[\left(\hat{f}(\boldsymbol{x}_0) - \mathbb{E}\left[\hat{f}(\boldsymbol{x}_0)\right]\right)^2\right]$$

$$= \sigma_\epsilon^2 + \mathrm{Bias}^2\left(\hat{f}(\boldsymbol{x}_0)\right) + \mathrm{Var}\left[\hat{f}(\boldsymbol{x}_0)\right]$$

- First term is the *irreducible error*
- Second term is the squared bias
- Third term is the variance

Bias-variance tradeoff



Example: k-NN regression

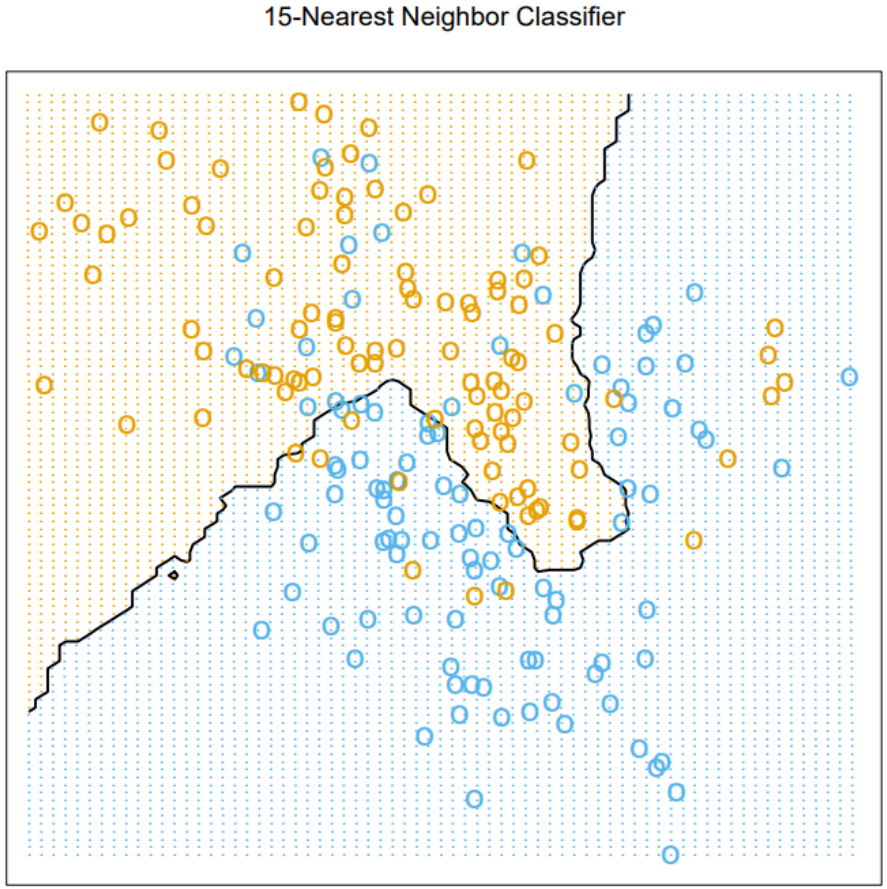
- Let $\boldsymbol{x}_{(\ell)}$ be the ℓth nearest neighbor of $\boldsymbol{x}_0$
- Assume, for simplicity, that the training points are fixed but the labels are random

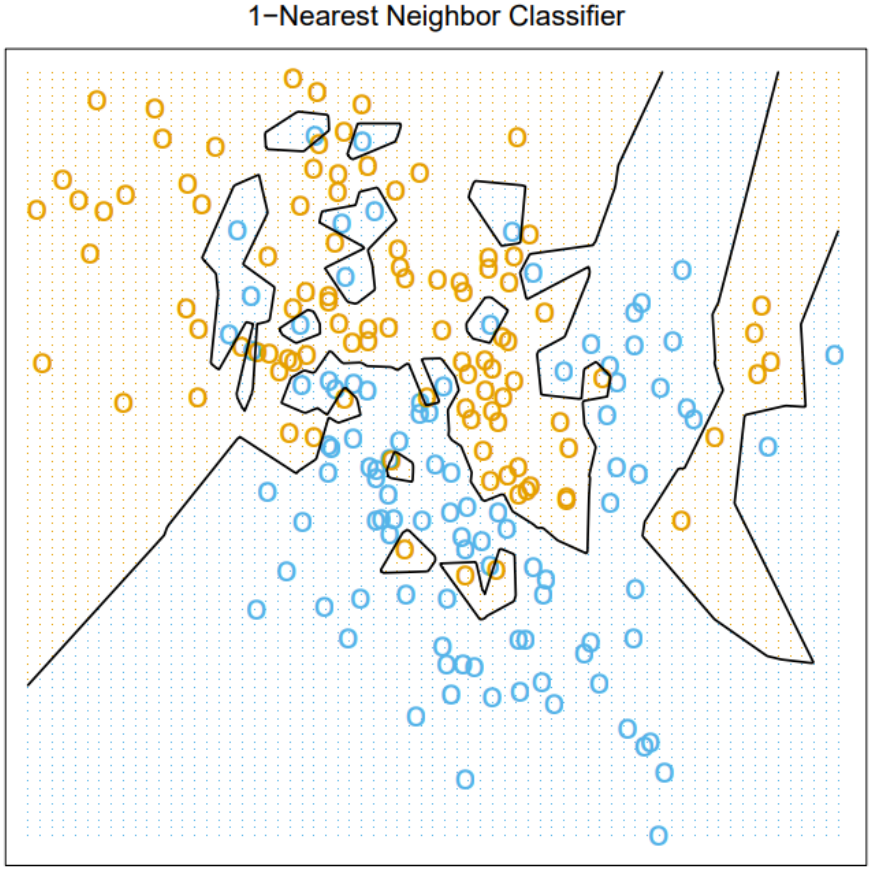
$$Err(\boldsymbol{x}_0) = \mathbb{E}\left[\left(Y - \hat{f}(\boldsymbol{x}_0)\right)^2 | X = \boldsymbol{x}_0\right]$$

$$= \sigma_\epsilon^2 + \left[f(x_0) - \frac{1}{k}\sum_{\ell=1}^{k} f\left(\boldsymbol{x}_{(\ell)}\right)\right]^2 + \frac{\sigma_\epsilon^2}{k}$$

- Complexity increases as k decreases
- Squared bias decreases and variance increases as k decreases

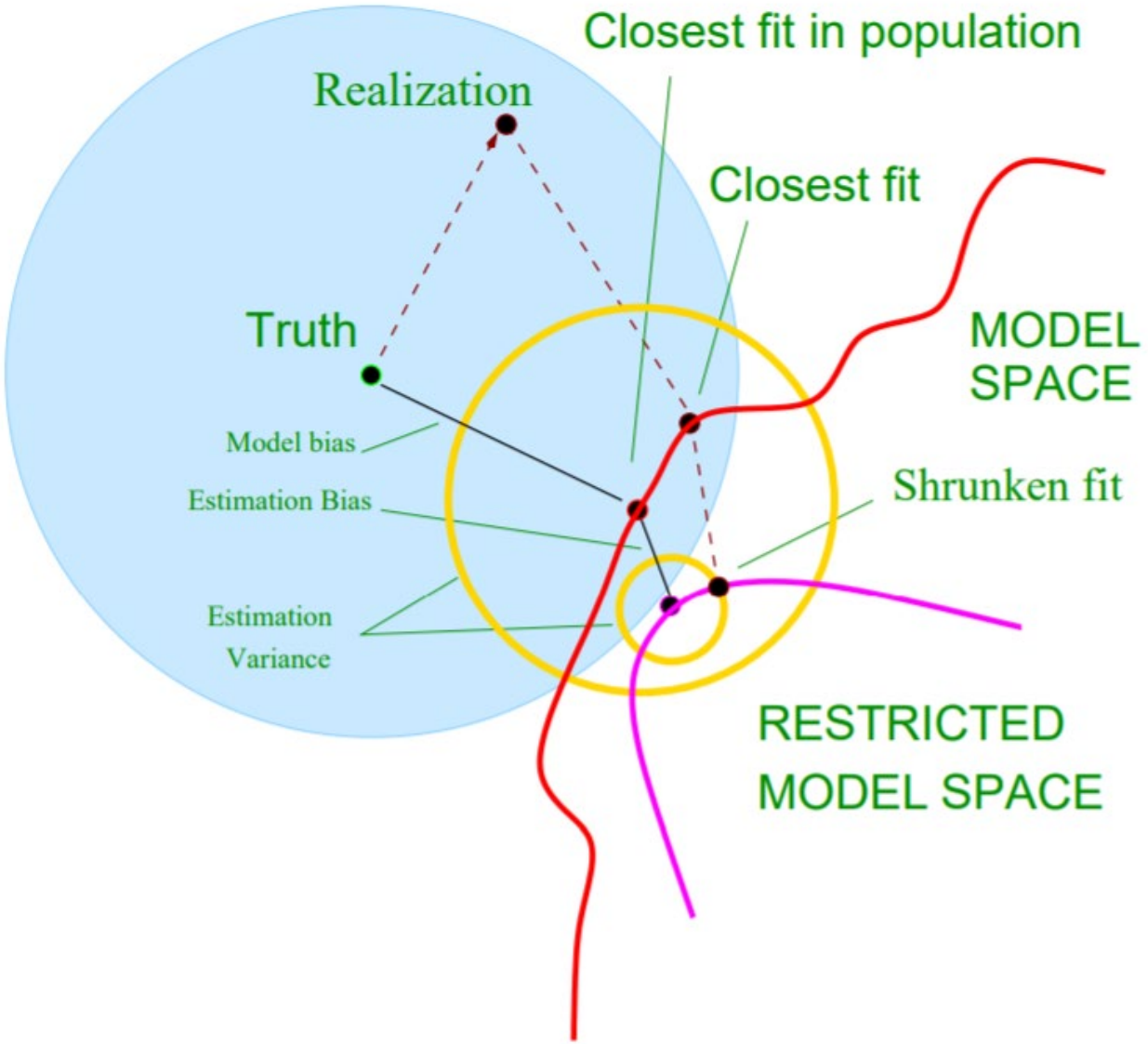
k-nearest neighbor classifier





Hastie et al., 2009

Bias-variance tradeoff

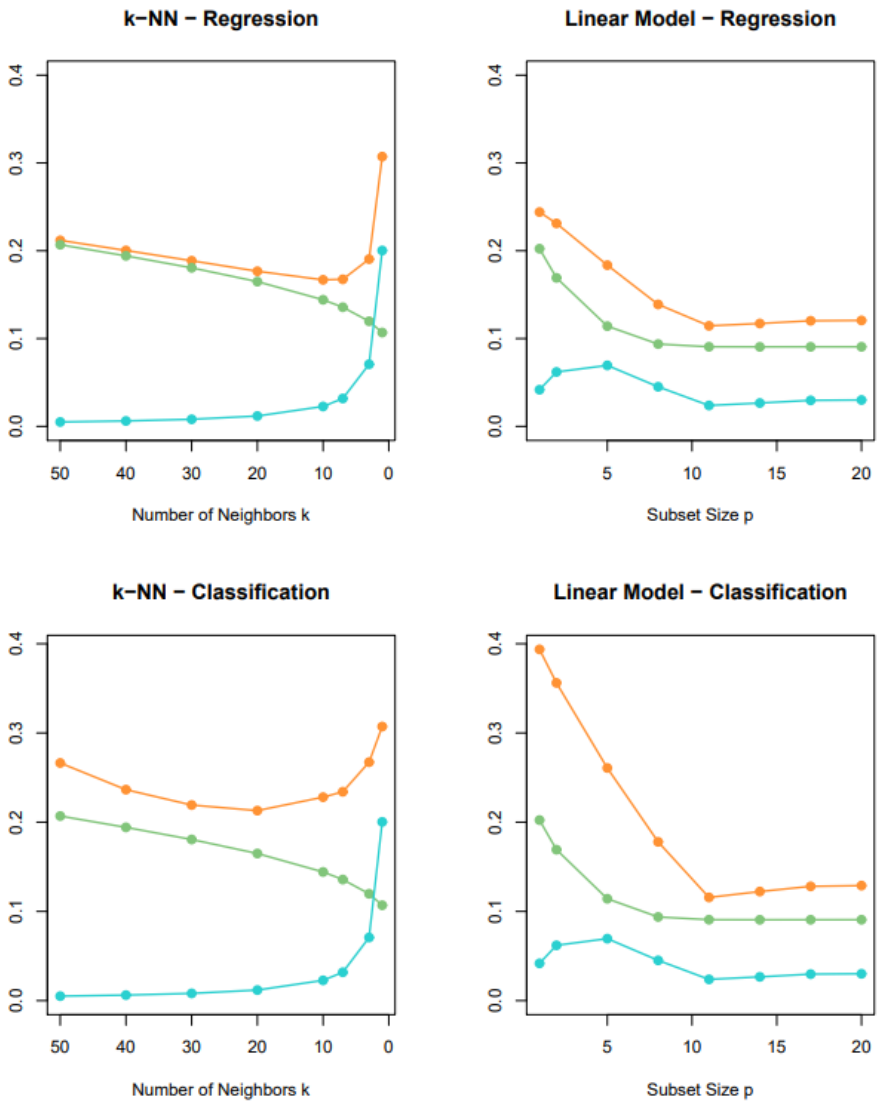


Classification vs. Regression

- The bias-variance tradeoff for classification is somewhat different than for regression
- The best model for regression may not be the best model for classification
- **Example**: 80 observations and 20 predictors, uniformly distributed in the hypercube $[0,1]^{20}$
- k-NN setup: $Y = 0$ if $X^{(1)} \leq 1/2$ and $Y = 1$ if $X^{(1)} > 1/2$
- Linear setup: $Y = 1$ if $\sum_{j=1}^{10} X^{(j)} \geq 5$, $Y = 0$ otherwise
- We'll look at the results using squared-error loss (regression) and the 0-1 loss (classification)

Classification vs. Regression

- Expected prediction error (orange); squared bias (green); variance (blue)
- Squared error loss (top); 0-1 loss (bottom)
- Bias and variance curves are the same for regression and classification, but the prediction error is different
- Why?

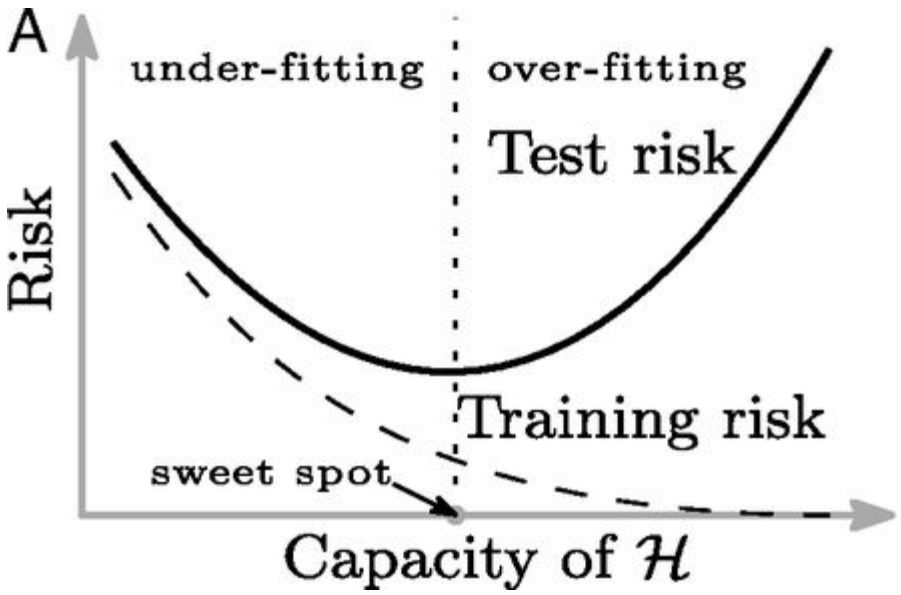


Classification vs. Regression

- Suppose for a given input, the true probability of class 1 is 0.9 while the expected value of our estimate is 0.6
- The squared bias is large, but the prediction error is zero (we make the right classification decision)
- ⇒ estimation errors that leave us on the right side of the decision boundary don't hurt us

Double Descent Curves

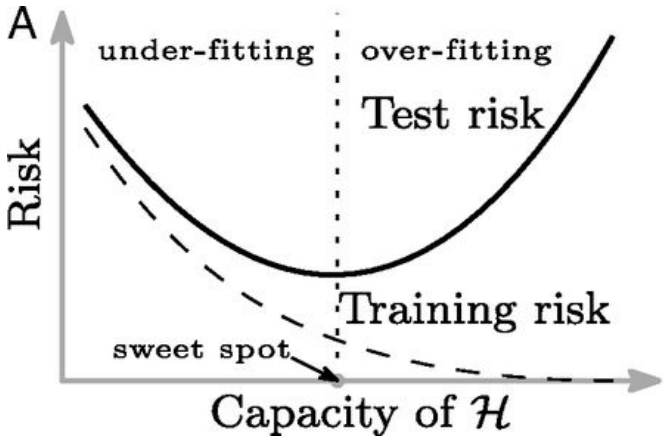
- Actually, the relationship between capacity and generalization isn't as clear as I've portrayed it
- This is the classical view on underfitting/overfitting
- Yet neural networks with lots of parameters often don't overfit. Why not?

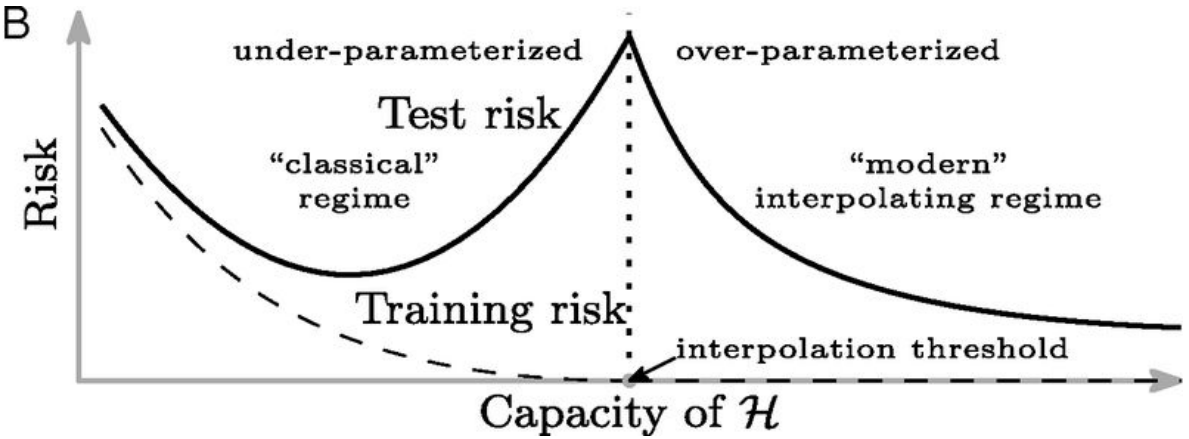


Belkin et al. (2019)

Double Descent Curves

- Left is the classical view on underfitting/overfitting
- Right is what actually happens in a lot of cases

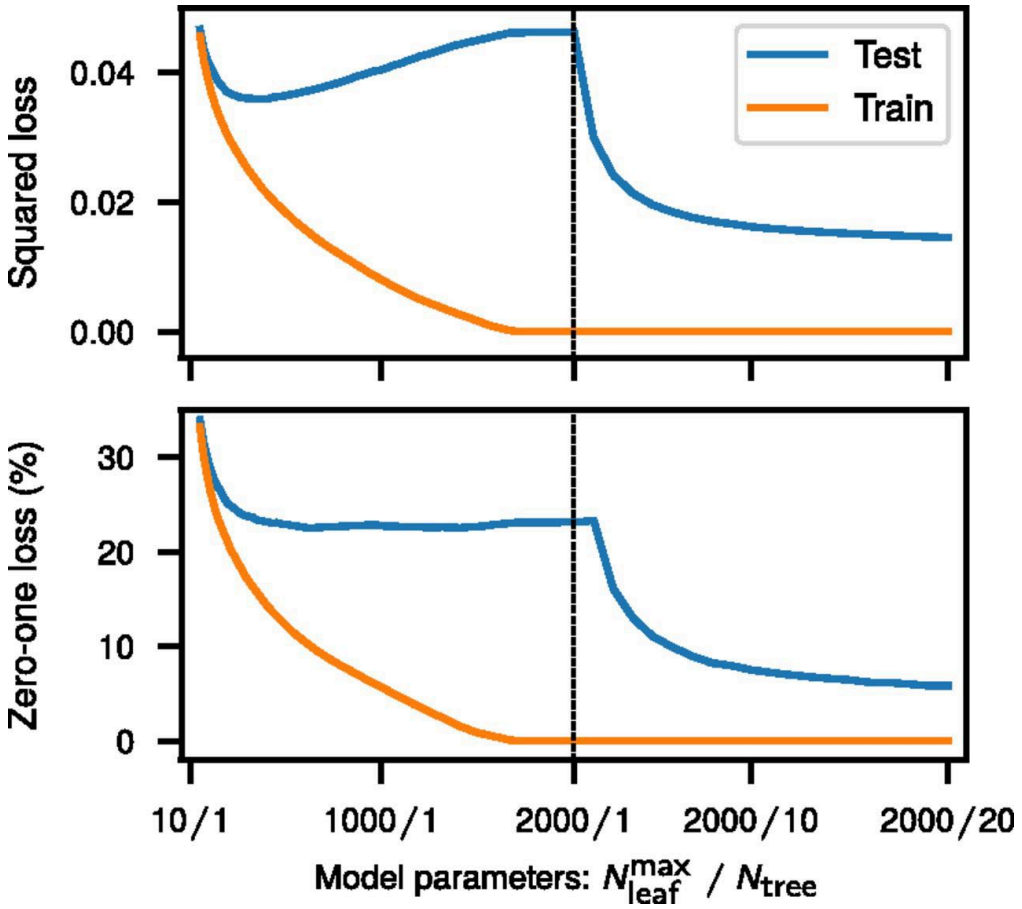




Belkin et al. (2019)

Double Descent Curves

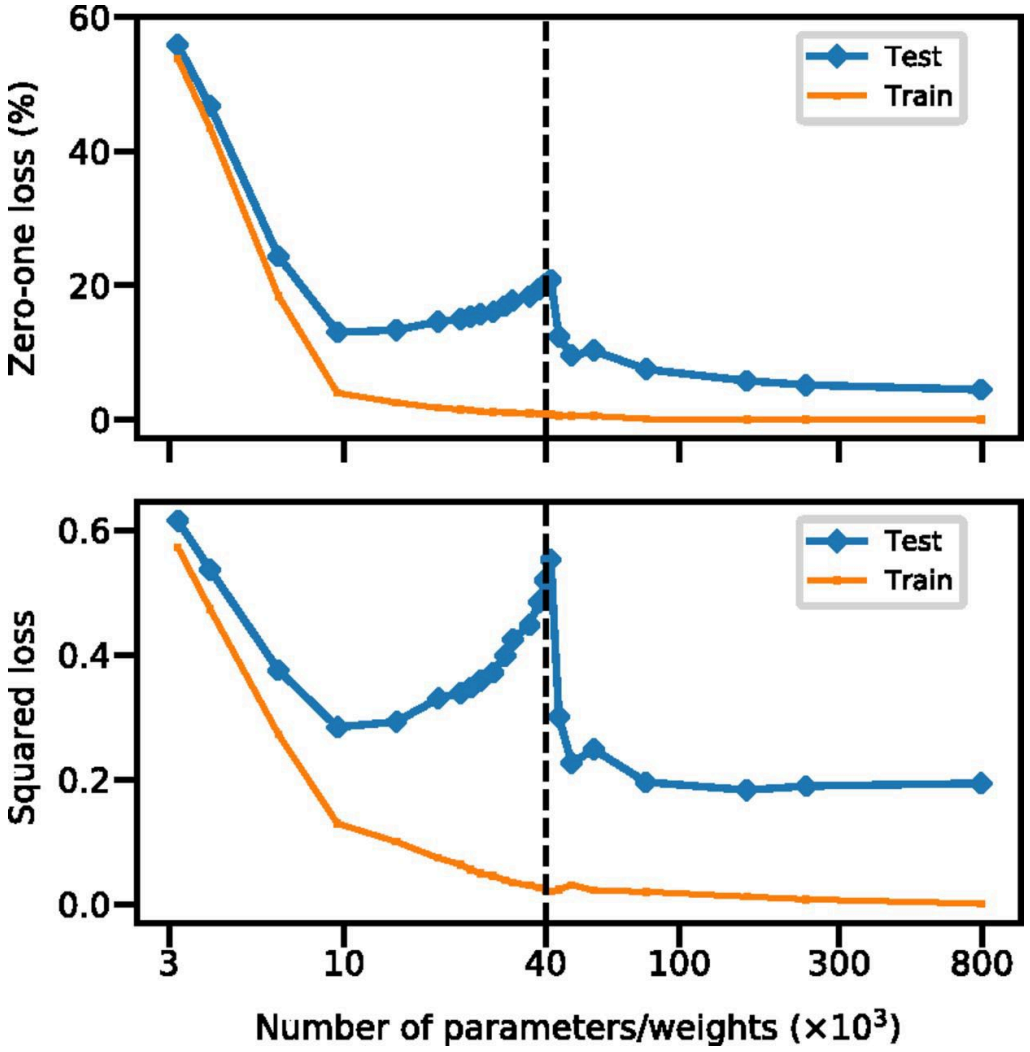
- Random forest example (MNIST)



Belkin et al. (2019)

Double Descent Curves

- Neural Network example (MNIST)



Belkin et al. (2019)

Double Descent Curves

- What's going on here?
- When a problem is overparameterized (# of parameters > # of samples), we often obtain the "minimum norm" solution
  - Choose the solution that minimizes the norm of the weights
  - This tends to give us a fairly smooth function that is able to "interpolate" well despite exactly fitting the training data
- **Note**: the interpolation transition point seems related to the number of samples
  - So if you can, it is often a good idea to have more parameters than samples
- **Note**: regularizations can sometimes have a somewhat similar effect, although they may also prevent us from reaching full interpolation potential

Model Selection by Error Estimation

- **Goal**: find hyperparameters that lead to the smallest risk (e.g. test error)
- Want to estimate the risk given access only to training data
- Let $\{\hat{f}_{\boldsymbol{\theta}}\}$ be a collection of models where $\boldsymbol{\theta}$ spans some range of hyperparameter values
- Consider a risk

$$R(\hat{f}_{\boldsymbol{\theta}}) = \mathbb{E}_{\boldsymbol{X},Y}\left[L\left(Y, \hat{f}_{\boldsymbol{\theta}}(\boldsymbol{X})\right)\right]$$

- A natural estimate from training data is

$$\hat{R}(\hat{f}_{\boldsymbol{\theta}}) = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, \hat{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right)$$

- Problem: biased, leads to overfitting

Holdout Error Estimate

- Idea: Partition training data
  $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m), (\boldsymbol{x}_{m+1}, y_{m+1}), \ldots, (\boldsymbol{x}_{m+n}, y_{m+n})$
- Train on $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)$ to get $\{\hat{f}_{\boldsymbol{\theta}}\}$
  - E.g. train on a grid of different $\boldsymbol{\theta}$ values
- Holdout error estimate:

$$\hat{R}_{HO}(\hat{f}_{\boldsymbol{\theta}}) = \frac{1}{n} \sum_{i=m+1}^{m+n} L\left(y_i, \hat{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right)$$

- Model selection:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \hat{R}_{HO}(\hat{f}_{\boldsymbol{\theta}})$$

K-Fold Cross Validation (CV)

- Fix an integer $2 \leq K \leq n$
- Idea: Partition training data into K roughly equally sized groups called "folds"
- Let $I_1, \ldots, I_K$ be a partition of $\{1, \ldots, n\}$ such that

$$|I_j| \approx \frac{n}{K}, \qquad j = 1, \ldots, K$$

- Define $\hat{f}_{\boldsymbol{\theta}}^{(i)}$ = model trained on $\{(\boldsymbol{x}_i, y_i)\}_{i \notin I_j}$
- The error estimate for fold j is

$$\hat{R}^{(j)}\left(\hat{f}_{\boldsymbol{\theta}}\right) = \frac{1}{|I_j|} \sum_{i \in I_j} L\left(y_i, \hat{f}_{\boldsymbol{\theta}}^{(j)}(\boldsymbol{x}_i)\right)$$

- K-fold cross validation error estimate:

$$\hat{R}_{CV}\left(\hat{f}_{\boldsymbol{\theta}}\right) = \frac{1}{K} \sum_{j=1}^{K} \hat{R}^{(j)}\left(\hat{f}_{\boldsymbol{\theta}}\right)$$

Thoughts on CV

- When $K = n$, the method is called *leave-one-out* CV
- 5- and 10-fold CV are generally recommended. Higher values of K tend to not be worth the added computational cost.
  - Although, for larger sample sizes, 2- and 3-fold CV may be enough
- To reduce the variance of the estimate and if you have the time/resources, it's good to compute several CV estimates based on different random partitions and average them.
- In classification, the sets I_k should be chosen so that the proportions of different classes in each fold are the same as in the full sample.
- The *bootstrap* is another common error estimation technique that can be used for model selection.
- After selecting a hyperparameter, the final model is obtained by retraining on all training data with the selected parameter.

Holdout vs. CV

- When should I use the holdout method vs. CV?
- Holdout approach can have high variance…
- But CV can be computationally intensive
  - Train and test on all of the data K times
- Typically, CV is used when the dataset isn't too large and the holdout method when the dataset is large
  - The variance from the holdout approach isn't as large if the test dataset is also large
  - Most deep learning approaches (which require large amounts of data) use the holdout method

Validation and Test Datasets

- Is it possible to still overfit when using CV or holdout approaches?
- Yes, unfortunately
  - Especially with the holdout approach
- One way to get around this is to create training, validation, and test datasets
- Train on the training, use the validation error to tune, and then apply the trained model to the test dataset for your final result
- What if you don't like the test error in this case?
- You could try changing your model, but then you risk overfitting again…
  - No good solution to this problem except to have another test dataset
- Common split is 50% training, 25% validation, 25% test

Group Exercise

1. Write out the steps for K-fold cross-validation in pseudocode.

Confusion Matrices

- In classification, sometimes we want more info than just the test error
- Confusion matrices break down the classification performance by class
- Extend easily to multiple classes
- Generally recommended

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Further Reading

- ISL, Chapter 5
- ESL, Chapter 7
- https://arxiv.org/pdf/2108.02497v1.pdf