Introduction to Modern AI
Week 3: Supervised Learning II - Non-parametric models, Ensembles, and Validation

Gavin Hartnett

PRGS, Winter Quarter 2022

Topics we will cover this week

- Non-parametric models
  - linear regression, logistic regression, and NNs are all examples of parametric models
  - There are many other important parametric models in addition to these
  - There is also a large collection of non-parametric models such as k-nearest neighbors and tree-based models
  - We will remain in the supervised learning setting
- Validation
  - We will discuss methods for validating the performance of a trained model.
  - Again, this applies to both parametric and non-parametric models, but we will focus on non-parametric models here.
- Ensembles
  - It is often desirable to aggregate or pool many individual models to form an ensemble of models
  - This can be done for parametric or non-parametric models, here we will primarily focus on ensembles of the latter

Review HW1 Solution

Go over the solutions in class

Non-Parametric Models: kNN and Trees

k-NN

- Can be used for regression or classification
- Regression
    - The prediction at a point x is just the average value of the k closest points to $\boldsymbol{x}$ in the dataset

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

    - These k-closest points are called the neighborhood $N_k(\boldsymbol{x})$
- Classification
    - The estimate for the probability that the class is c is just the fraction of points in the neighborhood that are class c

$$\hat{p}(y = c|\boldsymbol{x}) = \frac{1}{k} \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} \delta_{y(\boldsymbol{x}),c}$$

k-NN

- k is a hyper-parameter
  - We are free to choose it
  - not directly optimized over, model is not differentiable wrt to it
  - k is unrelated to number of classes K
- The existence of a neighborhood function $N_k(\boldsymbol{x})$ presupposes a distance metric
  - Euclidean distance (L^2 norm):

$$||\boldsymbol{x} - \boldsymbol{x}'||_2 = \sqrt{\sum_{i=1}^{d}(x_i - x_i')^2}$$

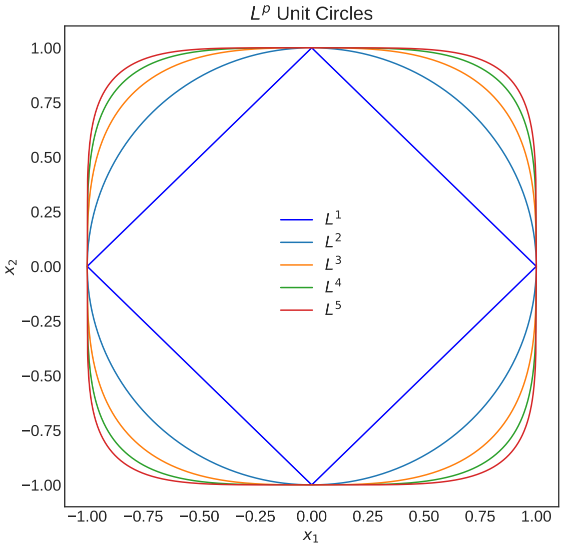
  - Taxicab or Manhattan distance (L^1 norm):

$$||\boldsymbol{x} - \boldsymbol{x}'||_1 = \sum_{i=1}^{d}|x_i - x_i'|$$

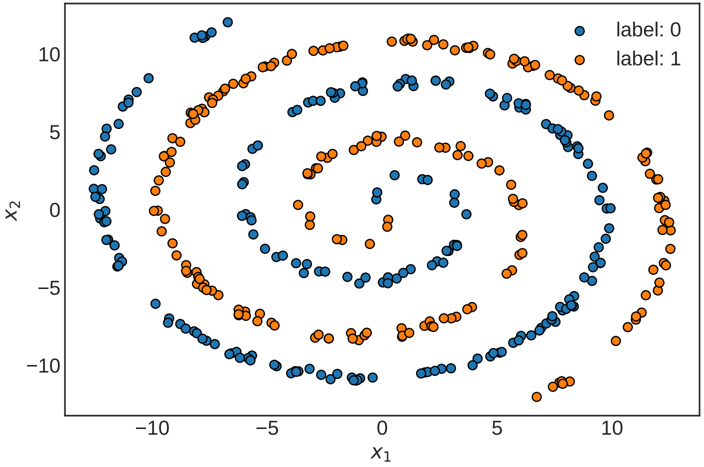
  - More generally, L^p norm:

$$||\boldsymbol{x} - \boldsymbol{x}'||_p = \sqrt[p]{\sum_{i=1}^{d}|x_i - x_i'|^p}$$

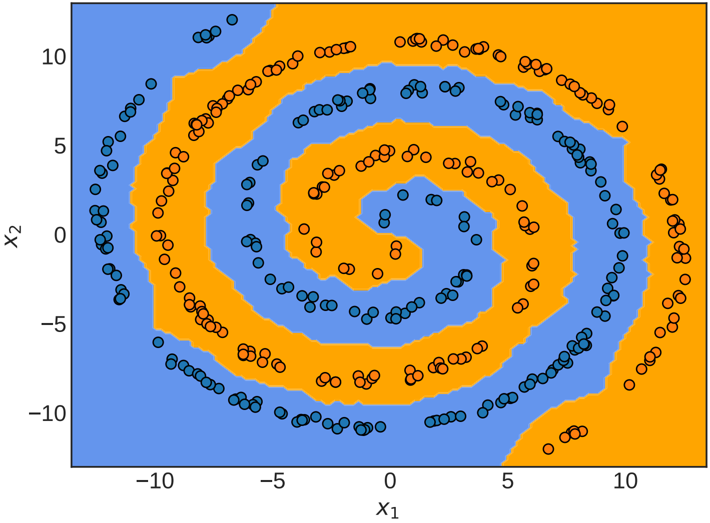
Aside L^p Norms



k-NN: Classification Example

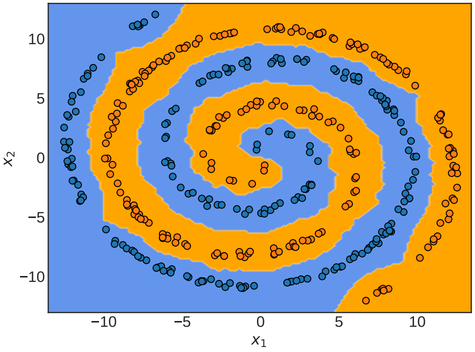


k-NN: Classification Example

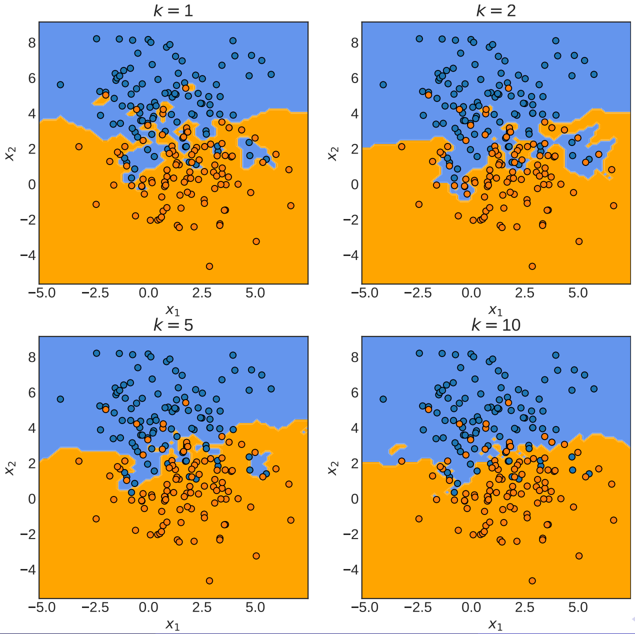


k-NN: Classification Example

- Here $k = 5$
- distance is Euclidean
- Accuracy is 100% (!)
- Compared to logistic regression
    - Decision boundary is non-linear, appears to be very flexible
    - Model has no parameters - predictions determined directly from dataset



k-NN: Classification Example #2



Tree-Based Models

- Tree-based methods are another powerful class of non-parametric models
- Such methods form the basis for many top-performing ML model, such as XGBoost
- We will discuss one popular method for dealing with trees, Classification And Regression Trees (CART), though others exist
- Single-tree models have drawbacks, but can become quite powerful when used in conjunction with ensembling methods (random forests)
- Next few slides closely follow Sec 9.2 of *Elements of Statistical Learning*

Tree-Based Models

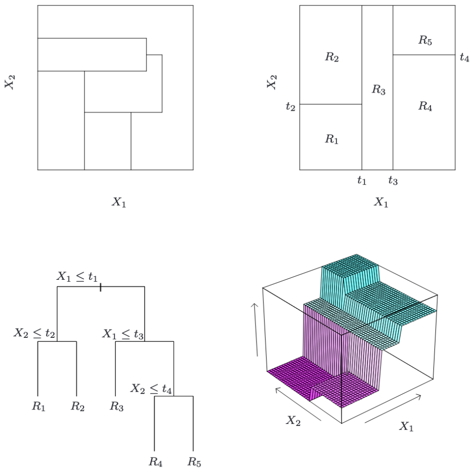


FIGURE 9.2. *Partitions and CART. Top right panel shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data. Top left panel shows a general partition that cannot be obtained from recursive binary splitting. Bottom left panel shows the tree corresponding to the partition in the top right panel, and a perspective plot of the prediction surface appears in the bottom right panel.*

Tree-Based Models

- Basic idea: use a *binary* tree to hierarchically split input domain into rectangles
- Regression: fit a constant to each domain

$$f(x) = \sum_{m=1}^{M} c_m \, I(x \in R_m)\,, \quad \hat{c}_m = \text{ave}(y_i | x_i \in R_m)$$

- Classification: assign a class $k(m)$ for each region $m = 1, ..., M$. Let $k(m)$ be the most popular class for training data in region R_m.
- Main question: how to fit (grow) the tree to data?

Tree-Based Models

How to grow a regression tree

- Scan over data dimension j (integer) and split point s (real number)
- These separate the data into two distinct regions:

$$R_1(j,s) = \{\boldsymbol{x} | x_j \leq s\}\,, \quad R_2(j,s) = \{\boldsymbol{x} | x_j > s\}\,.$$

- Fit the constants $\hat{c}_1, \hat{c}_2$ according to $\hat{c}_m = \mathrm{ave}(y_i | \boldsymbol{x}_i \in R_m)$
- Compute the MSE for both regions:

$$\sum_{\boldsymbol{x}_i \in R_1} (y_i - \hat{c}_1)^2 + \sum_{\boldsymbol{x}_i \in R_2} (y_i - \hat{c}_2)^2$$

- Find the (j,s) pair that minimizes this
- Repeat process for each region separately

Tree-Based Models

Some comments

- When to stop growing the tree? How large should it be?
- General idea is to grow a large tree first, then prune it by collapsing nodes (merging regions)
  - Larger trees are more expressive, but have higher variance
- Are trees really non-parametric?

Model Validation, Bias-Variance Trade-Off, Regularization

Train/Test Split

- Then primary goal in ML is to use data to develop a model that performs well on unseen data (called *generalization*)
- We would like to have some idea of how the model will perform on such unseen data
- The main approach to this problem is to split the data into a train and test set
  - Train set: used to train/fit the model
  - Test set: used to evaluate model performance. Proxy for "unseen" data.
- Generally, more data corresponds to a better model, so we want to maximize N_{train}.
- However, having a good estimate for model performance is also important, so want N_{test} to be large as well.
- No correct way to split the data, often 70/30 or 80/20 is done.

Train/Validation/Test Split

- Often the data is actually split into 3 datasets
  - Train set: used to train/fit *multiple* models
  - Validation set: used to select the best model
  - Test set: used to evaluate final model performance. Proxy for "unseen" data.
- Often we train multiple models instead of just 1
  - Ensemble methods
  - Cross-validation
  - Regularization
- Once we use the validation set to select the best model, it has been "burned"
- Want test set to give us as unbiased a view of the final model performance as possible

Bias-Variance Trade-Off

- We've now observed a few interesting points hinting at a fundamental concept in ML dubbed the *bias-variance trade-off*
- The central goal of supervised learning is to develop a model that correctly captures the signal, but not the noise
  - If only the signal is captured, the model will do well when applied to previously unseen data
  - If the noise is captured too much, the model will not generalize and is said to be overfit
- By making the model more expressive (e.g., by adding more parameters or reducing k in kNN) the model is capable of representing a wider class of functions (less bias)
- On the other hand, with more expressive models there is a danger that the model will fit the noise as well as the signal (high variance)
- Synonyms: capacity, complexity, expressivity

Bias-Variance Trade-Off

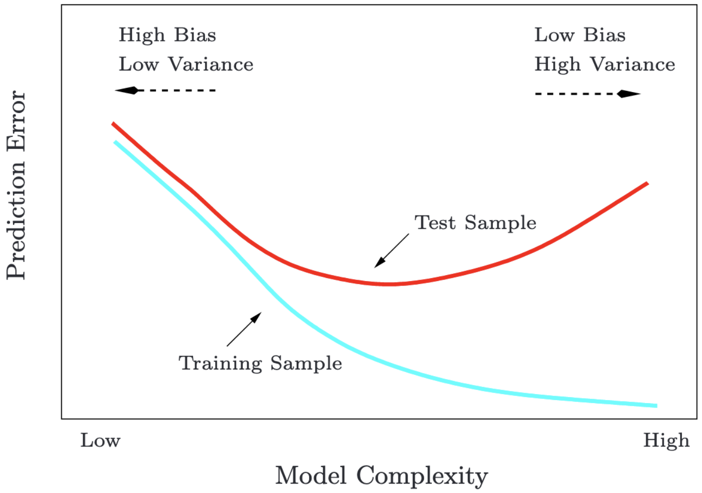


FIGURE 2.11. *Test and training error as a function of model complexity.*

Bias-Variance Trade-Off

Let's be more precise here

- Assume $y = f(x) + \epsilon$ is the true relationship between x, y
- Here ϵ is a random variable with $\mathbb{E}[\epsilon] = 0$, $\mathrm{Var}[\epsilon] = \sigma^2$
  - Recall that $\mathrm{Var}[X] = \mathbb{E}\left[X^2 - (\mathbb{E}[X])^2\right]$
  - And that $\mathbb{E}$ just means expectation, i.e. $\mathbb{E}[f(X)] = \int \mathrm{d}x\, p(x) f(x)$
- Also, let the predictions be given by $\hat{y}(x) = \hat{f}(x; \mathcal{D})$
- Can show (HW exercise!) that the MSE decomposes into 3 terms:

$$\begin{aligned}\mathrm{MSE}(x) &= \mathbb{E}_\epsilon \left(\hat{y}(x) - y\right)^2 \\ &= \mathrm{Bias}[\hat{f}(x;\mathcal{D})]^2 + \mathrm{Var}[\hat{f}(x;\mathcal{D})] + \sigma^2\end{aligned}$$

- $\mathrm{Bias}[\hat{f}(x)] = (f(x) - \hat{f}(x;\mathcal{D}))$
- Bias term: represents error due to incorrect model assumptions
- Variance term: represents error due to overfitting
- σ^2 term: irreducible error due to noise

Bias-Variance Trade-Off

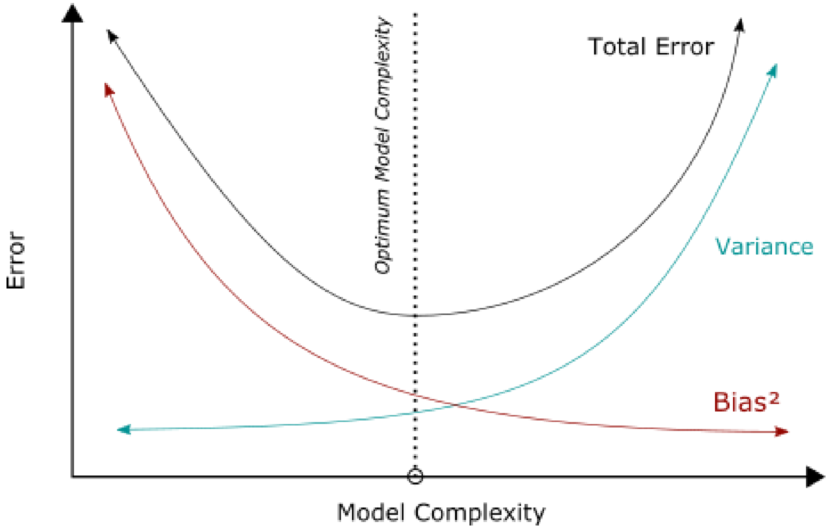


Figure: Source: Wikipedia

Regularization

- Q: Confronted with the fundamental tension of the bias-variance trade-off, how should we make the best determination of model complexity?
- A: Evaluate a range of models with different complexities on a validation set, and take the model with the best performance on this set
- Q: What's the best way to generate a range of similar models with varying complexities?
- One answer is regularization.

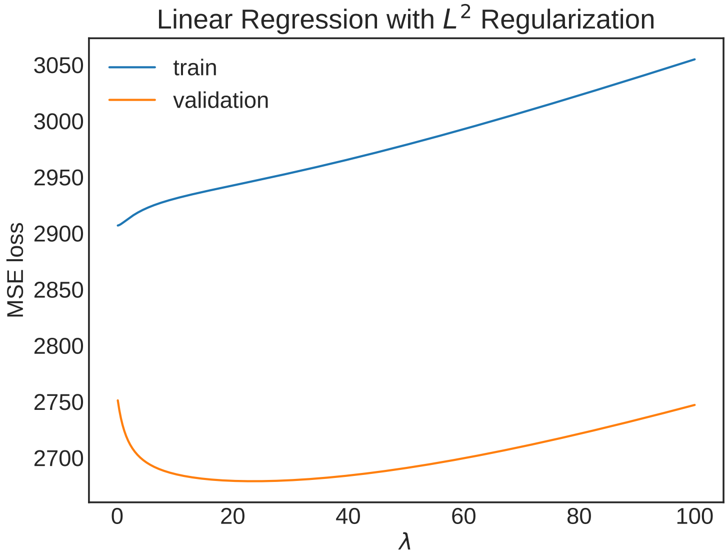
Regularization via Weight Penalties

- There are many forms of regularization
- For parametric models, a common approach is to use weight penalties

$$\text{loss}'(\boldsymbol{x}, y; \boldsymbol{\theta}) = \text{loss}(\boldsymbol{x}, y; \boldsymbol{\theta}) + \lambda\, \Omega(\boldsymbol{\theta})$$

- $\Omega(\boldsymbol{\theta})$ is zero for $\boldsymbol{\theta} = 0$, grows as $\boldsymbol{\theta}$ grows
- Common examples
  - $\Omega(\boldsymbol{\theta}) = \frac{1}{2}||\boldsymbol{\theta}||_2^2$
  - $\Omega(\boldsymbol{\theta}) = ||\boldsymbol{\theta}||_1$
- $\lambda \geq 0$: hyper-parameter that controls strength of regularization
- Regularization biases the model to explore less of the parameter space
- Effectively reduces complexity of learned model (complexity shrinks as λ grows)
- Often bias parameters are excluded from regularization

Regularization Example: Linear Regression



L^2 Regularization

- L^2 regularization is also called weight decay, ridge regression
- Loss function:

$$\text{loss}'(\boldsymbol{x}, y; \boldsymbol{\theta}) = \text{loss}(\boldsymbol{x}, y; \boldsymbol{\theta}) + \frac{\lambda}{2} ||\boldsymbol{\theta}||_2^2$$

- Gradient:

$$\nabla_\theta \text{loss}'(\boldsymbol{x}, y; \boldsymbol{\theta}) = \nabla_\theta \text{loss}(\boldsymbol{x}, y; \boldsymbol{\theta}) + \lambda \, \boldsymbol{\theta}$$

- GD Update:

$$\begin{aligned} \boldsymbol{\theta}^{(t+1)} &= \boldsymbol{\theta}^{(t)} - \alpha \left(\nabla_\theta \text{loss}(\boldsymbol{x}, y; \boldsymbol{\theta}^{(t)}) + \lambda \, \boldsymbol{\theta}^{(t)} \right) && (1) \\ &= (1 - \alpha\lambda)\boldsymbol{\theta}^{(t)} - \alpha \nabla_\theta \text{loss}(\boldsymbol{x}, y; \boldsymbol{\theta}^{(t)}) && (2) \end{aligned}$$

- At each step, shrink the weights by a multiplicative factor
- What is the effect over multiple steps?

L^2 Regularization

- Let's revisit quadratic example:

$$L(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}^T\boldsymbol{w}$$

- $\boldsymbol{w}, \boldsymbol{b} \in \mathbb{R}^d$, and $\boldsymbol{A} \in \mathbb{R}^{d\times d}$. also assume $\boldsymbol{A}$ is symmetric and invertible, meaning $\boldsymbol{A}^{-1}$ exists
- Gradient is:

$$\nabla_{\boldsymbol{w}} L = \boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}$$

- Optimal solution is then $\boldsymbol{w}^* = \boldsymbol{A}^{-1}\boldsymbol{b}$

L^2 Regularization

- Quadratic loss function:

$$L(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}^T\boldsymbol{w} + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$$
$$= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{A}'\boldsymbol{w} - \boldsymbol{b}^T\boldsymbol{w}$$

- Same as old problem, but with $\boldsymbol{A}' = \boldsymbol{A} + \lambda\mathbb{1}$
- Optimal solution: $\boldsymbol{w}^* = (\boldsymbol{A}')^{-1}\boldsymbol{b}$
- Diagonalize:

$$\boldsymbol{A} = \boldsymbol{O}^{-1}\boldsymbol{\Lambda}\boldsymbol{O}\,, \qquad \boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, ..., \lambda_d)\,.$$
$$\boldsymbol{b}' = \boldsymbol{O}\boldsymbol{b}\,, \qquad \boldsymbol{x} = \boldsymbol{O}\boldsymbol{w}\,.$$

- Optimal x_a^* values are shrunk, smaller eigen-directions are shrunk more

$$\boldsymbol{x}^* = O\boldsymbol{w}^* = (\Lambda + \lambda\mathbb{1})^{-1}\,\boldsymbol{b}'\,, \qquad x_a^* = \frac{1}{1 + \lambda/\lambda_a} \times \text{old solution}$$

L^1 Regularization

- Called LASSO in statistics
- Loss function:

$$\text{loss}'(\boldsymbol{x}, y; \boldsymbol{\theta}) = \text{loss}(\boldsymbol{x}, y; \boldsymbol{\theta}) + \lambda \, ||\boldsymbol{\theta}||_1$$

- Gradient:

$$\nabla_\theta \text{loss}'(\boldsymbol{x}, y; \boldsymbol{\theta}) = \nabla_\theta \text{loss}(\boldsymbol{x}, y; \boldsymbol{\theta}) + \lambda \, \text{sign}(\boldsymbol{\theta})$$

- GD Update:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \left(\nabla_\theta \text{loss}(\boldsymbol{x}, y; \boldsymbol{\theta}^{(t)}) + \lambda \, \text{sign}(\boldsymbol{\theta}^{(t)}) \right) \tag{3}$$

- Tends to induce sparsity - small parameters get sent to zero

L^1, L^2 Regularization

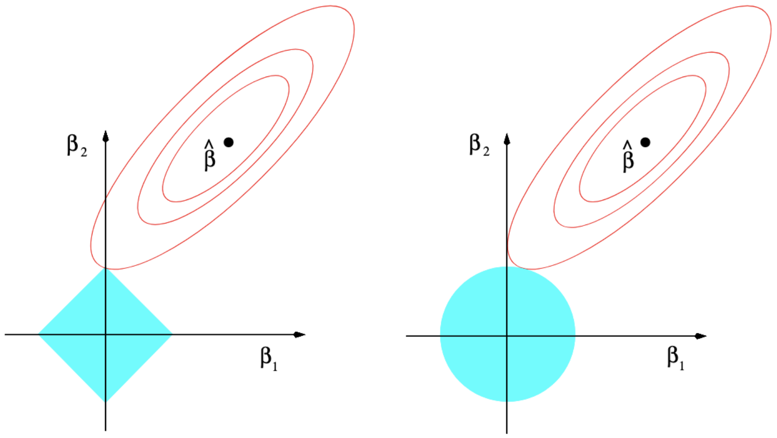


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \le t$ *and* $\beta_1^2 + \beta_2^2 \le t^2$*, respectively, while the red ellipses are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

Model Aggregation and Ensembling

Bootstrapping

- Sometimes data is a scarce resource
- In which case, throwing out 20-25% to do model selection with the validation set, or generalization error estimation with the test set, is quite expensive
- Resampling methods can be used to more efficiently use the limited data you have to estimate various statistical quantities
- Bootstrapping is a popular such method
- Pay attention, because bootstrapping will be important when we consider aggregate/ensemble methods, especially random forests

Bootstrapping

- Start with a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1,\ldots,N}$
- By sampling *with replacement*, generate B new datasets (say, $B = 100$ or so)
- Note: some data points will be duplicated, some will be absent
- Prob that a given observation will be in a given bootstrap sample: $1 - (1 - 1/N)^N \approx 1 - e^{-1} \approx 0.632$
  - single draw: $P(i) = 1/N$
  - single draw $P(\neg i) = 1 - 1/N$
  - $P(i \notin \mathcal{D}_b) = (1 - 1/N)^N$
  - $P(i \in \mathcal{D}_b) = 1 - (1 - 1/N)^N$
  - note that

$$e^x = \lim_{N\to\infty} \left(1 + \frac{x}{N}\right)^N$$

- Train B distinct models, one for each bootstrap dataset $\mathcal{D}_b$
- Use these to estimate error or model performance

Bootstrapping

- Summary:
  - Train B models on B independent bootstrap datasets
  - Evaluate model performance by averaging metric of choice across original dataset
  - Can you spot the problem?

Bootstrapping

- Summary:
  - Train B models on B independent bootstrap datasets
  - Evaluate model performance by averaging metric of choice across original dataset
  - Can you spot the problem?
- Each data point used to evaluate the model performance has a 0.632 probability of being in each bootstrap
- Solution: evaluate metric on the so-called out of bag sample
- When evaluating the contribution from datapoint i, only use bootstrap samples that did not include i

$$\text{boostrap avg}(h(\hat{y}, y)) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{|C^{-i}|} \sum_{b \in C^{-i}} h\left(f^{b}(\boldsymbol{x}_i), y_i\right)$$

Bootstrap Aggregating (Bagging)

- Bootstraps were introduced as a validation method
- Goal is not to improve the model, but to improve our understanding of model performance and/or do model selection
- Should be possible to leverage the basic idea to improve our models
- This is achieved by bootstrap aggregating (bagging)
- Regression: average output across bootstrap samples

$$\hat{f}_{\text{Bag}}(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(\boldsymbol{x})$$

- Classification: select most popular class

$$\hat{y}_{\text{Bag}}(\boldsymbol{x}) = \text{argmax}_k \hat{f}_{\text{Bag}}(\boldsymbol{x}; k)$$

Bootstrap Aggregating (Bagging)

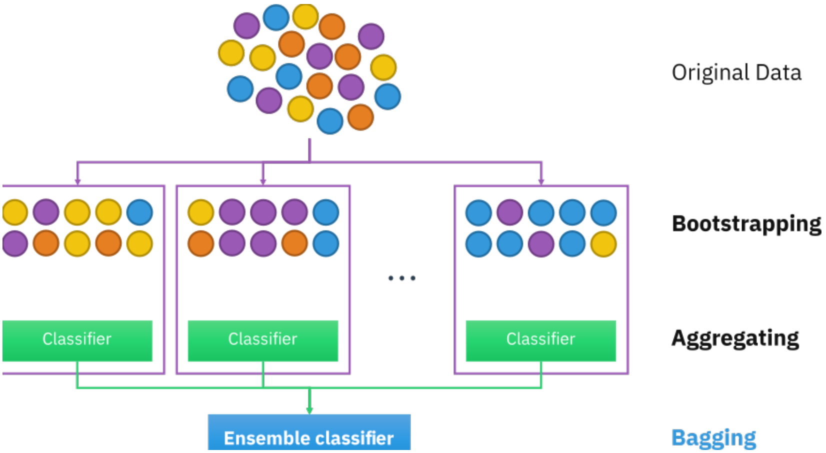


Figure: Source: https://en.wikipedia.org/wiki/Bootstrap_aggregating

Recap

- Introduced general notion of non-parametric models
- As specific examples, introduced kNN and CART
- Introduced validation method, bias-variance trade-off, and weight-norm regularization
- Boostrapping: take an ensemble of models to reduce variance of the evaluation metric
- Bagging: take an ensemble of models to reduce variance of model predictions
- kNN and trees are generally low-bias, high variance models
- They benefit from model aggregation, but you lose interpretability