Machine Learning

Lecture 3 - Evaluating predictive performance and hyperparameter learning

Mattias Villani

Department of Statistics
Stockholm University

Department of Computer and Information Science
Linköping University

mattiasvillani.com @matvil mattiasvillani

Lecture overview

- Loss functions and evaluation metrics
- Evaluating a classifier
- Estimating generalization
- Generalization gap
- Bias-variance trade-off

Loss functions and cost functions

- Parametric model $y|\mathrm{x} \sim f_{\boldsymbol{\theta}}(y|\mathrm{x})$.
- Learn the parameters by minimizing a cost function

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} \overbrace{L\left(\hat{y}(\mathrm{x}_i; \boldsymbol{\theta}), y_i\right)}^{\text{loss function}}}_{\text{cost function } J(\boldsymbol{\theta})}$$

- **Squared error** for regression:

$$L\left(\hat{y}(\mathrm{x}_i; \boldsymbol{\theta}), y_i\right) = \left(\hat{y}(\mathrm{x}_i; \boldsymbol{\theta}) - y_i\right)^2$$

- Squared error loss = negative Gaussian log-likelihood

$$\log p(y_i|\mathrm{x}_i, \boldsymbol{\beta}) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\left(y_i - \mathrm{x}_i^\top \boldsymbol{\beta}\right)^2$$

Loss functions and cost functions

- Learn the parameters by minimizing a cost function

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} \overbrace{L\left(\hat{y}(\mathrm{x}_i;\boldsymbol{\theta}), y_i\right)}^{\text{loss function}}}_{\text{cost function} J(\boldsymbol{\theta})}$$

- **Cross-entropy** for binary classification $y \in \{0, 1\}$

$$L\left(\hat{y}(\mathrm{x}_i;\boldsymbol{\theta}), y_i\right) = -y_i \log \Pr(y_i = 1|\mathrm{x}_i, \theta) - (1 - y_i) \log \Pr(y_i = 0|\mathrm{x}_i, \theta)$$

- Cross-entropy loss = negative Bernoulli log-likelihood

$$\Pr(Y_i = y_i|\mathrm{x}_i, \boldsymbol{\beta}) = \Pr(y_i = 1|\mathrm{x}_i, \boldsymbol{\beta})^{y_i} \Pr(y_i = 0|\mathrm{x}_i, \boldsymbol{\beta})^{1-y_i}$$

where for logistic regression

$$\Pr(y_i = 1|\mathrm{x}_i, \boldsymbol{\beta}) = \frac{\exp(\mathrm{x}_i^T \boldsymbol{\beta})}{1 + \exp(\mathrm{x}_i^T \boldsymbol{\beta})}.$$

- See Bonus slides for a little more on (cross-)entropy.

Detecting fraudulent banknotes

- Dataset with 1372 photographed banknotes. 610 fake.
- Raw data: $400 \times 400 = 160000$ gray-scale pixels.
- The 160000 pixel variables are condensed to four **features**:
  - variance of Wavelet Transformed image
  - skewness of Wavelet Transformed image
  - kurtosis of Wavelet Transformed image
  - entropy of image
- **Deep learning** on raw images (more later!).

Detecting fraudulent banknotes

- 1000 images for training. Predictions on 372 test images.
- Logistic regression

$$\Pr(\text{Fraud} = \text{True}|\text{features}) = \frac{\exp(\mathbf{x}^\top \beta_c)}{\sum_{j=1}^{C} \exp(\mathbf{x}^\top \beta_j)}$$

- **Decision**: signal fraud if $\Pr(\text{Fraud} = \text{True}|\text{Features}) > 0.5$.
- **Confusion matrix**

		Truth: No fraud	Truth: Fraud
Decision	No Fraud	208	1
	Fraud	3	160

Evaluating a classifier - confusion matrix

- **Confusion** matrix:

		Truth: Positive	Truth: Negative
Decision	Positive	tp	fp
	Negative	fn	tn

- tp = true positive, fp = false positive
  fn = false negative, tn = true negative.
- Example:

		Truth: No Fraud	Truth: Fraud
Decision	No Fraud	208	1
	Fraud	3	160

Evaluating a classifier - Accuracy

- **Accuracy** is the proportion of correctly classified items

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fn + fp}$$

		Truth	
		Positive	Negative
Decision	Positive	**tp**	fp
	Negative	fn	**tn**

- Note: evaluation criteria (e.g. accuracy) need not be the same as the training loss function (e.g. cross-entropy).

Evaluating a classifier - Precision

- **Precision** is the proportion of truly positive items among those signaled as positive:

$$\text{Precision} = \frac{tp}{tp + fp}$$

		Truth	
		Positive	Negative
Decision	Positive	tp	fp
	Negative	fn	tn

- High precision:
  - trustworthy positives
  - people pointed out as fraudulent are almost always frauds.

Evaluating a classifier - Recall

- **Recall** is the proportion of signaled positive items among those that are truly positive:

$$\text{Recall} = \frac{tp}{tp + fn}$$

		Truth	
		Positive	Negative
Decision	Positive	tp	fp
	Negative	fn	tn

- High recall:
  - will find the positive items.
  - frauds will be caught.
- Recall is also called **True Positive Rate** (**TPR**) or **sensitivity**.
- There is a trade-off between Precision and Recall.

Evaluating a classifier - False Positive Rate

- **False Positive Rate** (**FPR**) is the proportion of signaled positive items among those that are truly negative:

$$\text{FPR} = \frac{fp}{fp + tn}$$

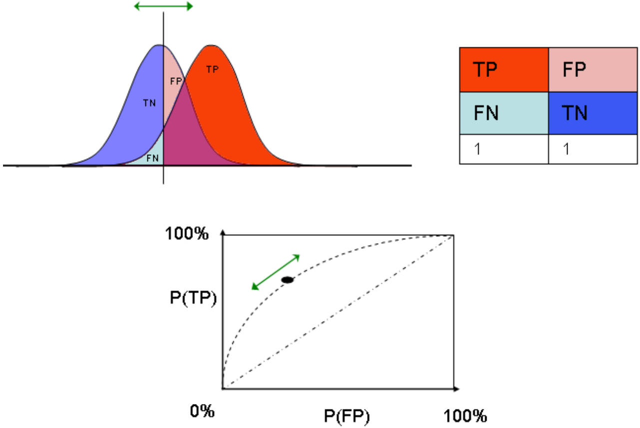
		Truth	
		Positive	Negative
Decision	Positive	tp	fp
	Negative	fn	tn

- Low FPR:
  - will very rarely signal a positive for a negative item.
  - people will not be falsely accused of fraud.

Evaluating a classifier - ROC curve

- Precision and recall depends on the **decision threshold.**
- $\Pr(\text{Spam}|\text{text in an email}) = 0.9$. Do we send it to the spam-box?
- Is $\Pr(\text{Fraud}|\text{features}) > 0.5$ a good **decision threshold**?
- **Optimal decisions** depend on the consequences. **Decision theory**.
- **ROC-curve**: Receiver Operating Characteristic.
- ROC: Plots the true positive rate (TPR) against the false positive rate (FPR) **at various thresholds**.
- **AUC** = Area Under Curve. Area under the ROC curve.

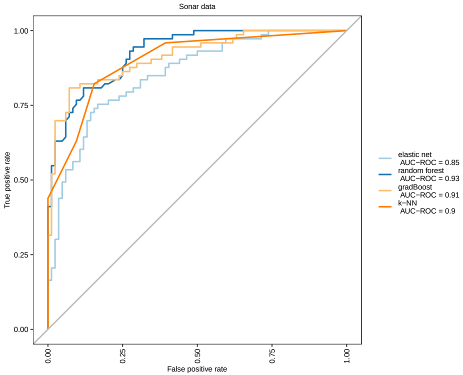
Evaluating a classifier - ROC



From Wikipedia.

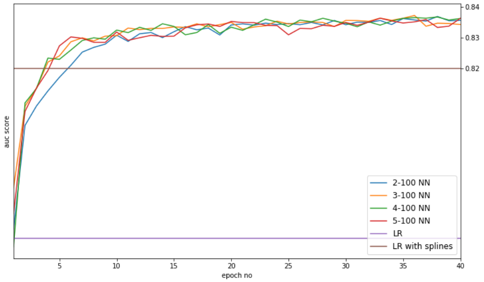
ROC - Sonar data

- Sonar data
  - Binary response (metal/rock)
  - 61 covariates (energy within a frequency band from sound impulse).



Predicting firm bankruptcy

- Comparing test AUC for
  - logistic reg (LR)
  - logistic reg with additive splines
  - neural networks (NN) with 2-5 layers, each with 100 nodes.



- epochs is number of iterations of the stochastic gradient decent optimization for the neural network, see Lecture 5.

Error function and generalization performance

- **Error function** compares prediction $\hat{y}(\mathrm{x}_\star)$ to actual y
  - **Squared error** for regression:

$$E(\hat{y}, y) = (\hat{y} - y)^2$$

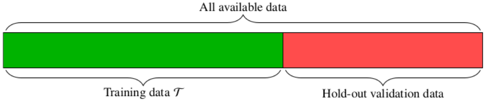
  - **Misclassification** for classification:

$$E(\hat{y}, y) = \mathbb{I}\{\hat{y} \neq y\} = \begin{cases} 0 & \text{if } \hat{y} = y \\ 1 & \text{if } \hat{y} \neq y \end{cases}$$

- Loss function $L(\hat{y}, y)$ need not be the same as error function $E(\hat{y}, y)$ used for performance evaluation.
  - k-NN does not have an explicit loss function.
  - Misclassification rate is discontinuous in $\boldsymbol{\theta}$. Hard to optimize.
- For all data: **mean squared error** and **misclassification rate**.
- **Generalization performance**: method's average performance on data drawn from a distribution $p(\mathrm{x}, y)$.

Generalization and Training-Test splits

- How estimate generalization performance? Training-Test split.
- Partition the data in two parts:
  - **Training data** $\mathcal{T} = \{y_i, x_i\}_{i=1}^{n}$ to estimate parameters.
  - **Test data** (hold-out) to estimate generalization.



- How to make the split:
  - Randomly
  - Systematic (time series: most recent data in test set).
- `Caret` package in R. 75% training, 25% test:

```
library(caret)
library(mlbench)
data(Sonar)
inTrain <- createDataPartition(y = Sonar$Class, p = .75, list = FALSE)
training <- Sonar[ inTrain,]
testing  <- Sonar[-inTrain,]
```

Generalization and Training-Test splits

- **Training data** $\mathcal{T} = \{y_i, \mathrm{x}_i\}_{i=1}^n$ gives estimated/trained model for prediction: $\hat{y}(\mathrm{x}_\star; \mathcal{T})$.
- **Expected new data error**

$$E_{\text{new}} \equiv \mathbb{E}_\star \left[E\left(\hat{y}(\mathrm{x}_\star; \mathcal{T}), y_\star\right) \right] = \int E\left(\hat{y}(\mathrm{x}_\star; \mathcal{T}), y_\star\right) p(\mathrm{x}_\star, y_\star) d\mathrm{x}_\star dy_\star$$

  - $E\left(\hat{y}(\mathrm{x}_\star; \mathcal{T}), y_\star\right)$ is prediction error when training on **fixed** $\mathcal{T}$.
  - $\mathbb{E}_\star$ is the expectation wrt **test data** from $(\mathrm{x}_\star, y_\star) \sim p(\mathrm{x}, y)$.
- E_{new} measures how a trained model **generalizes** to new data from $p(\mathrm{x}, y)$.
- Goal of ML: minimize E_{new}. But E_{new} is unknown!

Training error

- The training error is computed from $\mathcal{T} = \{y_i, \mathrm{x}_i\}_{i=1}^{n}$

$$E_{\text{train}} \equiv \frac{1}{n}\sum_{i=1}^{n} E\left(\hat{y}(\mathrm{x}_i; \mathcal{T}), y_i\right)$$

- Typically $E_{\text{train}} < E_{\text{new}}$, due to **overfitting** the training data.
- Error on **hold-out validation data** $\{y_j', \mathrm{x}_j'\}_{j=1}^{n_v}$:

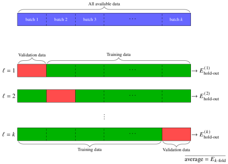
$$E_{\text{hold-out}} \equiv \frac{1}{n_v}\sum_{j=1}^{n_v} E\left(\hat{y}(\mathrm{x}_j'; \mathcal{T}), y_j'\right)$$

- If $(y_i', \mathrm{x}_i') \sim p(\mathrm{x}, y)$, $E_{\text{hold-out}}$ is an unbiased estimate of E_{new}.
- Problem:
  - small variance of $E_{\text{hold-out}}$ requires large n_v ...
  - but large n_v means less training data ...
  - less training data means larger E_{new}. ☹

k-fold cross-validation

- Split the data in k folds or batches: $B_1, \ldots, B_k$.
- For each batch $\ell = 1, 2, \ldots, k$:
  - train the model on $B_{-\ell} \equiv \{B_1, \ldots, B_{\ell-1}, B_{\ell+1}, \ldots, B_k\}$.
  - predict the data in B_ℓ and compute error function $E^{(\ell)}_{\text{hold}-\text{out}}$
- Compute the crossvalidation estimate of E_{new}

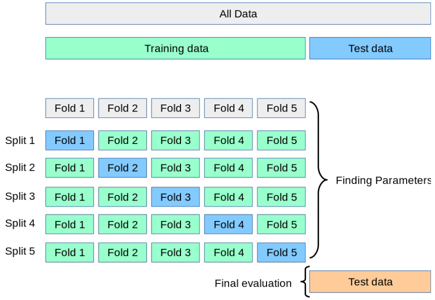
$$E_{\text{hold}-\text{out}} = \frac{1}{k}\sum_{\ell=1}^{k} E^{(\ell)}_{\text{hold}-\text{out}}$$



- $E^{(\ell)}_{\text{hold}-\text{out}}$ unbiased for E_{new} with $\mathcal{T} = B_{-\ell}$ for each ℓ.
- $E_{\text{hold}-\text{out}}$ (slightly) biased for E_{new} with $\mathcal{T} = \{\text{all data}\}$.
- **Leave-one-out** cross-validation: $k = n$. Requires n re-fits.

Cross-validation for hyperparameter learning

- Cross-validation is often used to **estimate hyperparameters**:
  - number of neighbors, k, in k-NN
  - tree depth in regression trees
- k-NN: $\hat{k}_{\text{cv}} = \arg\min_k E_{\text{hold-out}}(k)$.
- $E_{\text{hold-out}}(\hat{k}_{\text{cv}})$ typically underestimates E_{new}, because we are optimizing over k when finding $\hat{k}_{\text{cv}}$.
- Solution: **set aside clean test data** to estimate E_{new}.



Cross-validation learning of Lasso shrinkage

- **One-standard deviation rule** for CV-estimation: the least complex model within one-standard error of the best model is chosen.

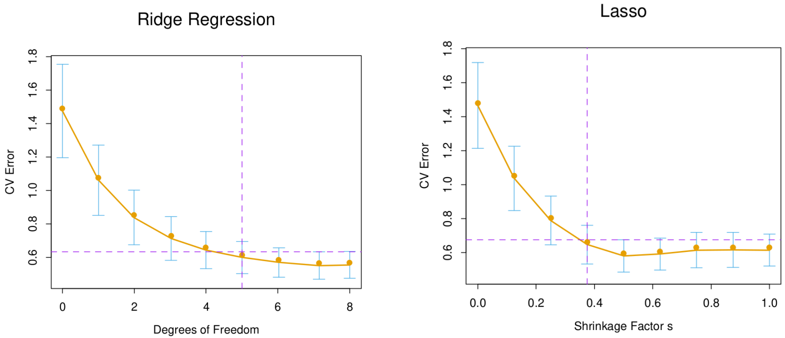


Figure from Hastie et al (2009).

Generalization gap

- E_{new} depends on a specific $\mathcal{T}$. Better notation: $E_{\text{new}}(\mathcal{T})$.
- **Training-averaged generalization error**:

$$\bar{E}_{\text{new}} \equiv \mathbb{E}_{\mathcal{T}}\left[E_{\text{new}}(\mathcal{T})\right]$$

- $\bar{E}_{\text{new}}$ is average E_{new} over all possible training data of size n.
- Training-averaged training error:

$$\bar{E}_{\text{train}} \equiv \mathbb{E}_{\mathcal{T}}\left[E_{\text{train}}(\mathcal{T})\right]$$

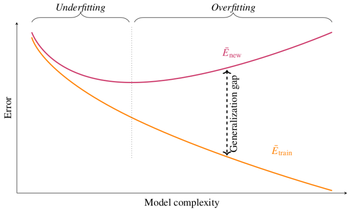
- **Generalization gap**

$$\text{generalization gap} = \bar{E}_{\text{new}} - \bar{E}_{\text{train}}$$

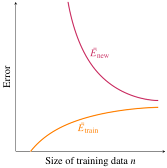
- Training error-generalization gap decomposition

$$\bar{E}_{\text{new}} = \bar{E}_{\text{train}} + \text{generalization gap}$$

Generalization gap



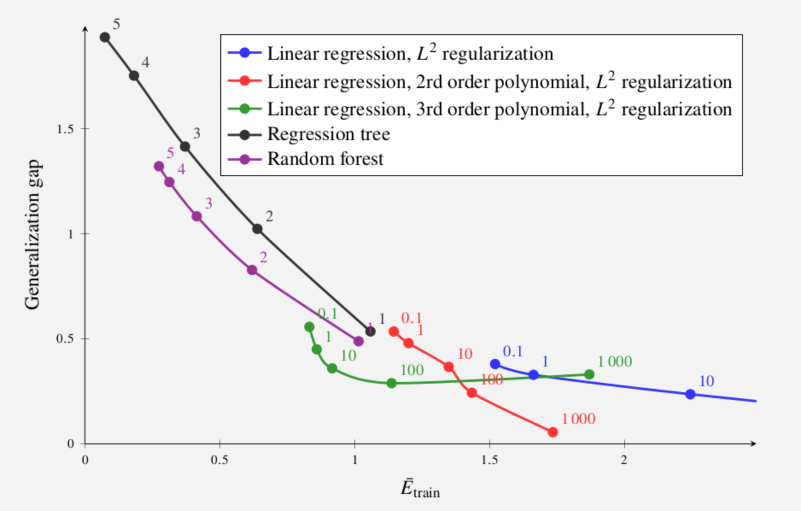
(a) Simple model



(b) Complex model

- If $E_{\text{hold-out}} \approx E_{\text{train}}$: Underfitting? Try increasing complexity.
- If $E_{\text{train}} \approx 0$ and $E_{\text{hold-out}}$ sizeable: Overfitting? Try decreasing complexity.

Generalization gap - five models example



Bias and Variance

- True relationship

$$y = f_0(\mathrm{x}) + \varepsilon, \quad \mathbb{V}(\varepsilon) = \sigma^2.$$

- Estimated model $\hat{y}(\mathrm{x}; \mathcal{T})$ (linear regression $\hat{y}(\mathrm{x}; \mathcal{T}) = \mathrm{x}^T \hat{\beta}$).
- Average trained model

$$\bar{f}(\mathrm{x}) \equiv \mathbb{E}_{\mathcal{T}}\left[\hat{y}(\mathrm{x}; \mathcal{T})\right]$$

- **Bias**

$$\mathrm{Bias}\left(\hat{y}(\mathrm{x}; \mathcal{T})\right) = \bar{f}(\mathrm{x}) - f_0(\mathrm{x})$$

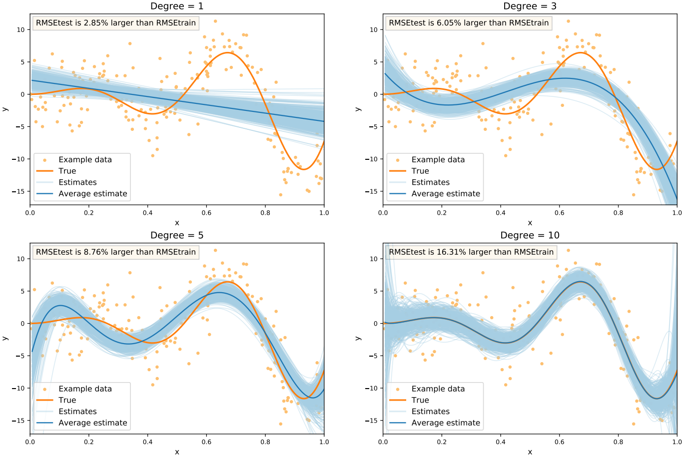
- **Variance**

$$\mathbb{V}_{\mathcal{T}}\left[\hat{y}(\mathrm{x}; \mathcal{T})\right] = \mathbb{E}_{\mathcal{T}}\left[\left(\hat{y}(\mathrm{x}; \mathcal{T}) - \bar{f}(\mathrm{x})\right)^2\right].$$

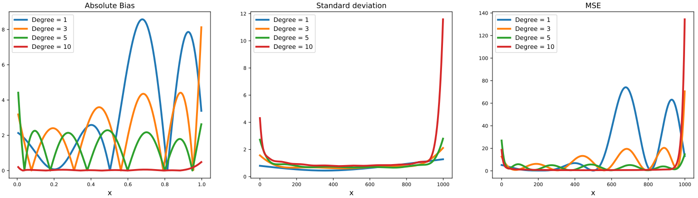
- **Mean Squared Error (MSE)**

$$\mathrm{MSE}\left(\hat{y}(\mathrm{x}; \mathcal{T})\right) = \mathbb{E}_{\mathcal{T}}\left[\left(\hat{y}(\mathrm{x}; \mathcal{T}) - f_0(\mathrm{x})\right)^2\right] = \mathbb{V}_{\mathcal{T}}\left[\hat{y}(\mathrm{x}; \mathcal{T})\right] + \mathrm{Bias}\left(\hat{y}(\mathrm{x}; \mathcal{T})\right)^2$$

Bias-Variance trade-off polynomials



Bias-Variance trade-off polynomials



Bias-Variance trade-off

- Bias-variance decomposition of

$$E_{\text{new}} \equiv \mathbb{E}_{\mathcal{T}}\left[\mathbb{E}_\star\left(\hat{y}(\mathrm{x}_\star;\mathcal{T}) - y_\star\right)^2\right]$$

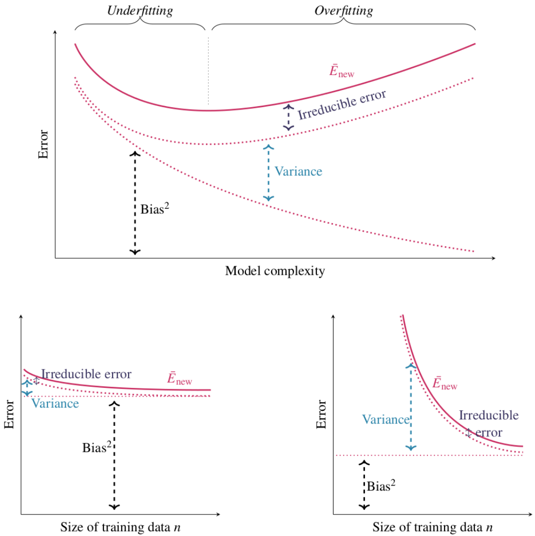
- Change order of the expectations and insert true model:

$$\bar{E}_{\text{new}} = \mathbb{E}_{\mathcal{T}}\left[\mathbb{E}_\star\left(\hat{y}(\mathrm{x}_\star;\mathcal{T}) - y_\star\right)^2\right] = \mathbb{E}_\star\left[\mathbb{E}_{\mathcal{T}}\left(\hat{y}(\mathrm{x}_\star;\mathcal{T}) - f_0(\mathrm{x}_\star) - \varepsilon\right)^2\right]$$

- Add and subtract $\bar{f}(\mathrm{x}_\star)$ and expand the square to get ...
- **Bias-variance decomposition**

$$\bar{E}_{\text{new}} = \underbrace{\mathbb{E}_\star\left[\left(\bar{f}(\mathrm{x}_\star) - f_0(\mathrm{x}_\star)\right)^2\right]}_{\text{Bias}^2} + \underbrace{\mathbb{E}_\star\left[\mathbb{E}_{\mathcal{T}}\left(\hat{y}(\mathrm{x}_\star;\mathcal{T}) - \bar{f}(\mathrm{x}_\star)\right)^2\right]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Irreducible error}}$$

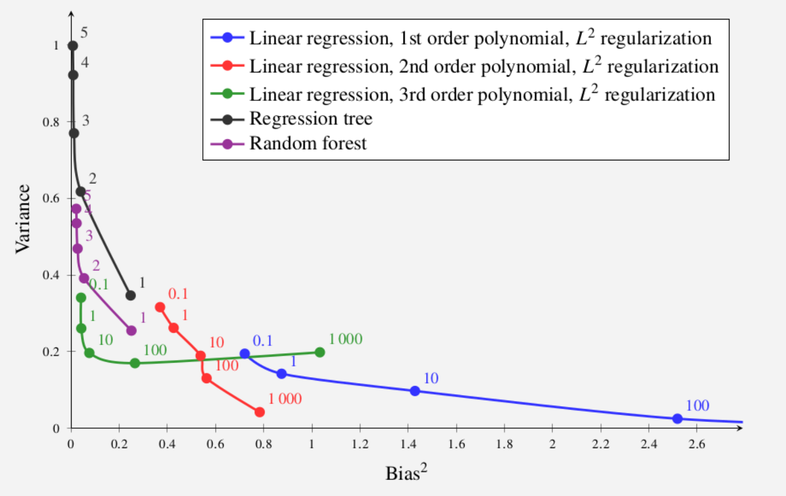
Bias-Variance trade-off



(a) Simple model

(b) Complex model

Bias-variance - five models example



Bias-variance - five models example

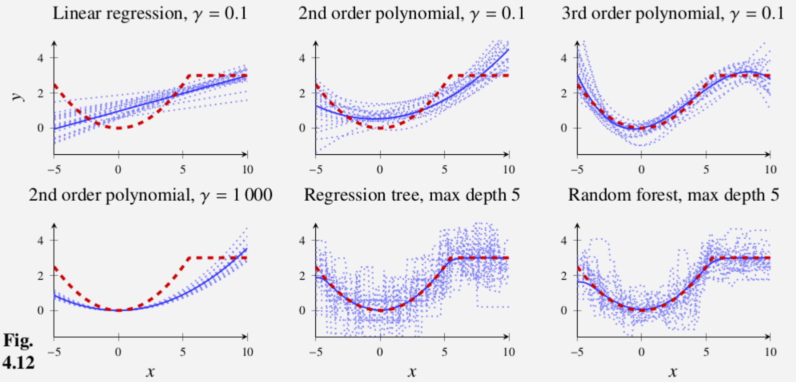
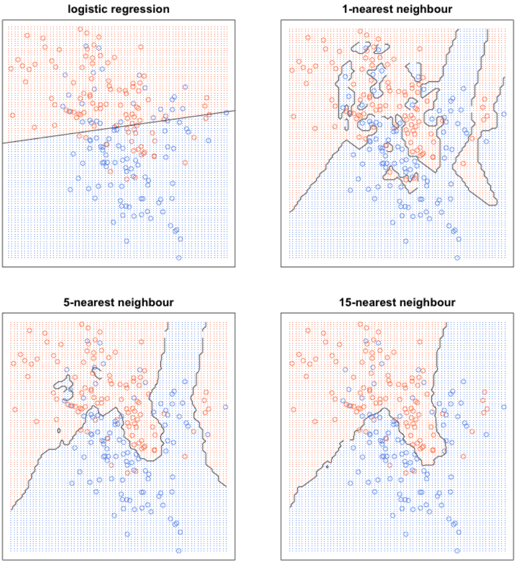
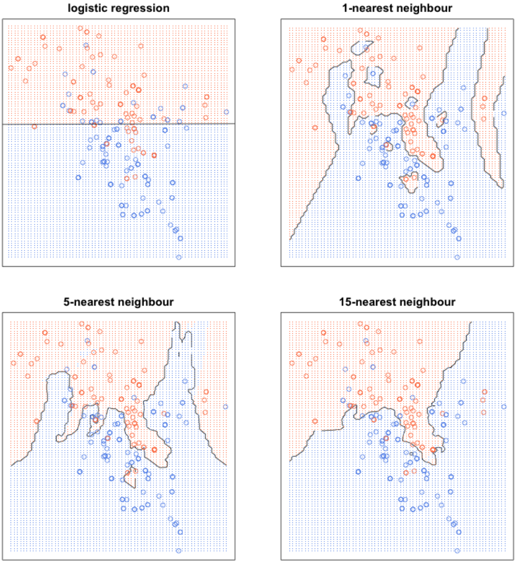


Fig. 4.12

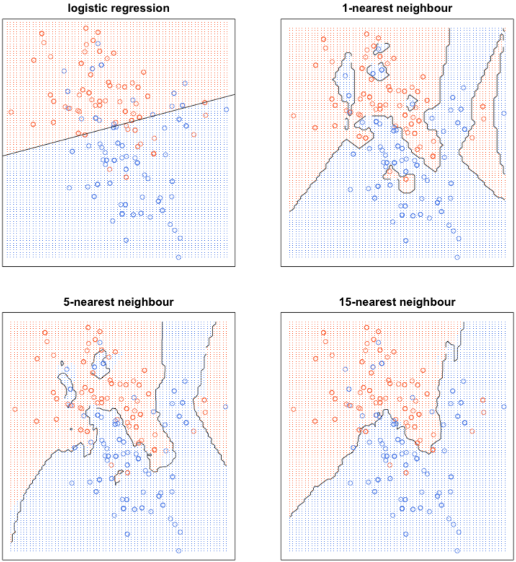
Bias-Variance trade-off classification



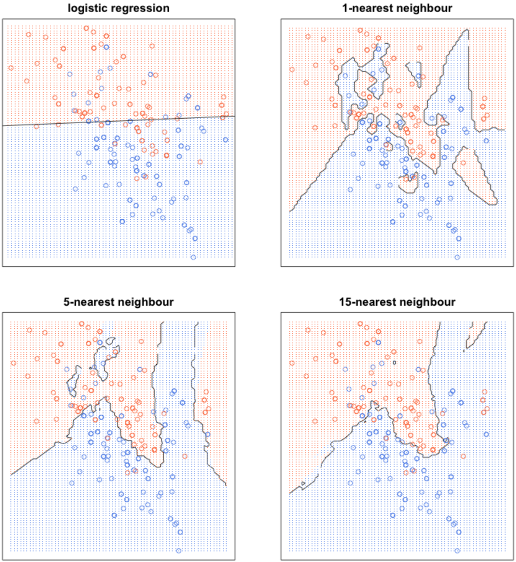
Bootstrap sample no 1



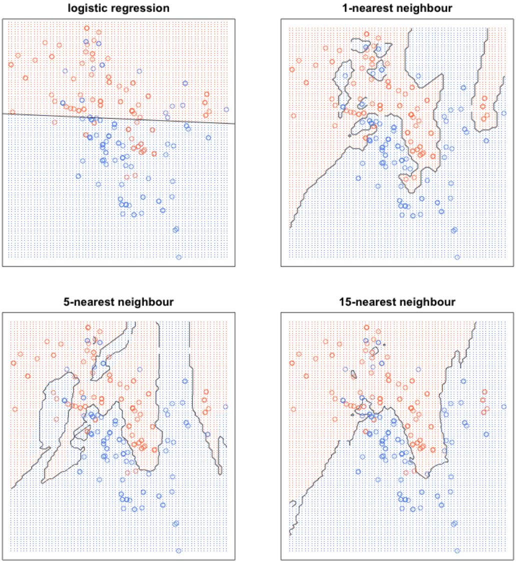
Bootstrap sample no 2



Bootstrap sample no 3



Bootstrap sample no 4



Bootstrap sample no 5