Model Evaluation, Selection and Ensembles

Machine Learning

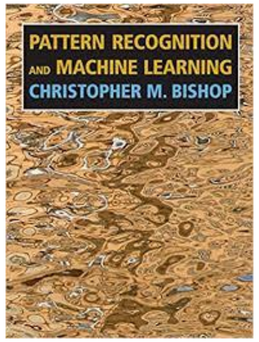
Daniele Loiacono

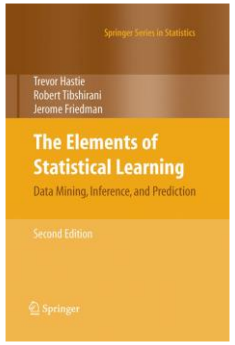
POLITECNICO
MILANO 1863

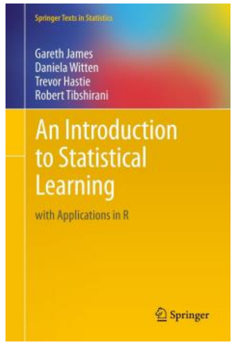
Outline and References

- Outline
  - Bias-Variance Tradeoff (PRML 3.2, ESL 7.3)
  - Model Assessment in Practice (ISL 5.1 and 6.1.3)
  - Feature Selection (ESL 3.3)
  - Dimensionality Reduction (PRML 12.1)
  - Bagging and Boosting (PRML 14.2, 14.3)

- References
  - This slides are based on material of prof. Marcello Restelli
  - Pattern Recognition and Machine Learning, Bishop [PRML]
  - Elements of Statistical Learning, Hastie et al. [ESL]
  - Introduction to Statistical Learning, James et al. [ISL]




Springer Series in Statistics
Trevor Hastie
Robert Tibshirani
Jerome Friedman
The Elements of Statistical Learning
Data Mining, Inference, and Prediction
Second Edition
Springer



Bias-Variance

How to evaluate a model?

- Given a dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$ and a model $\widehat{t_i} = y(\mathbf{x}_i)$
- How do we choose a model y?
- Is the loss function L computed on $\mathcal{D}$ a good way to evaluate a model?
  - For example, for regression could we use $RSS_y = \sum_{\{\mathbf{x}_i, t_i\} \in \mathcal{D}} (t_i - y(\mathbf{x}_i))^2$
- No! This is the performance measure we want to compute:

$$\mathbb{E}[L] = \int \int L(t, y(\mathbf{x})) p(\mathbf{x}, t) \mathrm{d}\mathbf{x} \mathrm{d}t$$

  - E.g., when using a squared loss function for regression:

$$\mathbb{E}[L] = \int \int (t - y(\mathbf{x}))^2 p(\mathbf{x}, t) \mathrm{d}\mathbf{x} \mathrm{d}t$$

We usually don't know $p(\mathbf{x}, t)$!
So what should we do?

Bias-Variance Decomposition

- The Bias-Variance is a framework to analyze the performance of models
- Definitions and assumptions
  - Data: $t_i = f(\mathbf{x}_i) + \varepsilon$, with $\mathbb{E}[\varepsilon] = 0$ and $Var[\varepsilon] = \sigma^2$
  - Model: $\widehat{t_i} = y(\mathbf{x}_i)$ learned from a sampled dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$
  - Performance: $\mathbb{E}\left[\left(t - y(\mathbf{x})\right)^2\right]$ (expected square error)
- Thus we can decompose the **expectected square error** as:

$$\mathbb{E}[(t - y(\mathbf{x}))^2] =$$

Bias-Variance Decomposition

- The Bias-Variance is a framework to analyze the performance of models
- Definitions and assumptions
  - Data: $t_i = f(\mathbf{x}_i) + \varepsilon$, with $\mathbb{E}[\varepsilon] = 0$ and $Var[\varepsilon] = \sigma^2$
  - Model: $\widehat{t_i} = y(\mathbf{x}_i)$ learned from a sampled dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$
  - Performance: $\mathbb{E}\left[\left(t - y(\mathbf{x})\right)^2\right]$ (expected square error)
- Thus we can decompose the **expectected square error** as:

$$\mathbb{E}[(t - y(\mathbf{x}))^2] = \mathbb{E}[t^2 + y(\mathbf{x})^2 - 2ty(\mathbf{x})]$$

Bias-Variance Decomposition

- The Bias-Variance is a framework to analyze the performance of models
- Definitions and assumptions
  - Data: $t_i = f(\mathbf{x}_i) + \varepsilon$, with $\mathbb{E}[\varepsilon] = 0$ and $Var[\varepsilon] = \sigma^2$
  - Model: $\widehat{t_i} = y(\mathbf{x}_i)$ learned from a sampled dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$
  - Performance: $\mathbb{E}\left[(t - y(\mathbf{x}))^2\right]$ (expected square error)
- Thus we can decompose the **expectected square error** as:

$$\begin{aligned}\mathbb{E}[(t-y(\mathbf{x}))^2] &= \mathbb{E}[t^2 + y(\mathbf{x})^2 - 2ty(\mathbf{x})] \\ &= \mathbb{E}[t^2] + \mathbb{E}[y(\mathbf{x})^2] - \mathbb{E}[2ty(\mathbf{x})]\end{aligned}$$

Bias-Variance Decomposition

- The Bias-Variance is a framework to analyze the performance of models
- Definitions and assumptions
  - Data: $t_i = f(\mathbf{x}_i) + \varepsilon$, with $\mathbb{E}[\varepsilon] = 0$ and $Var[\varepsilon] = \sigma^2$
  - Model: $\widehat{t_i} = y(\mathbf{x}_i)$ learned from a sampled dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$
  - Performance: $\mathbb{E}\left[\left(t - y(\mathbf{x})\right)^2\right]$ (expected square error)
- Thus we can decompose the **expectected square error** as:

$$\begin{aligned}\mathbb{E}[(t - y(\mathbf{x}))^2] &= \mathbb{E}[t^2 + y(\mathbf{x})^2 - 2ty(\mathbf{x})] \\ &= \mathbb{E}[t^2] + \mathbb{E}[y(\mathbf{x})^2] - \mathbb{E}[2ty(\mathbf{x})] \\ &= \mathbb{E}[t^2] \pm \mathbb{E}[t]^2 + \mathbb{E}[y(\mathbf{x})^2] \pm \mathbb{E}[y(\mathbf{x})]^2 - 2f(\mathbf{x})\mathbb{E}[y(\mathbf{x})]\end{aligned}$$

Bias-Variance Decomposition

- The Bias-Variance is a framework to analyze the performance of models
- Definitions and assumptions
  - Data: $t_i = f(\mathbf{x}_i) + \varepsilon$, with $\mathbb{E}[\varepsilon] = 0$ and $Var[\varepsilon] = \sigma^2$
  - Model: $\widehat{t_i} = y(\mathbf{x}_i)$ learned from a sampled dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$
  - Performance: $\mathbb{E}\left[\left(t - y(\mathbf{x})\right)^2\right]$ (expected square error)
- Thus we can decompose the **expectected square error** as:

$$
\begin{aligned}
\mathbb{E}[(t - y(\mathbf{x}))^2] &= \mathbb{E}[t^2 + y(\mathbf{x})^2 - 2ty(\mathbf{x})] \\
&= \mathbb{E}[t^2] + \mathbb{E}[y(\mathbf{x})^2] - \mathbb{E}[2ty(\mathbf{x})] \\
&= \mathbb{E}[t^2] \pm \mathbb{E}[t]^2 + \mathbb{E}[y(\mathbf{x})^2] \pm \mathbb{E}[y(\mathbf{x})]^2 - 2f(\mathbf{x})\mathbb{E}[y(\mathbf{x})] \\
&= Var[t] + \mathbb{E}[t]^2 + Var[y(\mathbf{x})] + \mathbb{E}[y(\mathbf{x})]^2 - 2f(\mathbf{x})\mathbb{E}[y(\mathbf{x})]
\end{aligned}
$$

Bias-Variance Decomposition

- The Bias-Variance is a framework to analyze the performance of models
- Definitions and assumptions
  - Data: $t_i = f(\mathbf{x}_i) + \varepsilon$, with $\mathbb{E}[\varepsilon] = 0$ and $Var[\varepsilon] = \sigma^2$
  - Model: $\widehat{t_i} = y(\mathbf{x}_i)$ learned from a sampled dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$
  - Performance: $\mathbb{E}\left[\left(t - y(\mathbf{x})\right)^2\right]$ (expected square error)
- Thus we can decompose the **expectected square error** as:

$$\begin{aligned}
\mathbb{E}[(t - y(\mathbf{x}))^2] &= \mathbb{E}[t^2 + y(\mathbf{x})^2 - 2ty(\mathbf{x})] \\
&= \mathbb{E}[t^2] + \mathbb{E}[y(\mathbf{x})^2] - \mathbb{E}[2ty(\mathbf{x})] \\
&= \mathbb{E}[t^2] \pm \mathbb{E}[t]^2 + \mathbb{E}[y(\mathbf{x})^2] \pm \mathbb{E}[y(\mathbf{x})]^2 - 2f(\mathbf{x})\mathbb{E}[y(\mathbf{x})] \\
&= Var[t] + \mathbb{E}[t]^2 + Var[y(\mathbf{x})] + \mathbb{E}[y(\mathbf{x})]^2 - 2f(\mathbf{x})\mathbb{E}[y(\mathbf{x})] \\
&= Var[t] + Var[y(\mathbf{x})] + (f(\mathbf{x}) - \mathbb{E}[y(\mathbf{x})])^2
\end{aligned}$$

Bias-Variance Decomposition

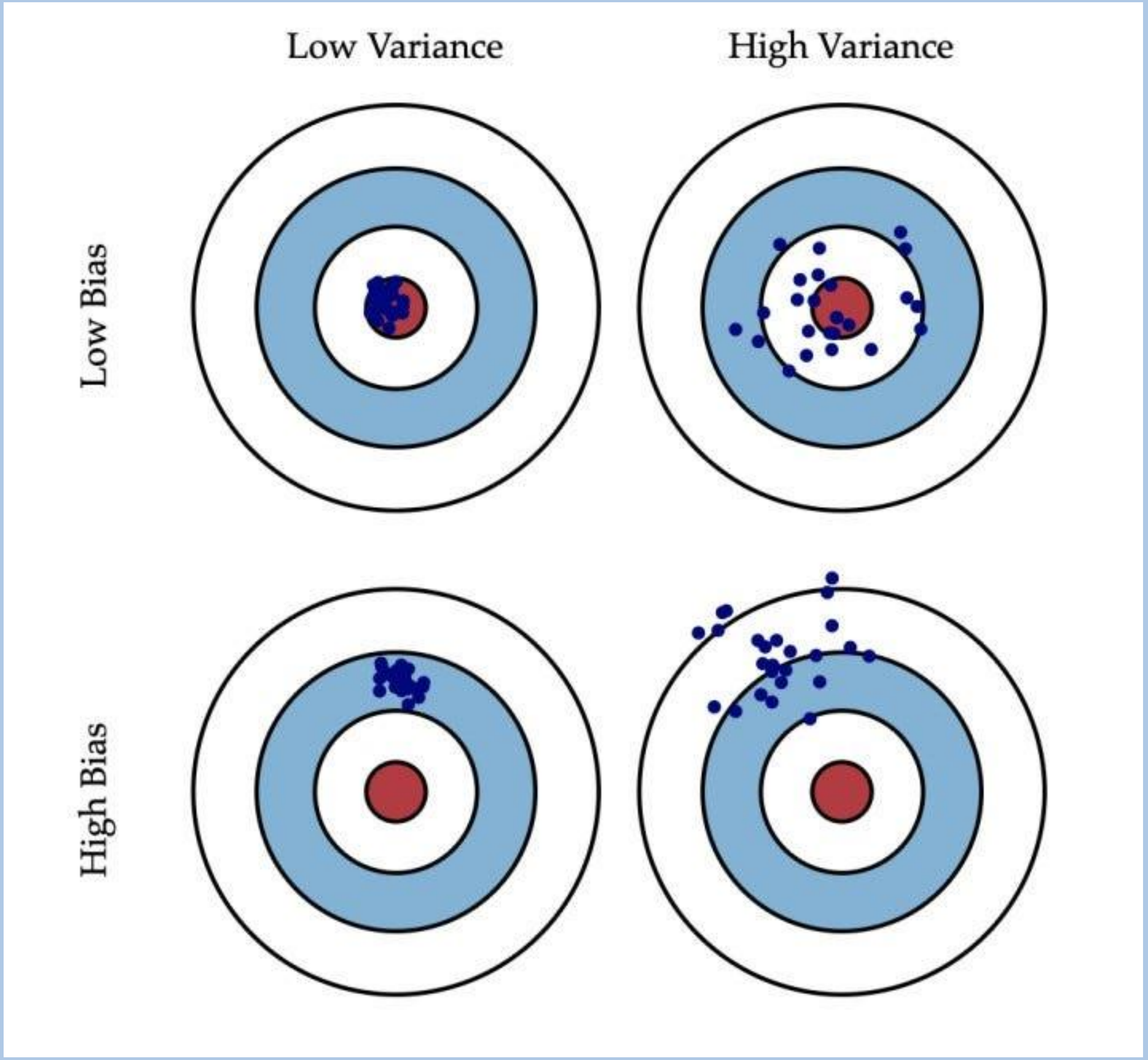
- The Bias-Variance is a framework to analyze the performance of models
- Definitions and assumptions
  - Data: $t_i = f(\mathbf{x}_i) + \varepsilon$, with $\mathbb{E}[\varepsilon] = 0$ and $Var[\varepsilon] = \sigma^2$
  - Model: $\widehat{t_i} = y(\mathbf{x}_i)$ learned from a sampled dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$
  - Performance: $\mathbb{E}\left[\left(t - y(\mathbf{x})\right)^2\right]$ (expected square error)
- Thus we can decompose the **expectected square error** as:

$$\begin{aligned}
\mathbb{E}[(t - y(\mathbf{x}))^2] &= \mathbb{E}[t^2 + y(\mathbf{x})^2 - 2ty(\mathbf{x})] \\
&= \mathbb{E}[t^2] + \mathbb{E}[y(\mathbf{x})^2] - \mathbb{E}[2ty(\mathbf{x})] \\
&= \mathbb{E}[t^2] \pm \mathbb{E}[t]^2 + \mathbb{E}[y(\mathbf{x})^2] \pm \mathbb{E}[y(\mathbf{x})]^2 - 2f(\mathbf{x})\mathbb{E}[y(\mathbf{x})] \\
&= Var[t] + \mathbb{E}[t]^2 + Var[y(\mathbf{x})] + \mathbb{E}[y(\mathbf{x})]^2 - 2f(\mathbf{x})\mathbb{E}[y(\mathbf{x})] \\
&= Var[t] + Var[y(\mathbf{x})] + (f(\mathbf{x}) - \mathbb{E}[y(\mathbf{x})])^2 \\
&= Var[t] + Var[y(\mathbf{x})] + \mathbb{E}[f(\mathbf{x}) - y(\mathbf{x})]^2
\end{aligned}$$

Bias-Variance Decomposition

- The Bias-Variance is a framework to analyze the performance of models
- Definitions and assumptions
  - Data: $t_i = f(\mathbf{x}_i) + \varepsilon$, with $\mathbb{E}[\varepsilon] = 0$ and $Var[\varepsilon] = \sigma^2$
  - Model: $\hat{t_i} = y(\mathbf{x}_i)$ learned from a sampled dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$
  - Performance: $\mathbb{E}\left[\left(t - y(\mathbf{x})\right)^2\right]$ (expected square error)
- Thus we can decompose the **expectected square error** as:

$$\begin{aligned}
\mathbb{E}[(t - y(\mathbf{x}))^2] &= \mathbb{E}[t^2 + y(\mathbf{x})^2 - 2ty(\mathbf{x})] \\
&= \mathbb{E}[t^2] + \mathbb{E}[y(\mathbf{x})^2] - \mathbb{E}[2ty(\mathbf{x})] \\
&= \mathbb{E}[t^2] \pm \mathbb{E}[t]^2 + \mathbb{E}[y(\mathbf{x})^2] \pm \mathbb{E}[y(\mathbf{x})]^2 - 2f(\mathbf{x})\mathbb{E}[y(\mathbf{x})] \\
&= Var[t] + \mathbb{E}[t]^2 + Var[y(\mathbf{x})] + \mathbb{E}[y(\mathbf{x})]^2 - 2f(\mathbf{x})\mathbb{E}[y(\mathbf{x})] \\
&= Var[t] + Var[y(\mathbf{x})] + (f(\mathbf{x}) - \mathbb{E}[y(\mathbf{x})])^2 \\
&= \underbrace{Var[t]}_{\sigma^2} + \underbrace{Var[y(\mathbf{x})]}_{\text{Variance}} + \underbrace{\mathbb{E}[f(\mathbf{x}) - y(\mathbf{x})]^2}_{\text{Bias}^2}
\end{aligned}$$


Low Variance
High Variance
Low Bias
High Bias

Bias-Variance Decomposition (2)

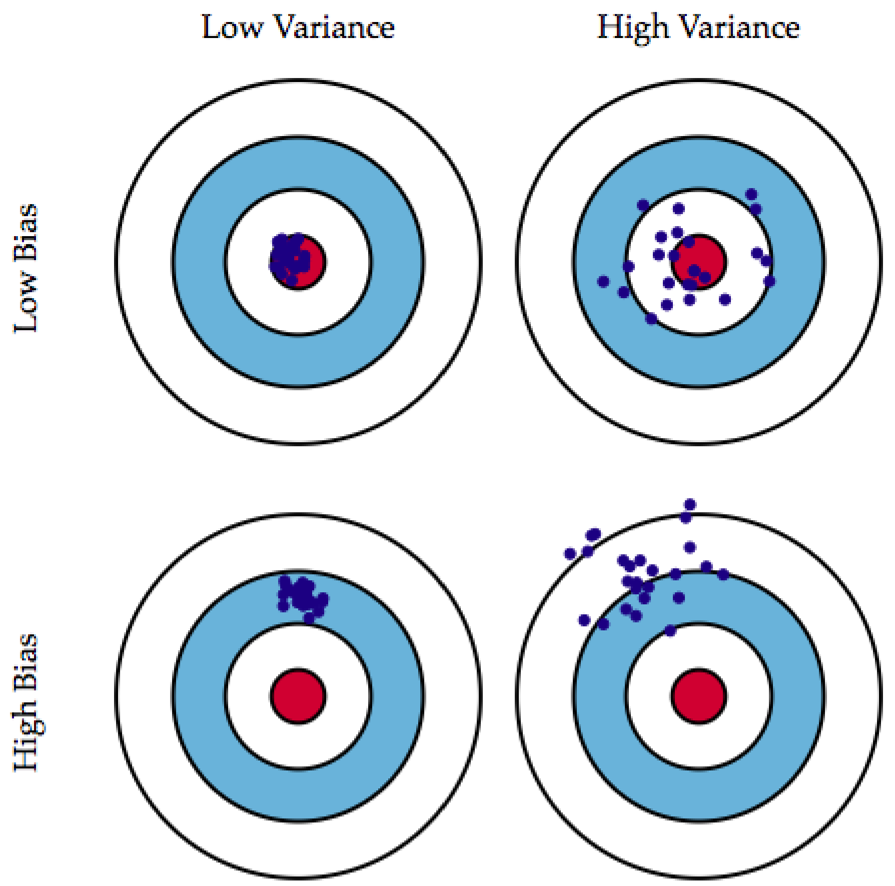
- **Model Variance**
  - If we sample several datasets $\mathcal{D}$ we will learn different models $y(\mathbf{x})$
  - variance measures the difference between each model learned from a particular dataset and what we expect to learn:

$$\text{variance} = \int \mathbb{E}\left[(y(\mathbf{x}) - \overline{y}(\mathbf{x}))^2\right] p(\mathbf{x}) d\mathbf{x}$$

$$\boxed{\overline{y}(\mathbf{x}) = \mathbb{E}[y(\mathbf{x})]}$$

  - Decreases with **simpler models**
  - Decreases with **more samples**

Credit: http://scott.fortmann-roe.com/docs/BiasVariance.html



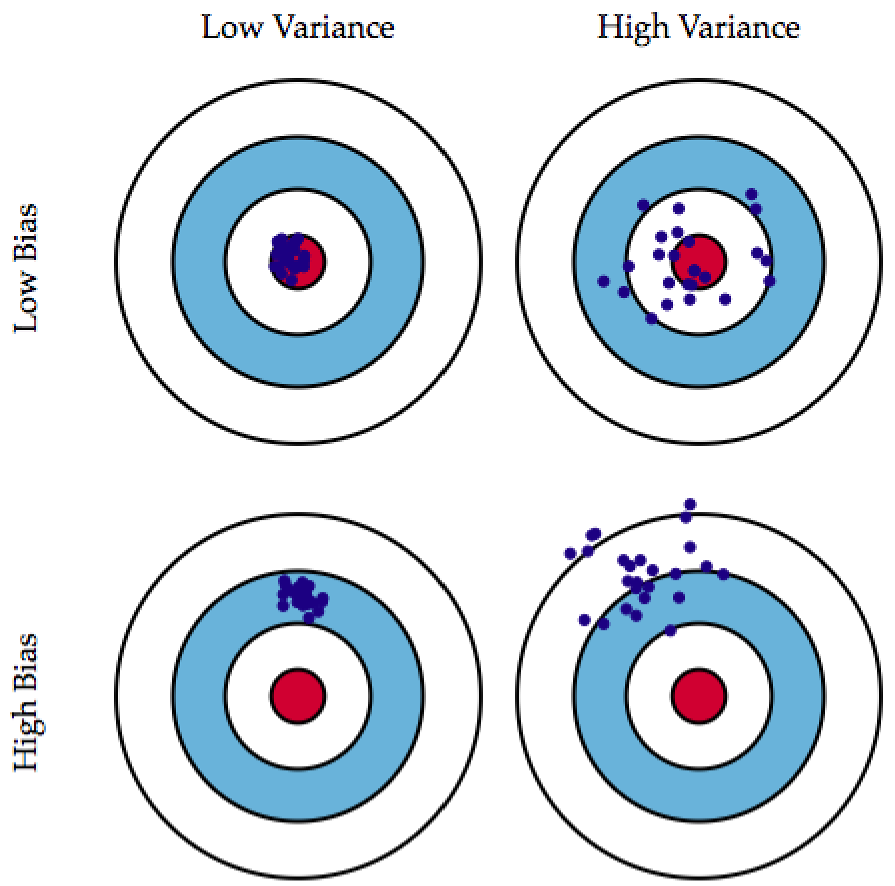
Bias-Variance Decomposition (3)

- **Model Bias**
  - bias measures the difference between **truth** (f) and what we expect to learn ($\mathbb{E}[y(\mathbf{x})]$):

$$\mathrm{bias}^2 = \int \left(f(\mathbf{x}) - \overline{y}(\mathbf{x})\right)^2 p(\mathbf{x})\mathrm{d}\mathbf{x}$$

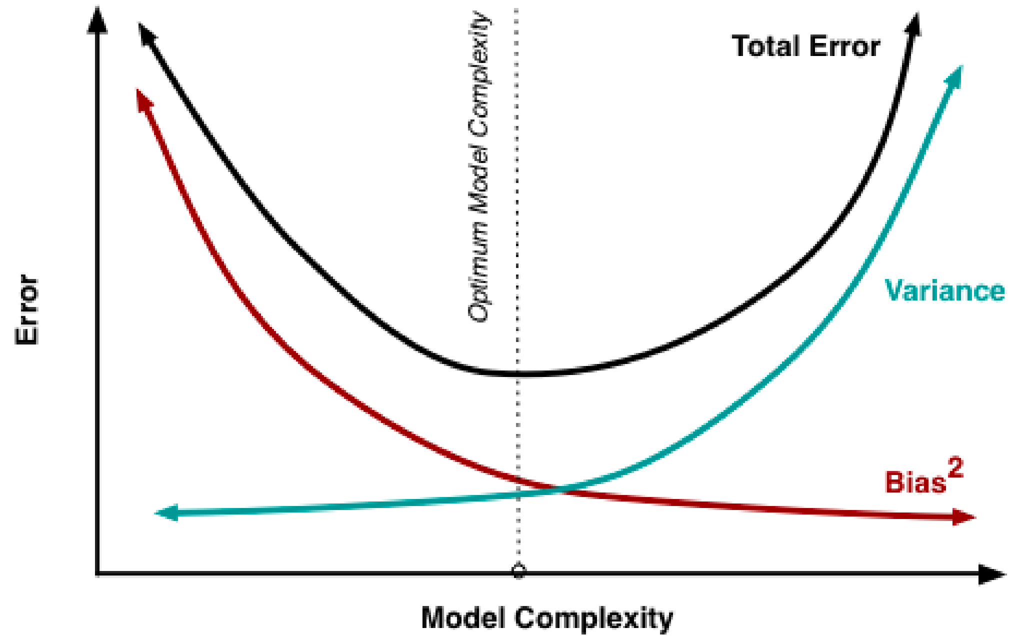
  - Decreases with **more complex models**
- **Data noise** (σ^2) is the variance of data and does not depend neither from data sampling nor from the model complexity

Credit: http://scott.fortmann-roe.com/docs/BiasVariance.html

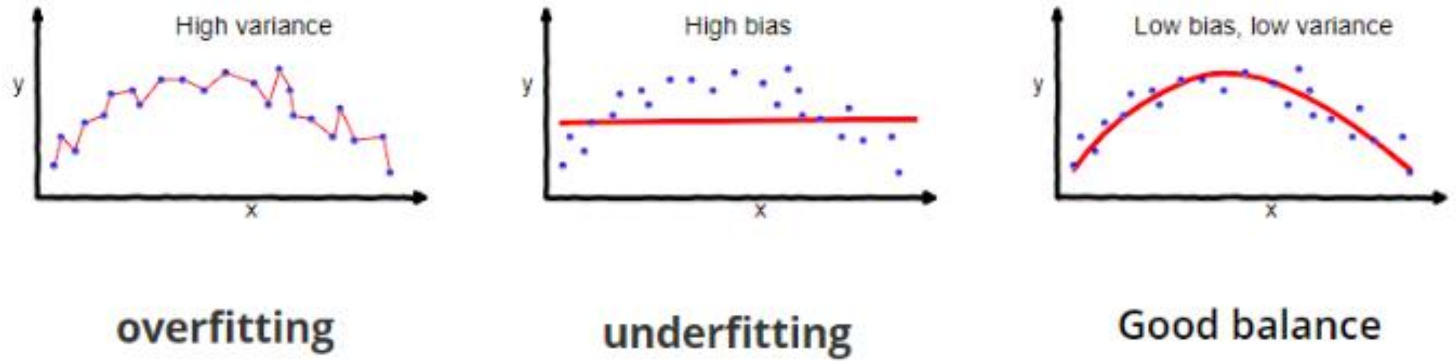


Bias-Variance Decomposition (4)

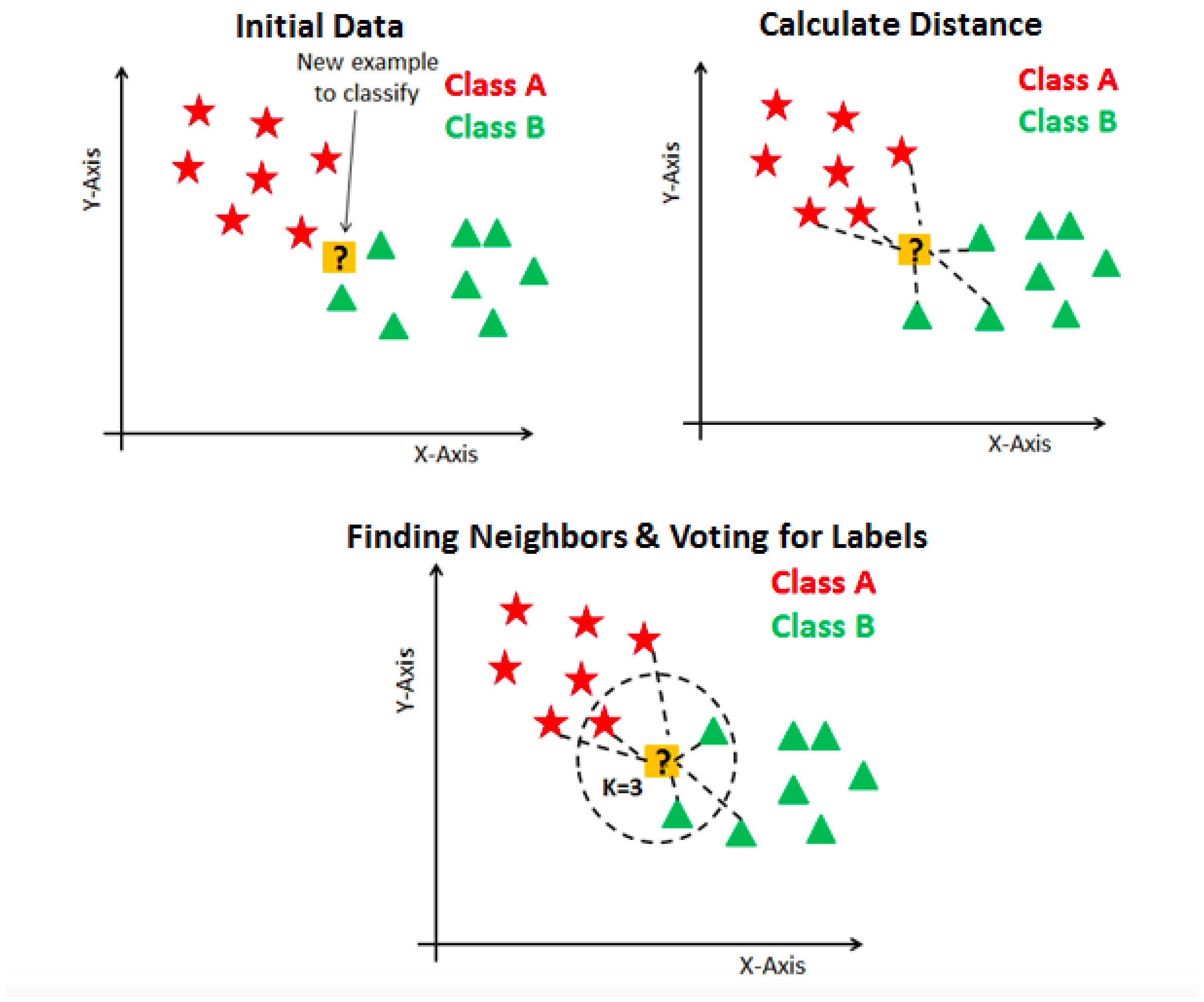
Credit: http://scott.fortmann-roe.com/docs/BiasVariance.html



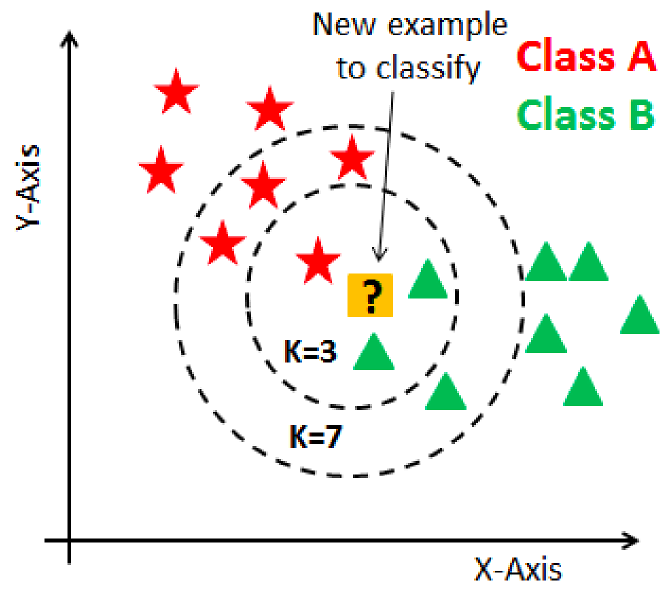
Credit: https://towardsdatascience.com/understanding-the-bias-variance-tradeoff–165e6942b229



A case study: Bias-Variance for K-NN



- Which is the best value for K?



Credit: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

A case study: Bias-Variance for K-NN (2)

- Bias-Variance analysis allows to understand how K affects the performance:

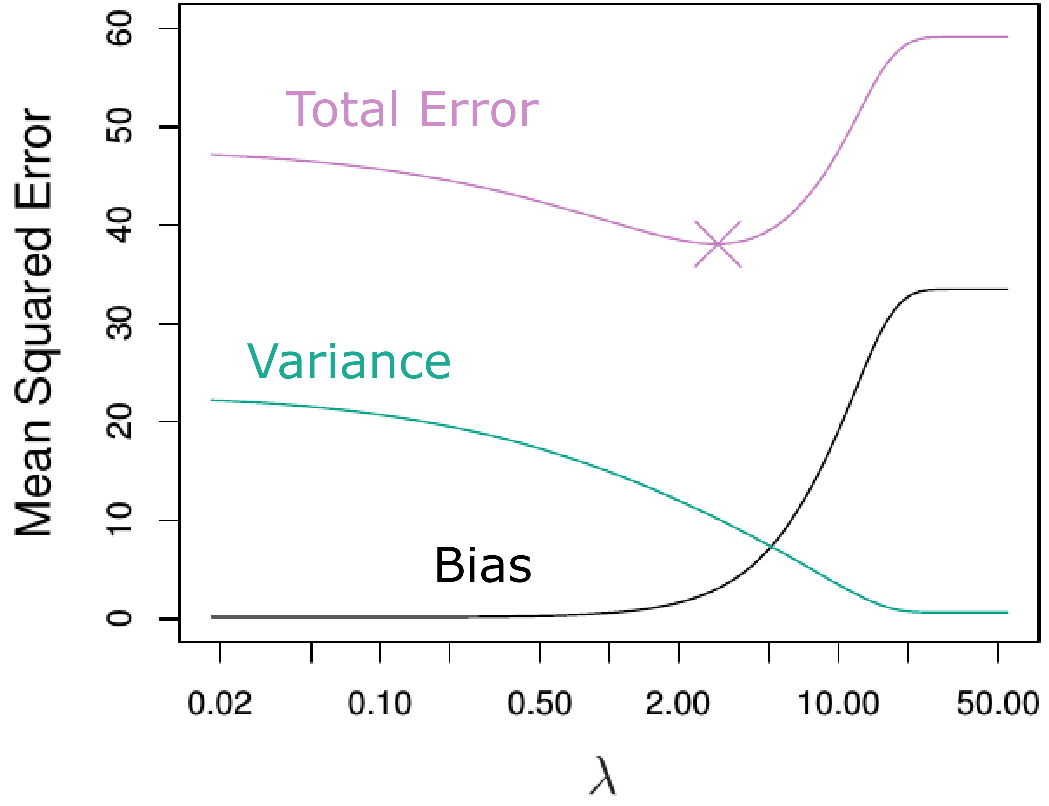
$$\mathbb{E}[(t^* - y(\mathbf{x}^*))^2] = \sigma^2 + \frac{\sigma^2}{K} + \left(f(\mathbf{x}^*) - \frac{1}{K}\sum_{i=1}^{K} f(\mathbf{x}_i) \right)^2$$

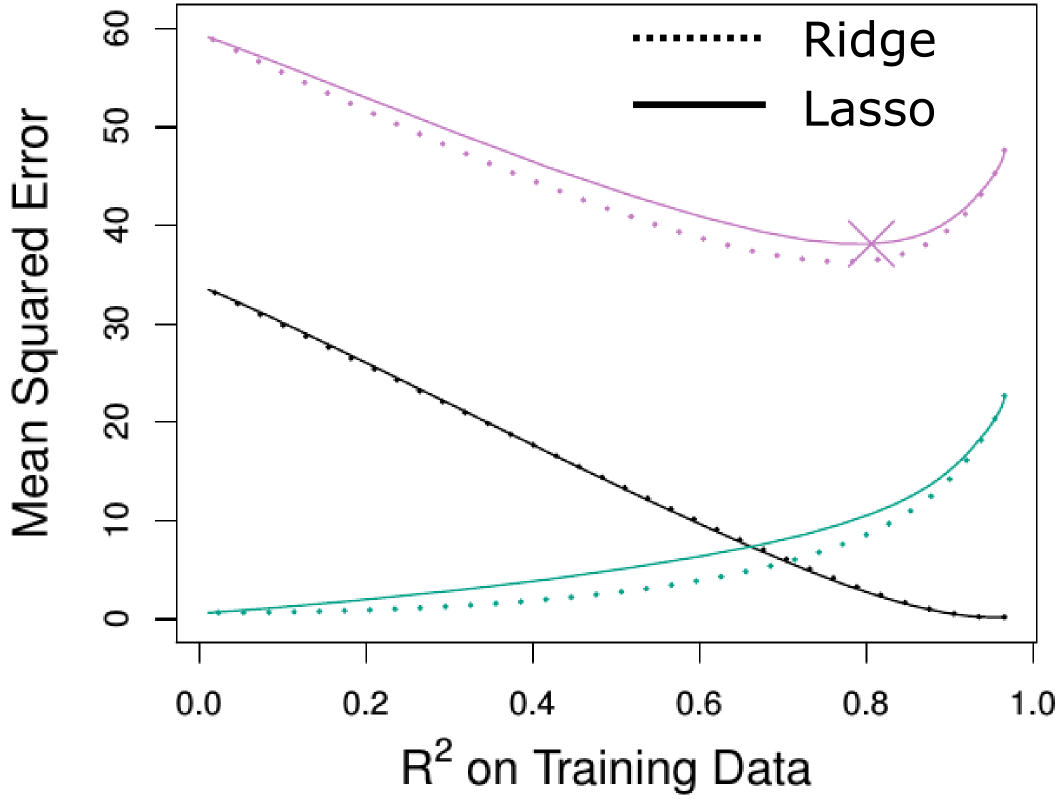
  - The data noise σ^2 is the **irreducible error**
  - The model **variance** $\frac{\sigma^2}{K}$ decreases as K increases
  - The model **bias** (the last *squared* term) depends on **smoothness** of problem space, but in general increases as K increases (because it increases the distance of the neighbours used to compute y)

Regularization and Bias-Variance

- The Bias-Variance decomposition explains why regularization allows to improve the error on unseen data
- Lasso outperforms Ridge regression when few features are related to the output

45 of 45 features correlated to output

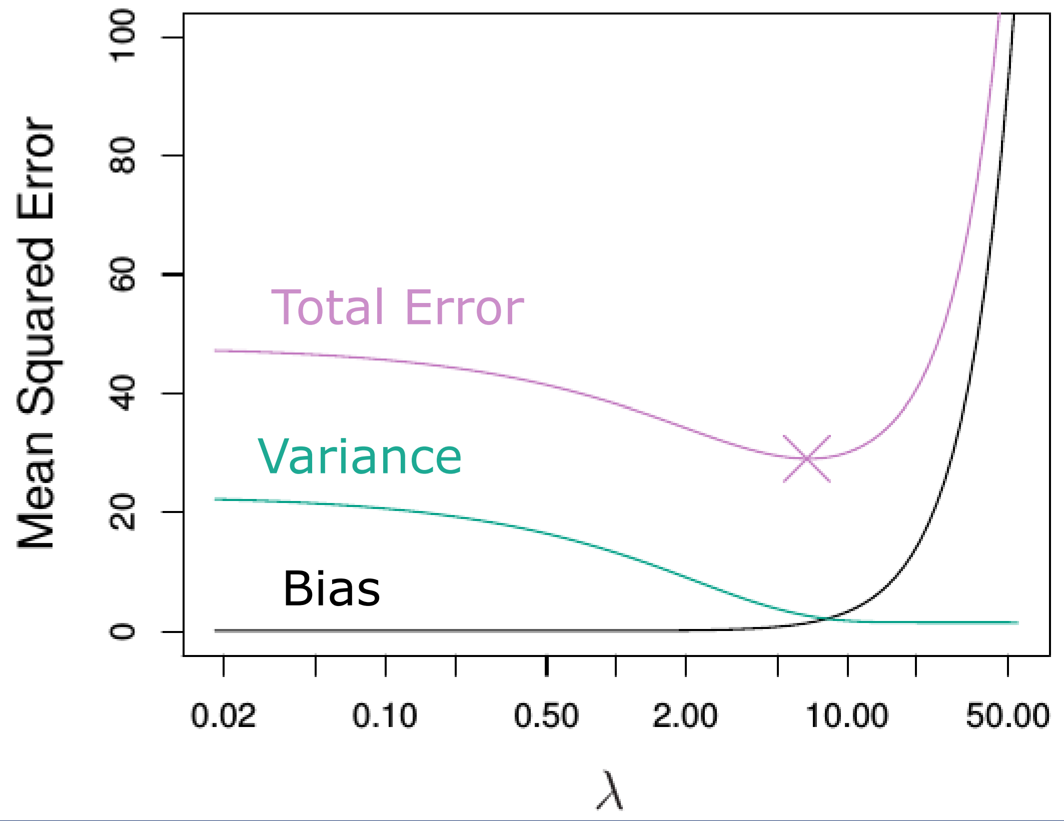


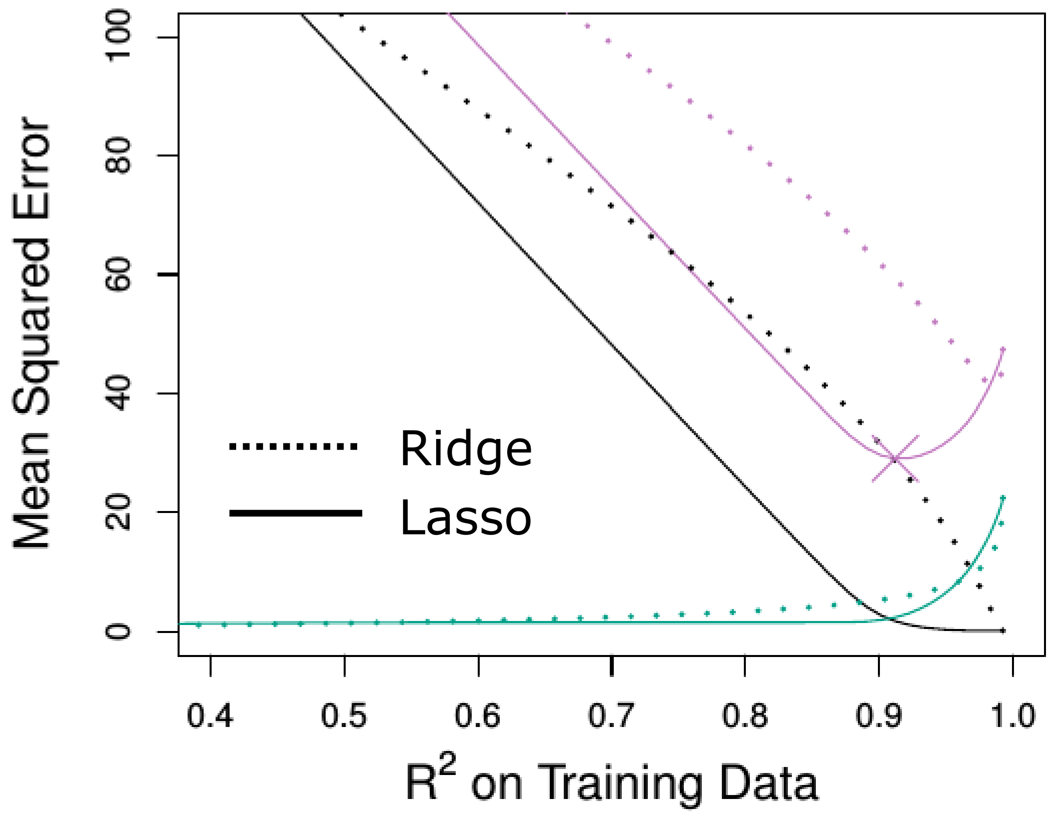


Regularization and Bias-Variance

- The Bias-Variance decomposition explains why regularization allows to improve the error on unseen data
- Lasso outperforms Ridge regression when few features are related to the output

2 of 45 features correlated to output





Model Assessment in Practice

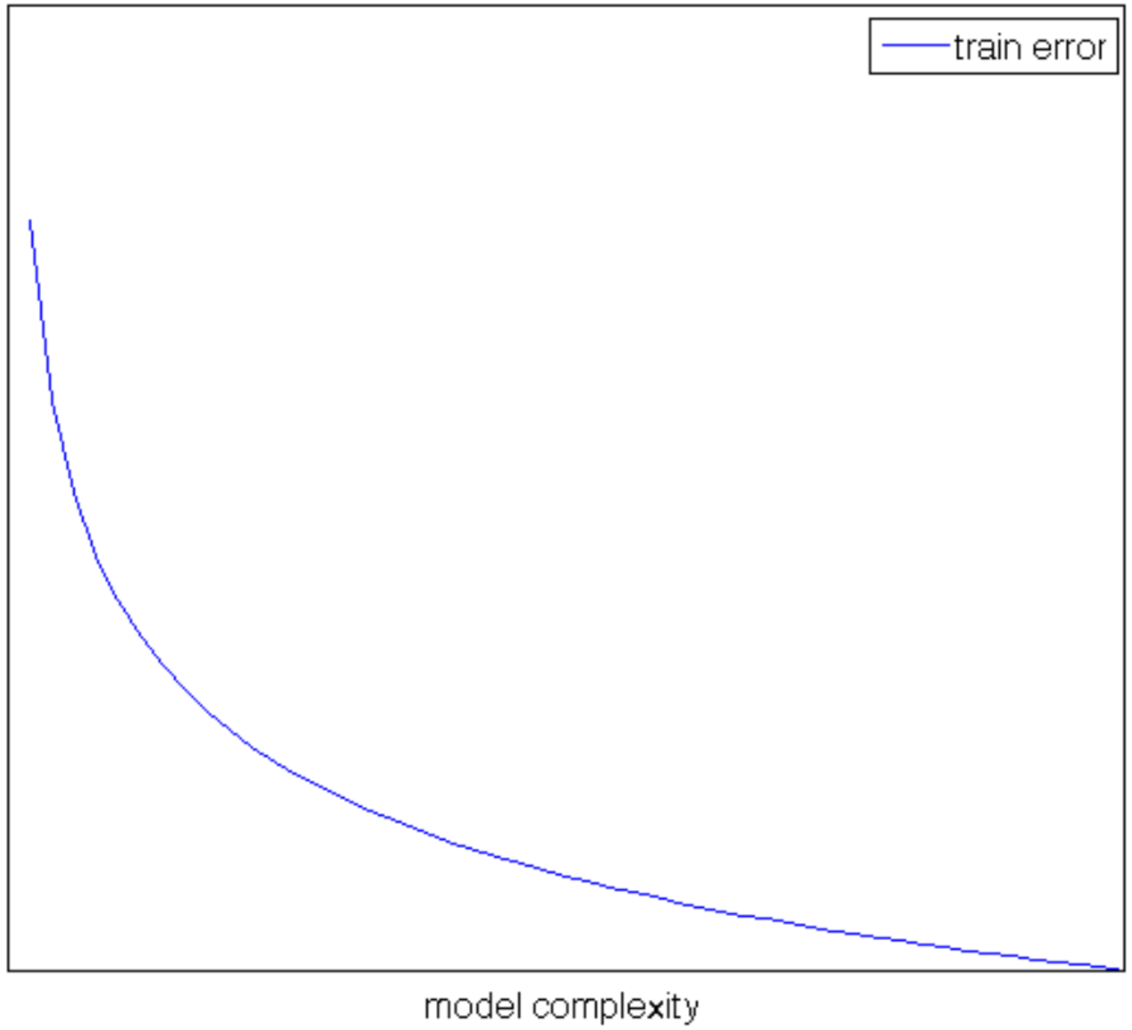
Training Error

- Given $\mathcal{D} = \{\mathbf{x}_i, t_i\}$ with $\mathrm{i} = 1, \dots, N$
- We can select a model based on the loss function L computed on $\mathcal{D}$:
  - Regression

$$L_{train} = \frac{1}{N}\sum_{n=1}^{N}(t_n - y(\mathbf{x}_n))^2$$

  - Classification

$$L_{train} = \frac{1}{N}\sum_{n=1}^{N}(I(t_n \neq y(\mathbf{x}_n)))$$



Prediction Error

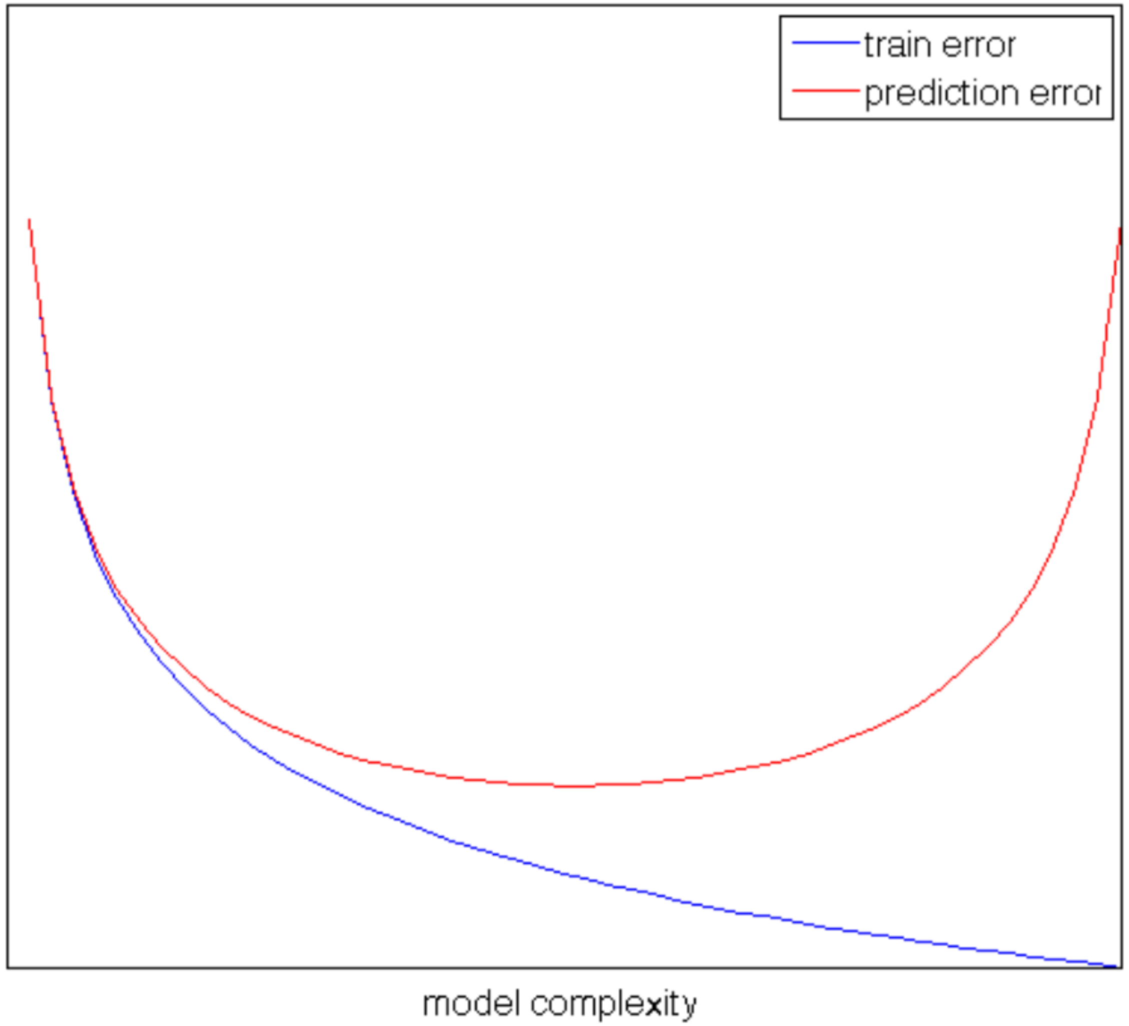
- We already saw that **training error** does not provide a good estimate of the error made on new data, the **prediction error**
  - Regression

$$L_{true} = \int\int (t - y(\mathbf{x}))^2 p(\mathbf{x}, t)\mathrm{d}\mathbf{x}\mathrm{d}t$$

  - Classification

$$L_{true} = \int\int I\left(t \neq y(\mathbf{x})\right) p(\mathbf{x}, t)\mathrm{d}\mathbf{x}\mathrm{d}t$$

- Unfortunately, we often are not able to model $p(\mathbf{x}, t)$



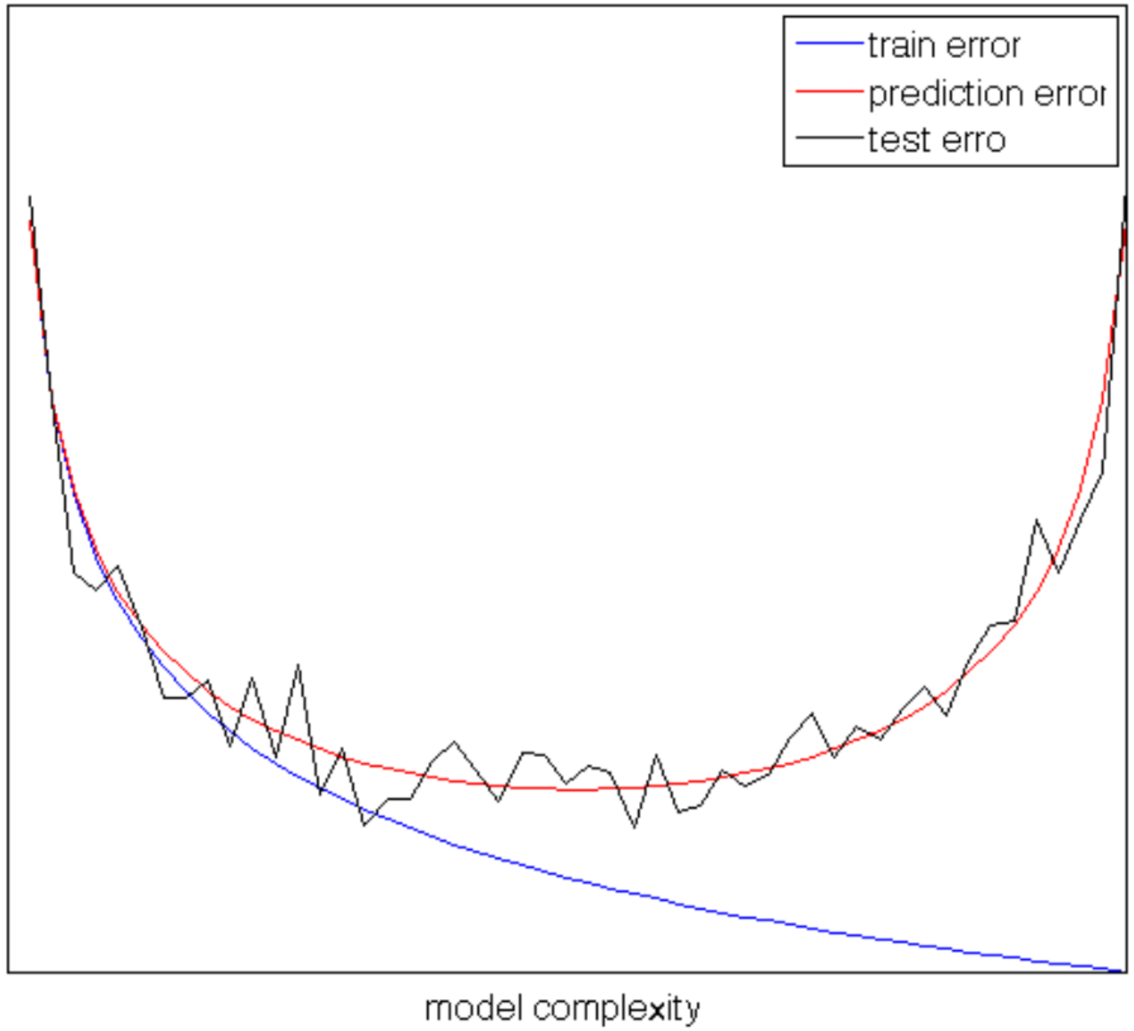
So what to do in practice? Test error

- What can we do in practice?
  - Split randomly data into a **training set** and **test set**
  - Optimize model parameters using the **training set**
  - Estimate the prediction error using the **test set**
    - Regression

$$L_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} (t_n - y(\mathbf{x}_n))^2$$

    - Classification

$$L_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} (I(t_n \neq y(\mathbf{x}_n)))$$



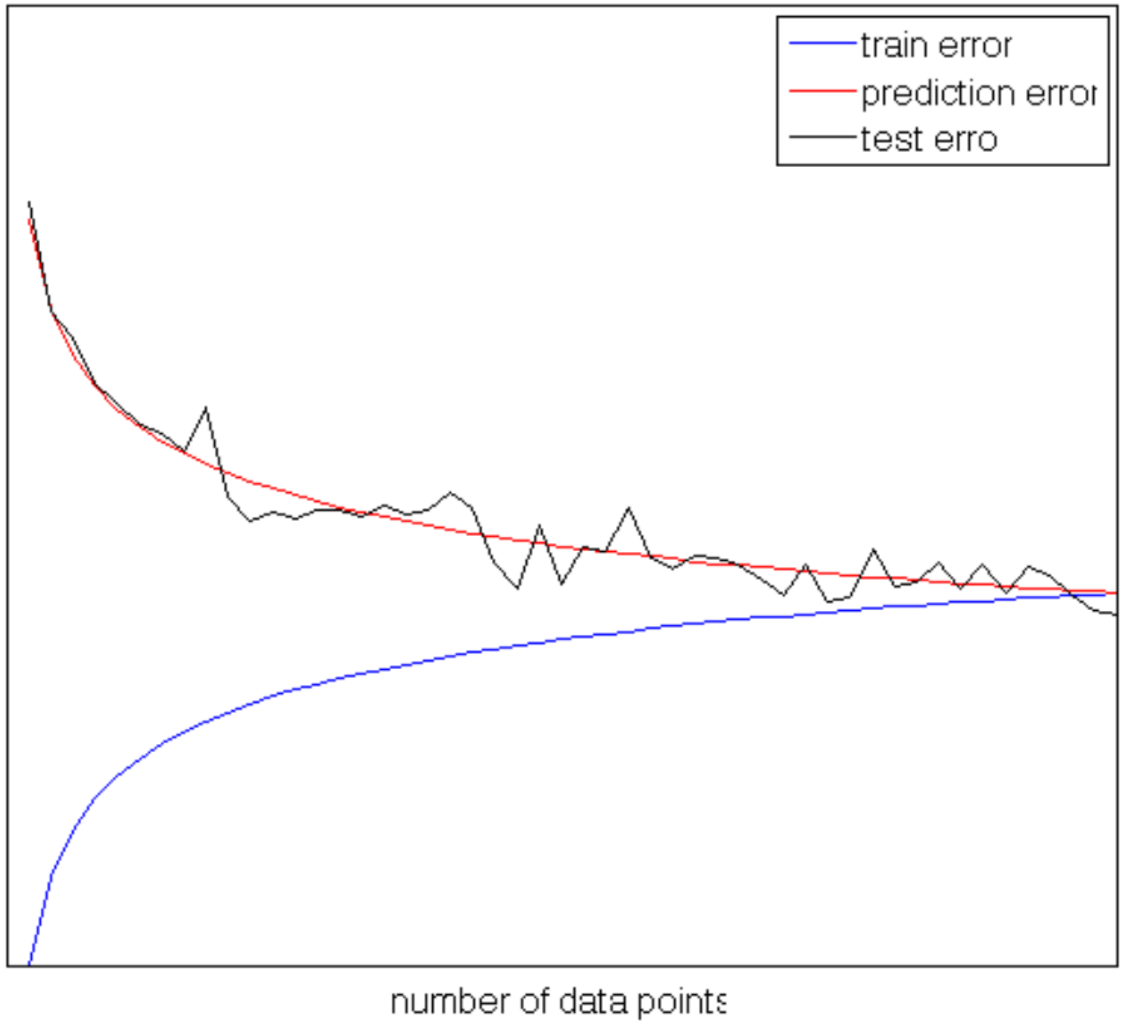
So what to do in practice? Test error

- What can we do in practice?
  - Split randomly data into a **training set** and **test set**
  - Optimize model parameters using the **training set**
  - Estimate the prediction error using the **test set**
    - Regression

$$L_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} (t_n - y(\mathbf{x}_n))^2$$

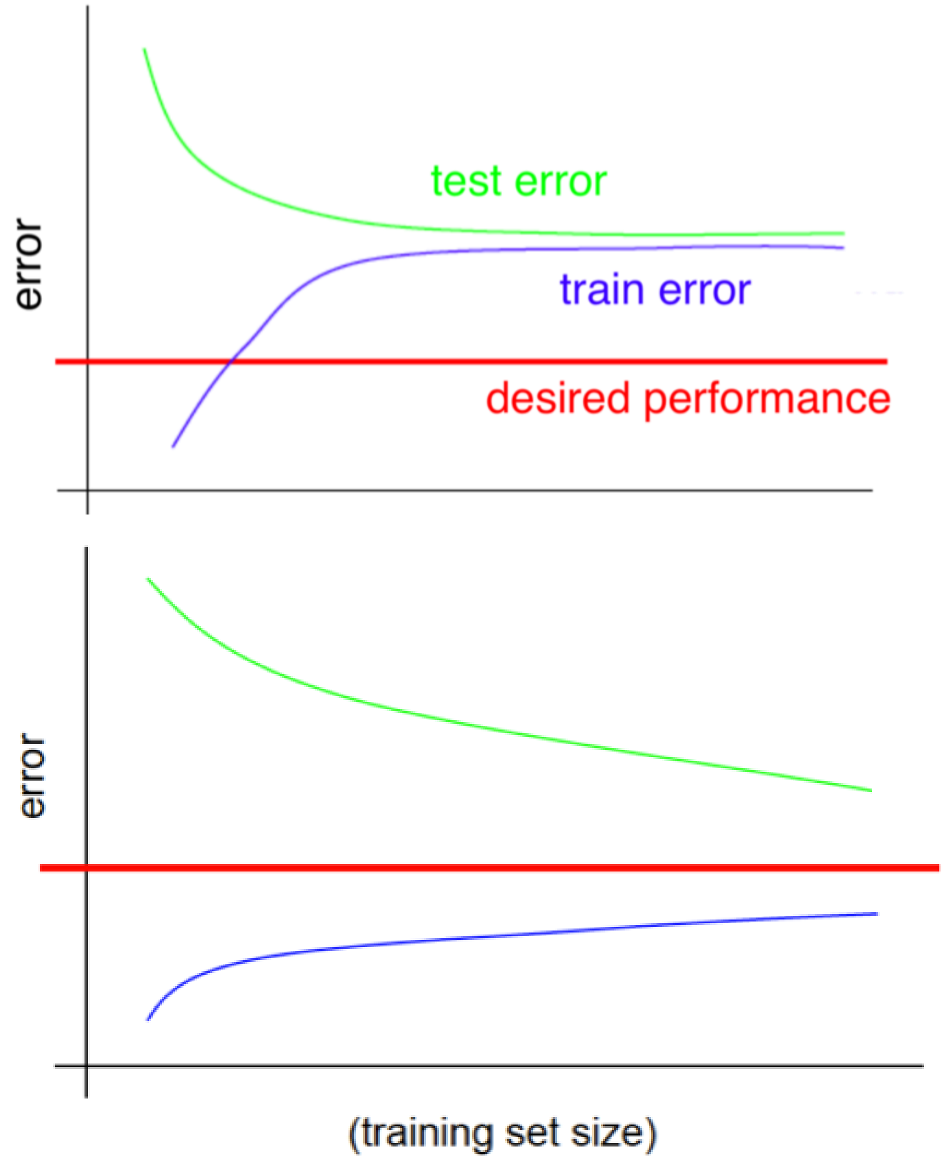
    - Classification

$$L_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} (I(t_n \neq y(\mathbf{x}_n)))$$



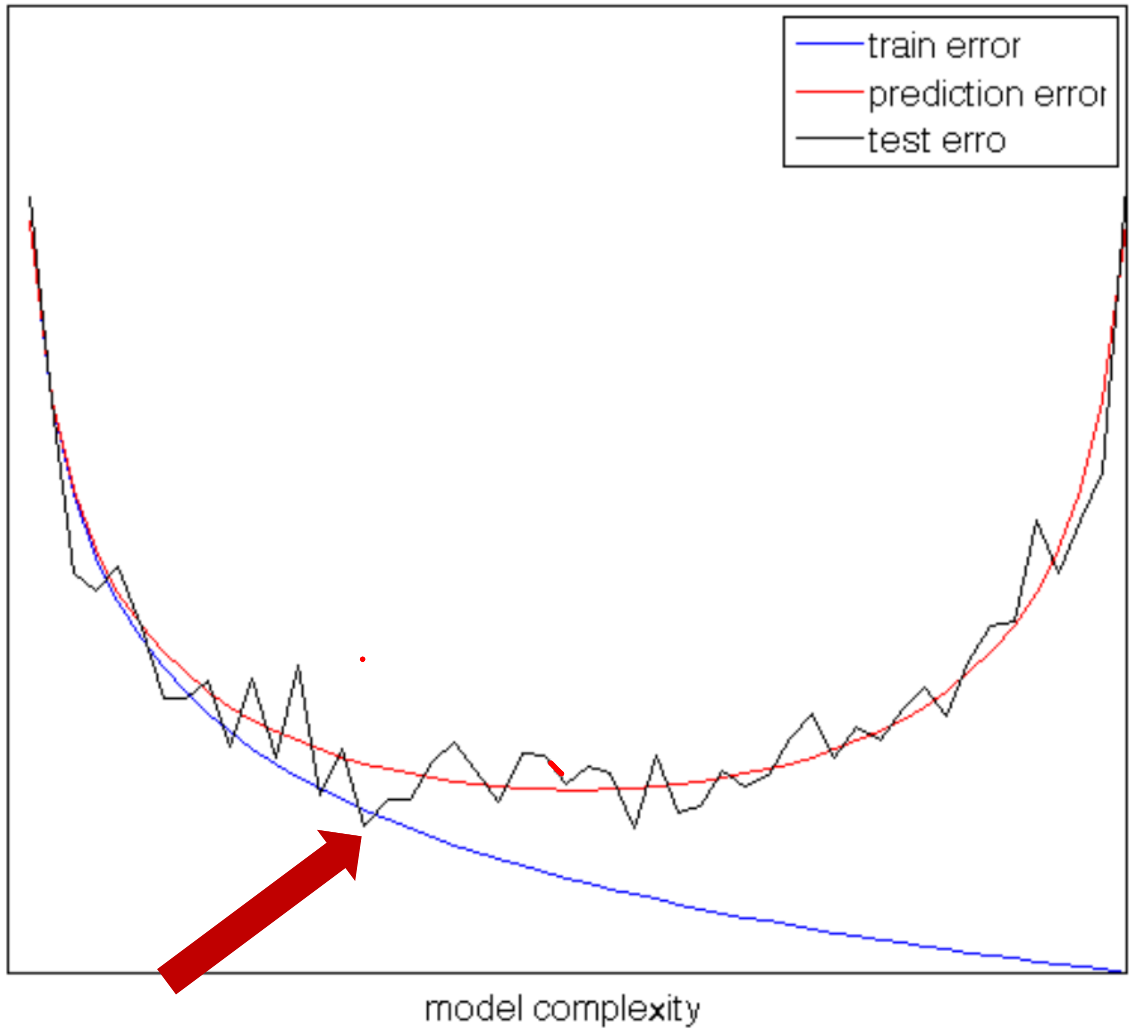
Analysis of train-test error

- The analysis of train-test errors allows to identify possible problems
  - **high bias**: training error is close to test error but they are both higher than expected
  - **high variance**: training error is smaller than expected and it slowly approaches the test error



Pitfalls of using test set

- Often data is limited and test error is usually small → prediction error can be either **overestimated** or **underestimated**
- Test error cannot be used for model selection → we can **overfit** test set
- Only if test set is **never used** for training and model selection, it can provide an **unbiased** estimate of prediction error
- So how to perform model selection?



Validation Set

- How to choose the best model and how do we set hyperparameters (e.g, α, λ)?
- We split data in three parts:
  - Training Data
  - Validation Data
  - Test Data
- How do we proceed?
  - We use **training data** to learn model parameters
  - For each model learned (i.e., different features and hyperparameters) we use **validation data** to compute the **validation error**
  - We select the model with the lowest **validation error** and finally use **test data** to estimate **prediction error**
- But...
  - to be reliable, validation data should be enough → less training data
  - we might **overfit validation data** and eventually not choose the best model

Leave-One-Out Cross Validation (LOO)

- For each sample $\{\mathbf{x}_i, t_i\} \in \mathcal{D}$
  - We train the model on $\mathcal{D} \backslash \{\mathbf{x}_i, t_i\}$
  - We compute the error of the resulting model on $\{\mathbf{x}_i, t_i\}$
- The estimate of the prediction error of our model will be the average of all the error computed using a single sample:

$$L_{LOO} = \frac{1}{N} \sum_{i=1}^{N} (t_i - y_{\mathcal{D}_i}(\mathbf{x}_i))^2$$

  - Where $y_{\mathcal{D}_i}$ is the model trained on $\mathcal{D} \backslash \{\mathbf{x}_i, t_i\}$
- L_{LOO} provides an **almost unbiased** estimate of prediction error (**slighlty pessimistic**)
- Unfortunately, LOO is **extremely expensive** to compute
  - As an example, even if training take just 1 second, computing LOO on 100K samples, would require 100K seconds (more than one day)!

K-Fold Cross Validation

- We randomly split the training data $\mathcal{D}$ into k folds: $\mathcal{D}_1, \dots, \mathcal{D}_k$
- For each fold $\mathcal{D}_i$
  - We train the model on $\mathcal{D} - \{D_i\}$
  - We compute the error on D_i

$$L_{\mathcal{D}_i} = \frac{k}{N} \sum_{(\mathbf{x}_n, t_n) \in \mathcal{D}_i} (t_n - y_{\mathcal{D} \setminus \mathcal{D}_i}(\mathbf{x}_n))^2$$

- Finally, we estimate the prediction error as the average error computed:

$$L_{k-fold} = \frac{1}{k} \sum_{i=1}^{k} L_{\mathcal{D}_i}$$

- L_{k-fold} provides a **slightly biased** estimate of prediction error (**pessimistic**) but it is much cheaper to compute (usually k=10 is used)

Complexity-Adjusted Model Evaluation

- Other metrics are used to evaluate models adjusting their training error based on their complexity:
  - Mallows's C_p : $C_p = \frac{1}{N}(RSS + 2M\hat{\sigma}^2)$
  - Akaike Information Criteria: $AIC = -2lnL + 2M$
  - Bayesian Information Criteria: $BIC = -2lnL + \mathrm{M}\ln(N)$
  - Adjusted R^2: $AdjustedR^2 = 1 - \frac{RSS/(N-M-1)}{TSS/(N-1)}$

M: number of parameters; N: number of samples; L: loss function;
$\hat{\sigma}^2$: estimate of noise variance;
RSS: residual sum of squares; TSS: total sum of squares

- AIC and BIC are generally used when maximizing the log-likelihood
- BIC generally penalize more than AIC model complexity

How to choose the right model complexity?

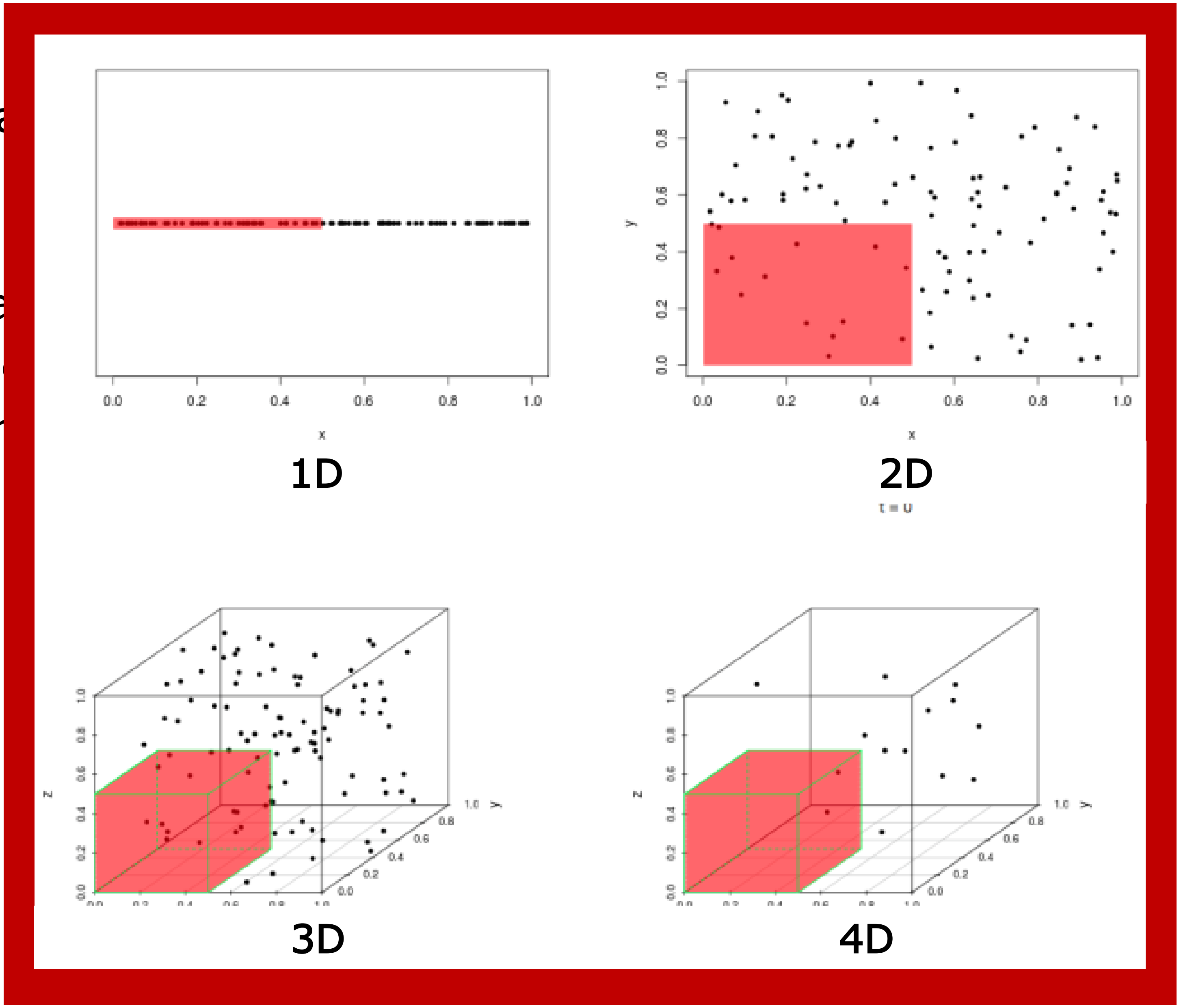
Curse of dimensionality

- Adding a feature means an exponential increase of volume of the input space

- Challenges
  - Computational cost
  - Amount of data…
  - … large model variance (overfitting)

Curse of dimensionality

- Adding a fea input space
- Challenges
  - Computa
  - Amount
  - … large r



Curse of dimensionality

- Adding a feature means an exponential increase in volume of the input space

- Challenges
  - Computational cost
  - Amount of data…
  - … large model variance (overfitting)

- A common pitfall is to think that using more features is always better:
  - Is $y = w_0 + w_1 x + w_2 x^2$ always better than $y = w_0 + w_1 x$, isn't it?
  - In fact, we can always set $w_2 = 0$
  - No! Increasing the number of features increases the probability of overfitting the data (also because I could not have enough data for a larger feature space)!

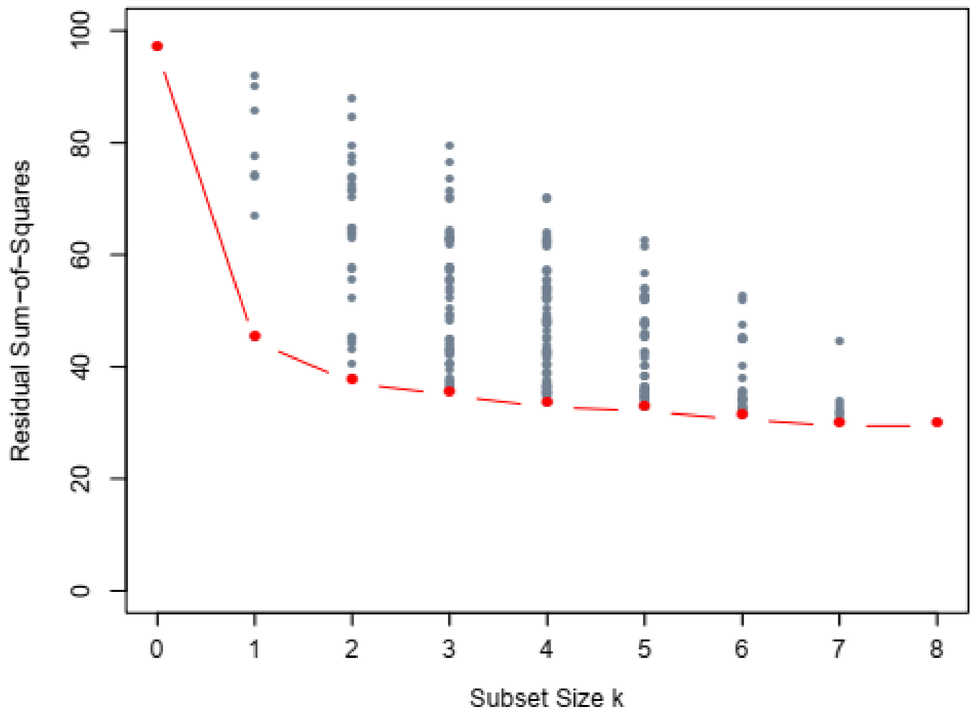
Reducing the variance

- We want to select the model with the lowest prediction error
- This can be achieved by reducing the variance of the model:
  - Feature Selection: we should design the feature space by selecting the **most effective subset** of all the possible features
  - Dimensionality Reduction: the input space can be mapped to a **lower-dimensional space**
  - Regularization: the values of the parameters are **shrunked toward zero**
- These three approaches are not necessarily mutually exclusive and they can be used toghether

Feature Selection

Best Subset Selection: a brute force

- The simplest idea seems to compare all the possible combinations of the features
- Assuming we have M features, for each $k = 1, \dots, M$:
  - Train all the $\binom{M}{k} = \frac{M!}{k!(M-K)!}$ models with exactly k features
  - Chose the best model
- Computation unfeasible!



Feature Selection in Practice

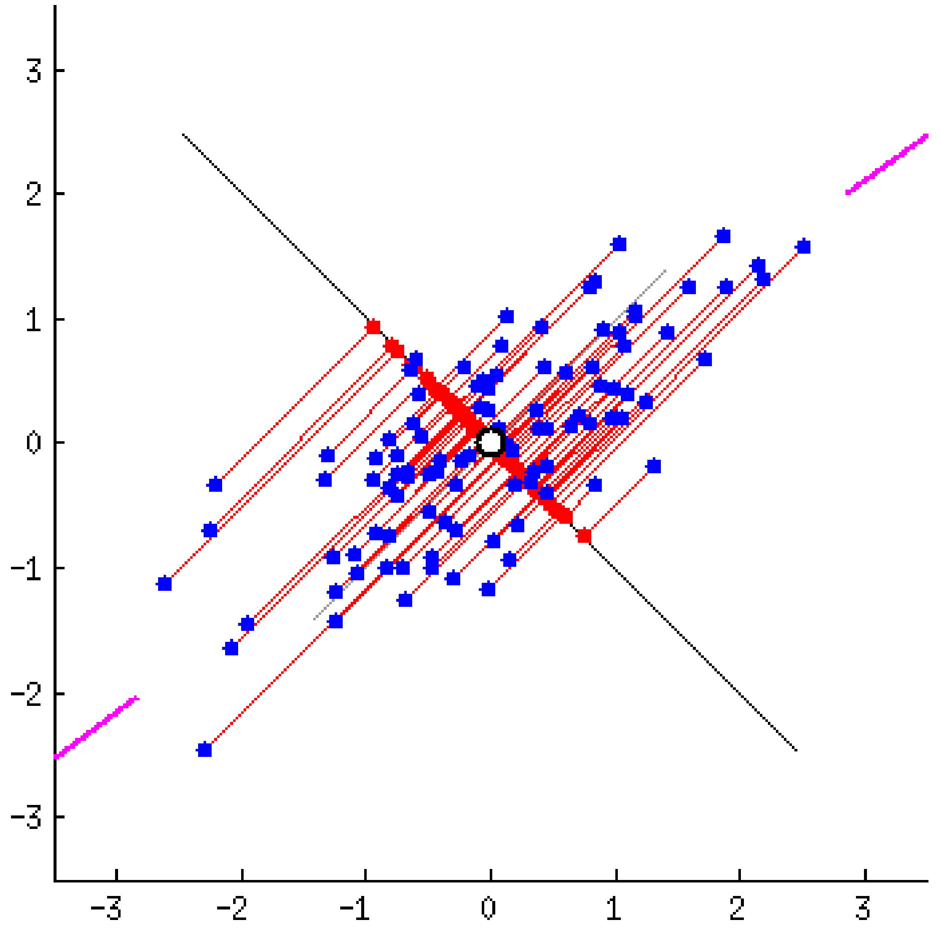
- **Filter**: features are ranked **independently** based on some evaluation metrics (e.g., correlation, variance, information gain, etc.) and the top k are selected
  - Very fast but fails to capture any subsest of **mutually dependent features**
- **Embedded**: feature selection is performed as a step of the machine learning approach used (e.g., lasso, decision trees, etc.)
  - Not expensive but the features identified are **specific** to the learning approach
- **Wrapper**: a search algorithm is used to find a subset of features that are evaluated by training a model with them and assessing its performance
  - Either a simpler model or a simple machine learning approach can be used to evaluate the features
  - Greedy algorithms are generally used to search the best subset of features

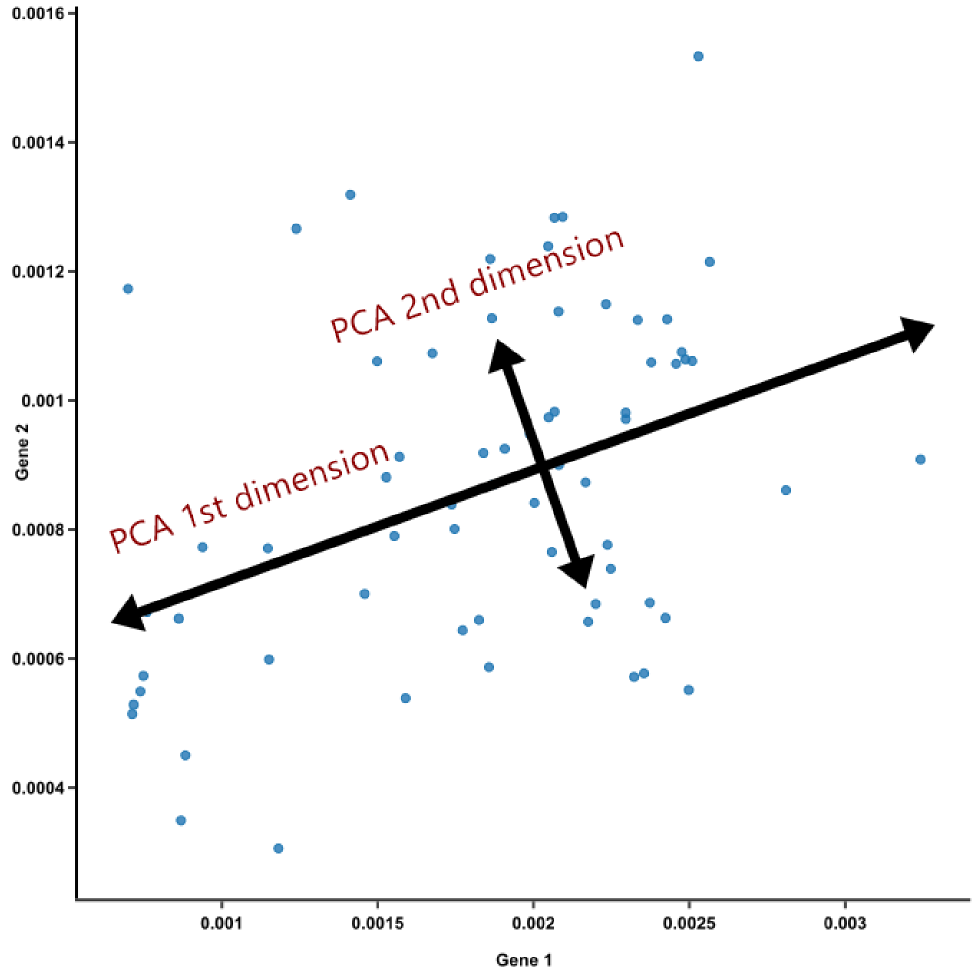
Dimensionality Reduction

Dimensionality Reduction

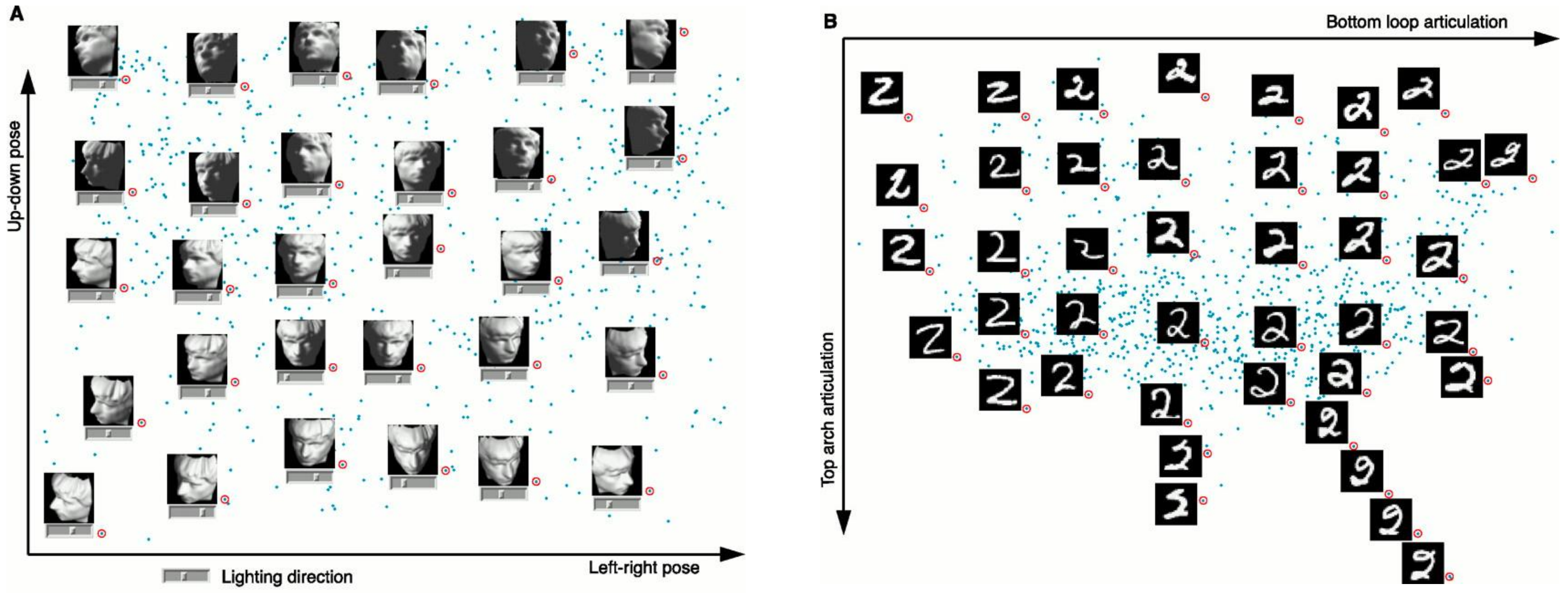
- Dimensionality reduction aims at reducing the dimensions of input space, but it differs from feature selection in two major respects:
  - it uses **all** the features and **maps** them into a lower-dimensionality space
  - it is an **unsupervised** approach
- There are many methods to perform dimensionality reduction:
  - **Principal Component Analysis (PCA)**
  - Indipendent Component Analysis (ICA)
  - Self-Organizing Maps
  - Autoencoders
  - ISOMAP
  - t-SNE
  - …

PCA underlying idea





DR: an example with complex data



[A global geometric framework for nonlinear dimensionality reduction. Tenenbaum, de Silva, and Langford. Science, 290(5500):2319–2323, 2000.]

Bagging and Boosting

Improving the Bias-Variance Tradeoff

- So far we saw how to reduce the variance, searching the best tradeoff with the increased bias...
- ... but it is possible to reduce the variance **without increasing bias**?
- Or is it possible to **reduce the bias**?
- We can achieve these results by using two **ensemble methods** that consist of learning **several models** and **combine** them:
  - Bagging
  - Boosting

Which is the idea behind bagging?

- Let assume to have N datasets and to learn from them N models, $y_1\, y_2, \ldots, y_N$
- Now let us compute an aggregate model as $y_{AGG} = \frac{1}{N}\sum_{i=1}^{N} y_i$
- If the datasets are **independent**, the model variance of y_{AGG} will be 1/N of the model variance of the single model y_i
  - To have an intuition of this, let assume to have random variable $\bar{x} = \frac{1}{N}\sum_N x$

$$Var(\bar{x}) = \frac{1}{\mathrm{N}^2}\sum_N Var(x) = \frac{1}{N} Var(x)$$

- However, we generally do not have N dasets!
- So, what can we do?

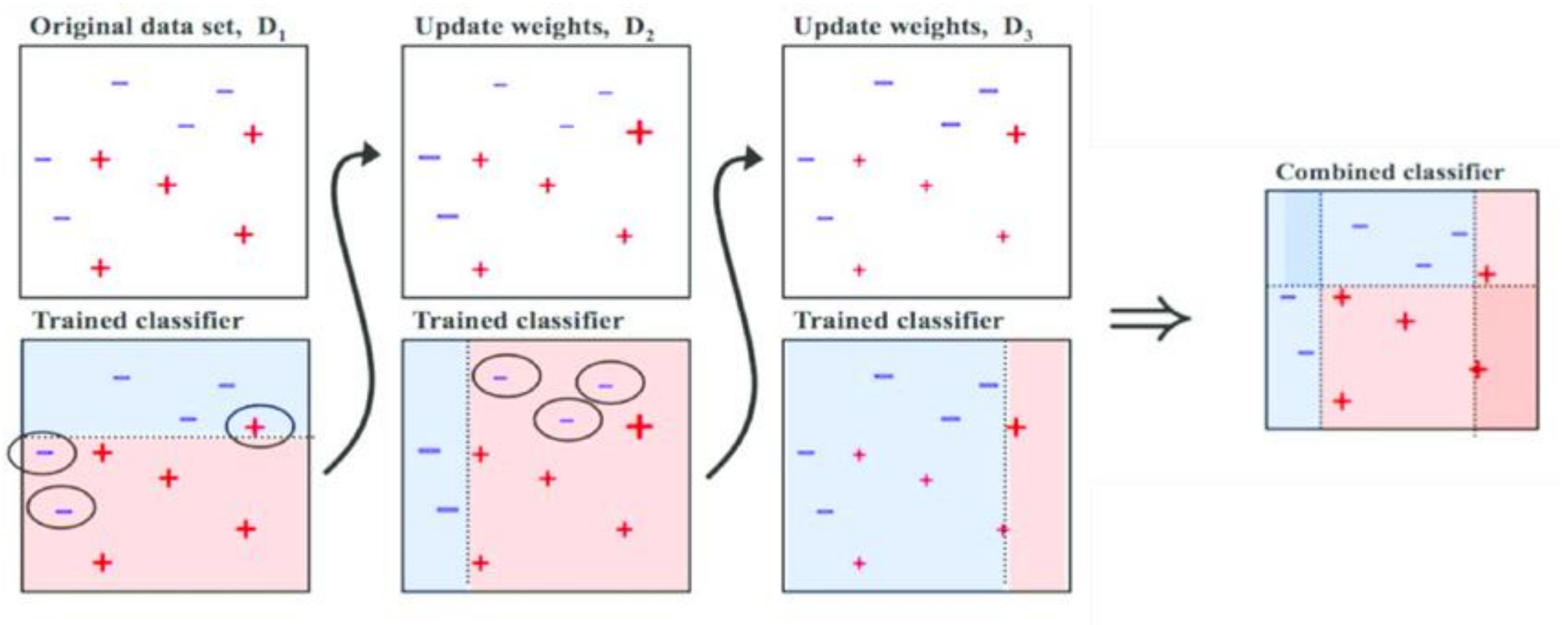
Bagging

- Bagging stands for Bootstrap Aggregation:
  - Generate N datasets applying **random sampling with replacement**
  - Train a model (classification or regression model) from **each** dataset generated
  - To compute the **prediction** for new samples, apply all the trained models and combine the outputs with **majority voting** (classification) or **averaging** (regression)
- Bagging is generally helpful and reduce the variance, although the sampled datasets are **not independent**
  - It helps with **unstable learners**, i.e., learners that change significantly with even small changes in the dataset (**low bias and high variance**)
  - It helps when we have a lot of overfitting (**low bias and high variance**)
  - It does not help when learners is **robust**, i.e., not sensitive to change in data (**usually higher bias but lower variance**)

What is Boosting?

- The goal of boosting is to achieve a **small bias** by using on simple (**weak**) learners
- The key idea behind boosting is to **iteratively train** a series of **weak learners**, with each iteration focusing on the samples that were misclassified in the previous iteration.
- As final result, an **ensemble model** is built by **combining** the outputs of all the weak learners trained

AdaBoost



Bagging vs Boosting

Bagging

- Reduces variance
- Not good for stable learners
- Can be applied with noisy data
- Usually helps but the difference might be small
- Naturally parallel

Boosting

- Reduces bias (generally without overfitting)
- Works with stable learnes
- Might have problem with noisy data
- Not always helps but it can makes the difference
- Serial