Deep Learning

A course about theory & practice

UNIVERSITÀ DI PAVIA

Predictions

Marco Piastra

An Empirical View

Regression with Feed-Forward Neural Network

Target function: $y = f^*(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d$

Dataset

$$D := \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$$

Representation *(Estimator)*

$$\tilde{y} = \boldsymbol{w} \cdot g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b, \quad \boldsymbol{W} \in \mathbb{R}^{h \times d}, \ \boldsymbol{w}, \boldsymbol{b} \in \mathbb{R}^h, b \in \mathbb{R}$$

Evaluation *(Mean Squared Error)*

$$L(D) := \frac{1}{N} \sum_{i=1}^{N} (\tilde{y}(\boldsymbol{x}^{(i)}) - y^{(i)})^2$$

Optimization *(Gradient descent, and its variants)*

$$\Delta \boldsymbol{W} = -\eta \frac{1}{N} \sum_{D} \frac{\partial}{\partial \boldsymbol{W}} L(\tilde{y}^{(i)}, y^{(i)}) \qquad \Delta \boldsymbol{b} = -\eta \frac{1}{N} \sum_{D} \frac{\partial}{\partial \boldsymbol{b}} L(\tilde{y}^{(i)}, y^{(i)})$$

$$\Delta \boldsymbol{w} = -\eta \frac{1}{N} \sum_{D} \frac{\partial}{\partial \boldsymbol{w}} L(\tilde{y}^{(i)}, y^{(i)}) \qquad \Delta b = -\eta \frac{1}{N} \sum_{D} \frac{\partial}{\partial b} L(\tilde{y}^{(i)}, y^{(i)})$$

Independent, Identically Distributed (iid)

Dataset

$$D = \left\{ (\boldsymbol{x}^{(i)}, y^{(i)}) \right\}_{i=1}^{N}$$

data item

Identically distributed

$$p^* \left(\boldsymbol{x}^{(i)}, y^{(i)} \right) = p^* \left(\boldsymbol{x}^{(j)}, y^{(j)} \right), \quad \forall i, j$$

where $p^*(\boldsymbol{x}, y)$ is the (*unknown*) **true** probability of all possible data items

Independent

$$p^* \left(\left\{ (\boldsymbol{x}^{(i)}, y^{(i)}) \right\}_{i=1}^{N} \right) = \prod_{i=1}^{N} p^* \left(\boldsymbol{x}^{(i)}, y^{(i)} \right)$$

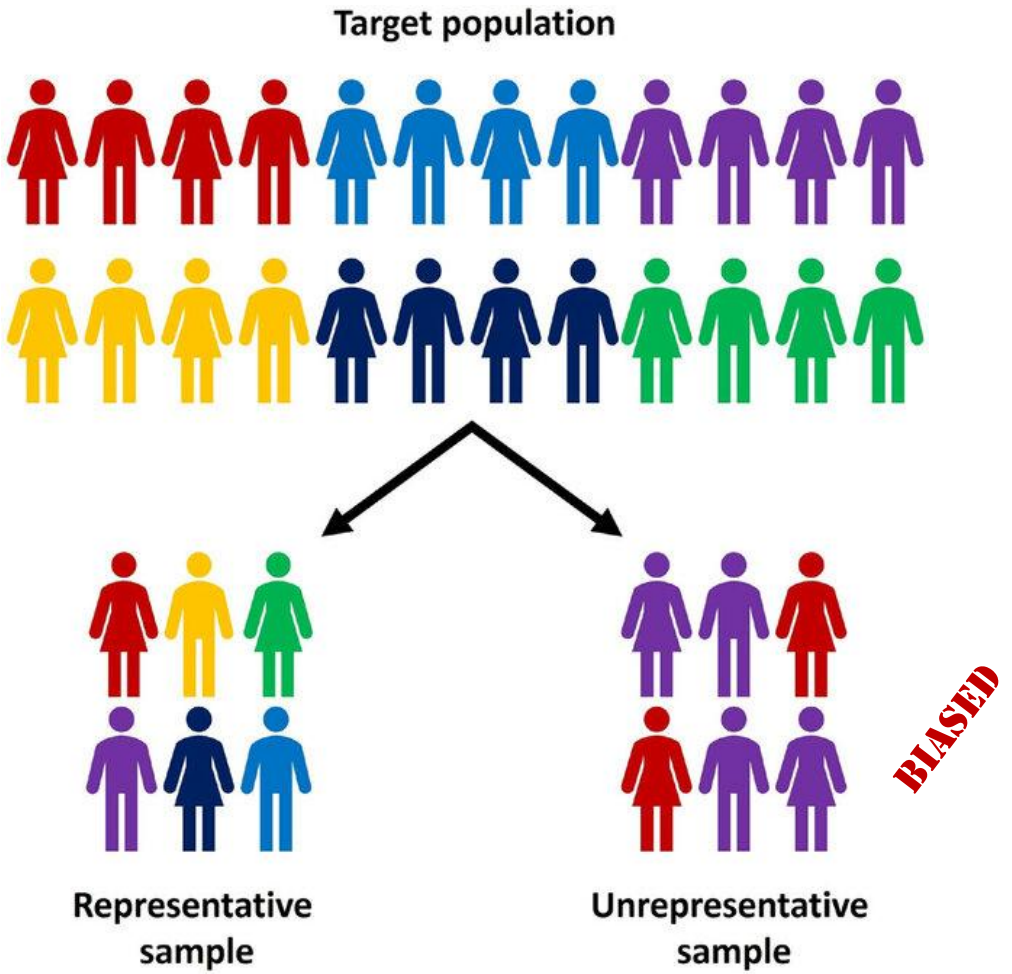
Unbiased Dataset

Is the dataset representative of the true distribution $p^*(\boldsymbol{x}, y)$ *of the population?*

$$p^*(\boldsymbol{x}, y)$$

probability in the dataset

$$p_D\left(\boldsymbol{x}^{(i)}, y^{(i)}\right) = \frac{N_{\boldsymbol{x}=\boldsymbol{x}^{(i)}, y=y^{(i)}}}{N}$$



[Image from https://cms.galenos.com.tr/Uploads/Article_53618/Diagn%20Interv%20Radiol-28-450-En.pdf]

Predictions?

Optimization:
The aim is finding the parameters that make the representation best approximating the target function *over the dataset*

However, in general $p_D(\boldsymbol{x}, y) \neq p^*(\boldsymbol{x}, y)$

Fundamental question:
How good is the approximator with data items that are *not* in the dataset?

In other terms, given $p_D(\boldsymbol{x}, y) = p_D(y|\boldsymbol{x})\, p_D(\boldsymbol{x})$

what happens when $p_D(\boldsymbol{x}) = 0$?

Predictions?

Problem:
Why should an approximator, however powerful, be able to deal with the case $p_D(\boldsymbol{x}) = 0$?

- **Inductive Bias**

  The structural assumptions embedded within the system's architecture,
  alongside those presumed to govern the target population:

  - **Continuity and Smoothness:**
    A Feed-Forward Neural Network synthesizes a continuous function.
    This forces the network to produce a "smooth" transition from the known regions $p_D(\boldsymbol{x}) > 0$
    into the 'unknown' $p_D(\boldsymbol{x}) = 0$
  - **Manifold Hypothesis:**
    The assumption that high-dimensional data points $\boldsymbol{x}$ where $p^*(\boldsymbol{x}) > 0$
    concentrate on or near a lower-dimensional, continuous manifold

Overfitting

When the training process becomes too specific to the training set

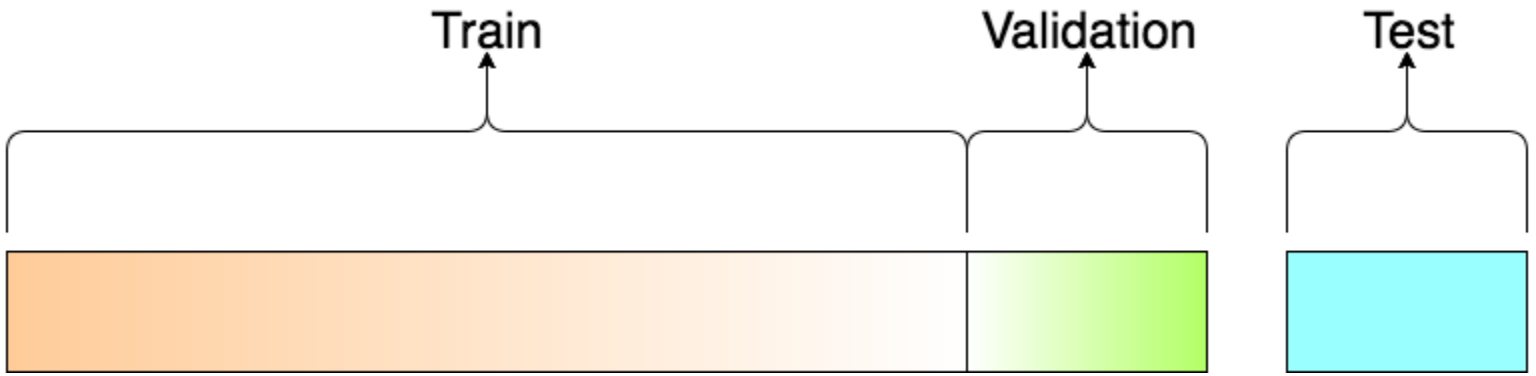
- ***Training set, validation set, test set***

  Splitting the dataset

$$D = D_{train} \cup D_{val} \cup D_{test}$$

$$\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N} = \{(\boldsymbol{x}^{(j)}, y^{(j)})\}_{j=1}^{N_{train}} \cup \{(\boldsymbol{x}^{(k)}, y^{(k)})\}_{k=1}^{N_{val}} \cup \{(\boldsymbol{x}^{(l)}, y^{(l)})\}_{l=1}^{N_{test}}$$

$$N_{train} \gg N_{val}, N_{test}$$



Overfitting

When the training process becomes too specific to the training set

- ***Training set, validation set***

  Splitting the dataset

  $$D = D_{train} \cup D_{val} \cup D_{test}$$

  $$\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N} = \{(\boldsymbol{x}^{(j)}, y^{(j)})\}_{j=1}^{N_{train}} \cup \{(\boldsymbol{x}^{(k)}, y^{(k)})\}_{k=1}^{N_{val}} \cup \{(\boldsymbol{x}^{(l)}, y^{(l)})\}_{l=1}^{N_{test}}$$

  $$N_{train} \gg N_{val}, N_{test}$$

  Training is made on D_{train} only

  At each *epoch* — *when the whole D_{train} has been processed*

  the loss function is evaluated on D_{val}

  After some epochs, the performance on D_{val} might get worse

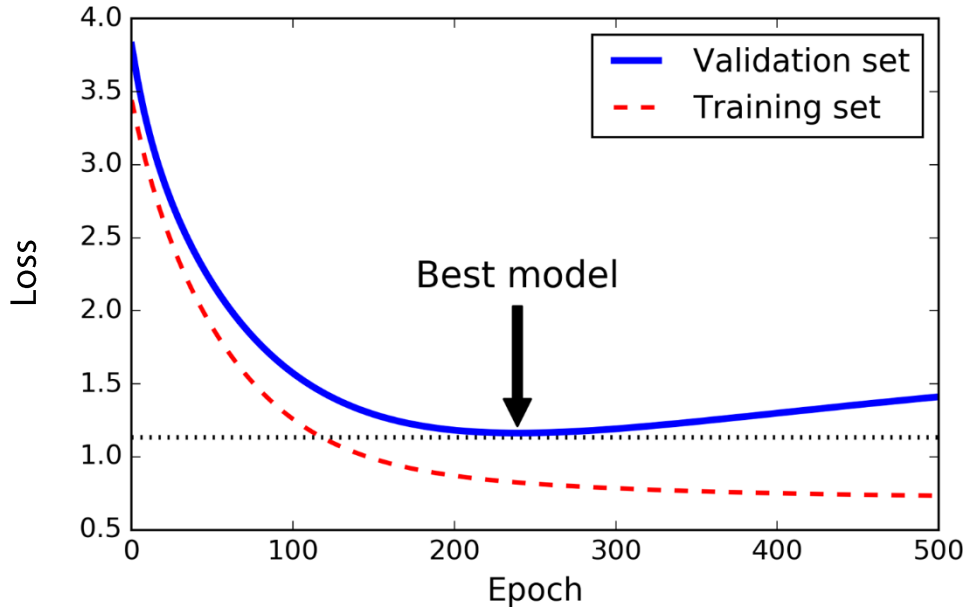


Image from https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html

k-Fold Cross-Validation

- ***One dataset, multiple splits***
  1) Divide the dataset into k splits (i.e. *folds*)
  2) Use k - 1 folds for training and 1 fold for testing
  3) Unless all combinations have been considered, change combination and go back to 2)

Consider the *average test loss* across all possible combinations

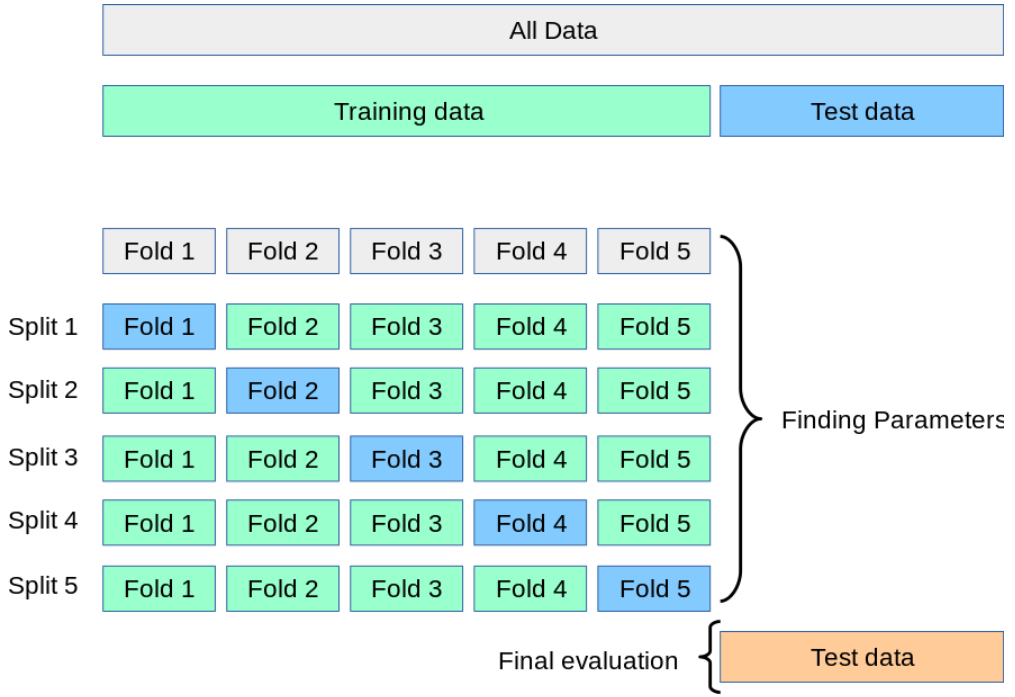


Image from https://www.kdnuggets.com/2020/01/data-validation-machine-learning.html

A More Formal Perspective

Two Kinds of Errors: Risk

Evaluation *(Mean Squared Error > Empirical Risk)*

$$L(D) = \frac{1}{N} \sum_{i=1}^{N} \left(\tilde{y}(\boldsymbol{x}^{(i)}) - y^{(i)} \right)^2$$

This what we use for optimization (a.k.a. learning)

Generalization *(Theoretical > Expected Risk)*

$$\mathcal{R}(\tilde{y}) := \mathbb{E}_{p^*} \left[(\tilde{y}(\boldsymbol{x}) - y)^2 \right]$$

This what we aim to minimize

*The (unknow) **true** probability of data items*

This is our true goal:
minimizing the risk of error we face when using our estimator on any input drawn from the true distribution p^*

The World of Data is Noisy

Assume a standard noise model (as previously done)

$$(\boldsymbol{x}^{(i)}, y^{(i)}) \implies y^{(i)} = f^*(\boldsymbol{x}^{(i)}) + \varepsilon$$
$$\text{where} \quad \varepsilon \sim \mathcal{N}(0, \sigma^2),$$
$$\text{which means} \quad \mu^{(i)} = f^*(\boldsymbol{x}^{(i)}), \ \forall i.$$

This implies:

$$\mathbb{E}_{p^*}[y \mid \boldsymbol{x}] = f^*(\boldsymbol{x})$$

$$\mathrm{Var}_{p^*}(y|\boldsymbol{x}) = \mathbb{E}_{p^*}[(y - f^*(\boldsymbol{x}))^2] = \sigma^2$$

This term is the **intrinsic variance** of the **true** population:
it is a property of the data-generating process itself
and remains constant regardless of which approximator we train

Estimators and Risk

Assume datasets are i.i.d. and unbiased

$$D \sim (p^*)^N$$

Define

$$\tilde{y}(D, \boldsymbol{\vartheta}) = \tilde{y}_D$$

as an estimator trained on dataset D

The ***Expected Risk*** *of using estimator* $\tilde{y}_{\hat{D}}$ *trained on a specific dataset* $\hat{D}$ *on any input* $\boldsymbol{x}$ *in the wild is:*

$$\mathcal{R}(\tilde{y}_{\hat{D}}(\boldsymbol{x})) := \mathbb{E}_{p^*}\left[\left(\tilde{y}_{\hat{D}}(\boldsymbol{x}) - y\right)^2\right] = \mathbb{E}_{D,\varepsilon}\left[\left(\tilde{y}_{\hat{D}}(\boldsymbol{x}) - y\right)^2\right]$$

This emphasize the coexistence of two independent stochastic processes:

- *The sampling of the dataset* $\hat{D}$
- *The intrinsic noise in the target* y

Bias and Variance

We use the notation

$$\mathbb{E}_D\left[\tilde{y}_D(\boldsymbol{x})\right]$$

for the ***expected prediction function****, namely the expected value produced for any input* $\boldsymbol{x}$ *by the ensemble of all possible estimators trained on datasets of size* N

$$\begin{aligned}
\mathcal{R}(\tilde{y}_{\hat{D}}(\boldsymbol{x})) &= \mathbb{E}_{D,\varepsilon}\left[\left(\tilde{y}_{\hat{D}}(\boldsymbol{x}) - y\right)^2\right] \\
&= \mathbb{E}_{D,\varepsilon}\left[\left(y - \mathbb{E}_D[\tilde{y}_D(\boldsymbol{x})] + \mathbb{E}_D[\tilde{y}_D(\boldsymbol{x})] - \tilde{y}_{\hat{D}}(\boldsymbol{x})\right)^2\right] \\
&\quad \cdots \\
&= \mathbb{E}_{D,\varepsilon}\left[\left(y - \mathbb{E}_D[\tilde{y}_D(\boldsymbol{x})]\right)^2\right] + \mathbb{E}_D\left[\left(\mathbb{E}_D[\tilde{y}_D(\boldsymbol{x})] - \tilde{y}_{\hat{D}}(\boldsymbol{x})\right)^2\right] \\
&= \mathbb{E}_{D,\varepsilon}\left[\left(f^*(\boldsymbol{x}) + \varepsilon - \mathbb{E}_D[\tilde{y}_D(\boldsymbol{x})]\right)^2\right] + \mathbb{E}_D\left[\left(\mathbb{E}_D[\tilde{y}_D(\boldsymbol{x})] - \tilde{y}_{\hat{D}}(\boldsymbol{x})\right)^2\right] \\
&\quad \cdots \\
&= \left(f^*(\boldsymbol{x}) - \mathbb{E}_D[\tilde{y}_D(\boldsymbol{x})]\right)^2 + \mathbb{E}_D\left[\left(\tilde{y}_{\hat{D}}(\boldsymbol{x}) - \mathbb{E}_D[\tilde{y}_D(\boldsymbol{x})]\right)^2\right] + \sigma^2
\end{aligned}$$

Bias and Variance

We use the notation

$$\mathbb{E}_D\left[\tilde{y}_D(\boldsymbol{x})\right]$$

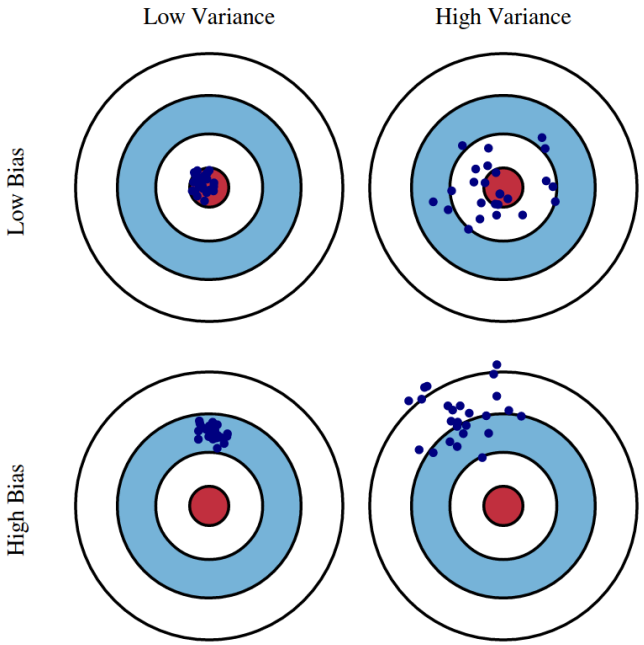
*for the **expected prediction function**, namely the expected value produced for any input $\boldsymbol{x}$ by the ensemble of all possible estimators trained on datasets of size N*

$$\mathcal{R}(\tilde{y}_{\hat{D}}(\boldsymbol{x})) = \underbrace{(f^*(\boldsymbol{x}) - \mathbb{E}_D[\tilde{y}_D(\boldsymbol{x})])^2}_{\mathrm{Bias}^2(\boldsymbol{x})} + \underbrace{\mathbb{E}_D\left[(\tilde{y}_{\hat{D}}(\boldsymbol{x}) - \mathbb{E}_D[\tilde{y}_D(\boldsymbol{x})])^2\right]}_{\mathrm{Variance}(\boldsymbol{x})} + \sigma^2$$

- $\mathrm{Bias}^2(\boldsymbol{x})$ *measures the **structural error**: how far the expected prediction function is from the true $f^*(\boldsymbol{x})$*
- $\mathrm{Variance}(\boldsymbol{x})$ *measures the **estimation error**: how much the estimator $\tilde{y}_{\hat{D}}(\boldsymbol{x})$ fluctuates around its own average due to the specific draw of the dataset $\hat{D}$*
- σ^2 *is the **irreducible error**: the intrinsic noise variance in the true distribution p^*, which no estimator can eliminate*

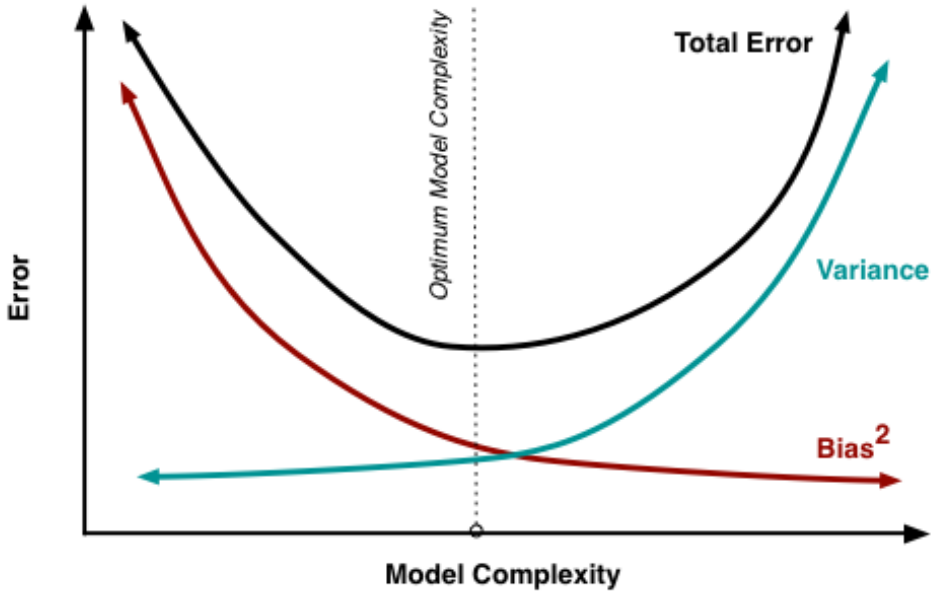
Bias and Variance

'Classical' figures need to be interpreted



These figures represents Bias and Variance as general concepts

They do not relate to the specific dataset $\hat{D}$



These are the two components of *overfitting*

*However, in what sense **model complexity** increases? (see next slide)*

[Images from http://scott.fortmann-roe.com/docs/BiasVariance.html]

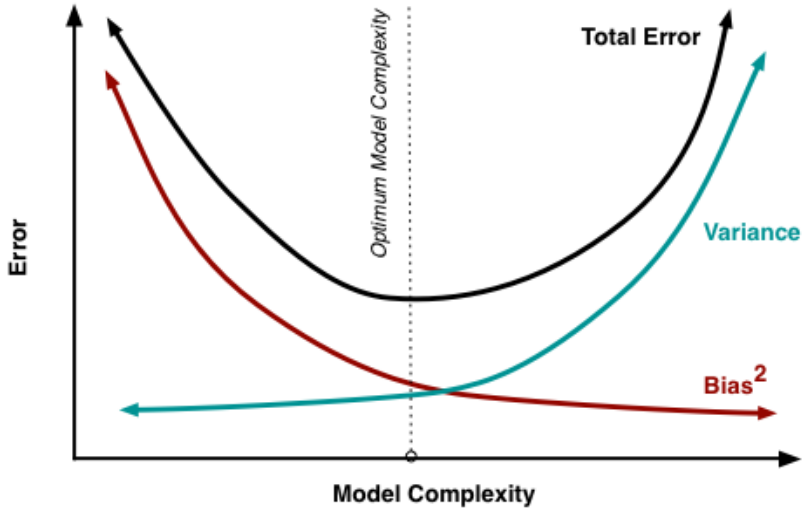
Bias and Variance

These are the two terms involved

$$\underbrace{(f^*(\boldsymbol{x}) - \mathbb{E}_D[\tilde{y}_D(\boldsymbol{x})])^2}_{\text{Bias}^2(\boldsymbol{x})} + \underbrace{\mathbb{E}_D\left[(\tilde{y}_{\hat{D}}(\boldsymbol{x}) - \mathbb{E}_D[\tilde{y}_D(\boldsymbol{x})])^2\right]}_{\text{Variance}(\boldsymbol{x})}$$

- *Bias does <u>not</u> depend on*
- *In Variance term* $\hat{D}$ *is kept fixed*

How can the model increase its <u>complexity</u>?

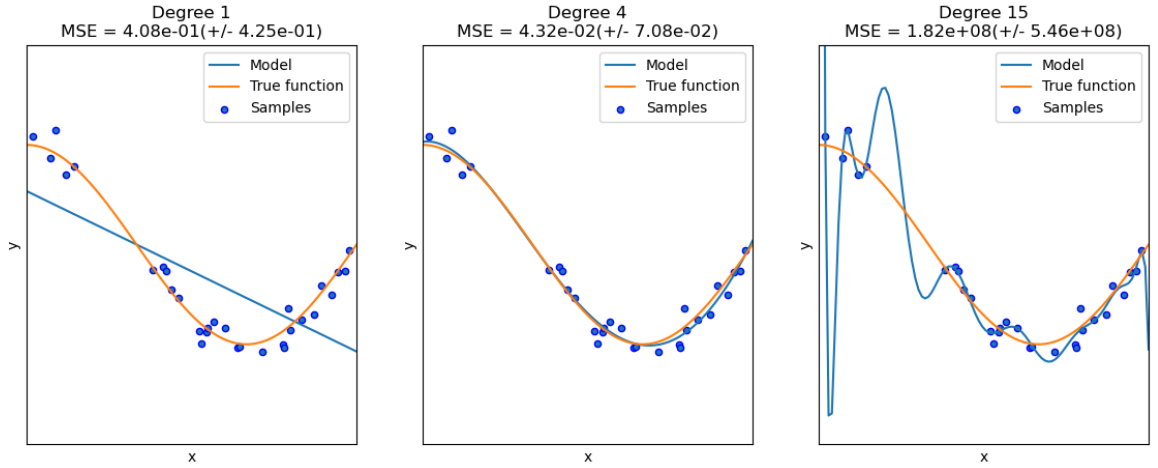


[Image from http://scott.fortmann-roe.com/docs/BiasVariance.html]

Recall that

$$\tilde{y}_{\hat{D}} = \tilde{y}(\hat{D}, \boldsymbol{\vartheta})$$

As the parameters $\boldsymbol{\vartheta}$ *change* during optimization, the estimator's capacity *increases*, allowing it to more closely fit the specific dataset $\hat{D}$



[Image from https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html]

Evaluating a Classifier

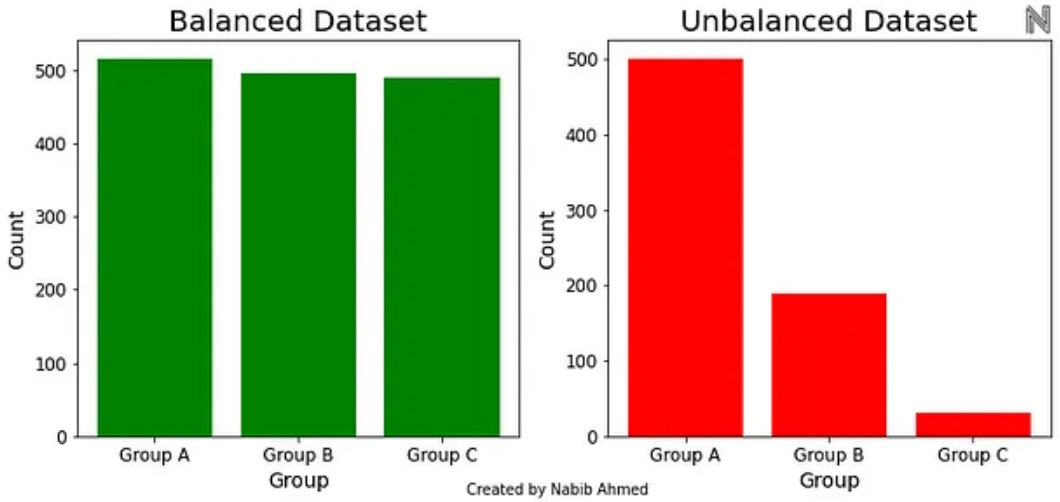
Balanced vs. Unbalanced

Sometimes, even when the dataset is unbiased

$$L(D) := \frac{1}{N} \sum_{i=1}^{N} (\tilde{y}(\boldsymbol{x}^{(i)}) - y^{(i)})^2$$

might be biased

When feature distributions
are unbalanced (in the population)
an estimator minimizing MSE
the expected error will be biased towards
over-represented classes



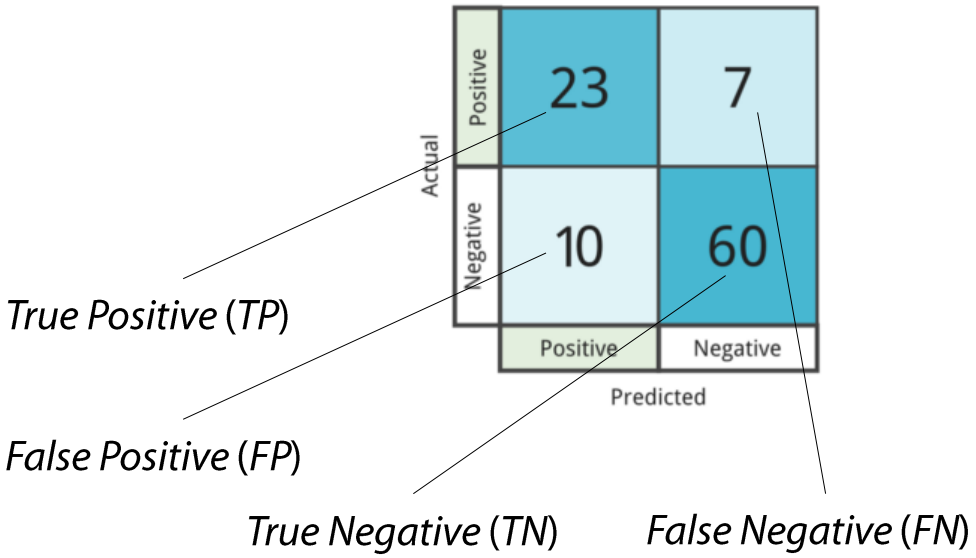
[Image from https://medium.com/@nahmed3536/data-bias-what-all-data-practitioners-should-be-aware-of-115eaeae48c]

Classifier: Confusion Matrix

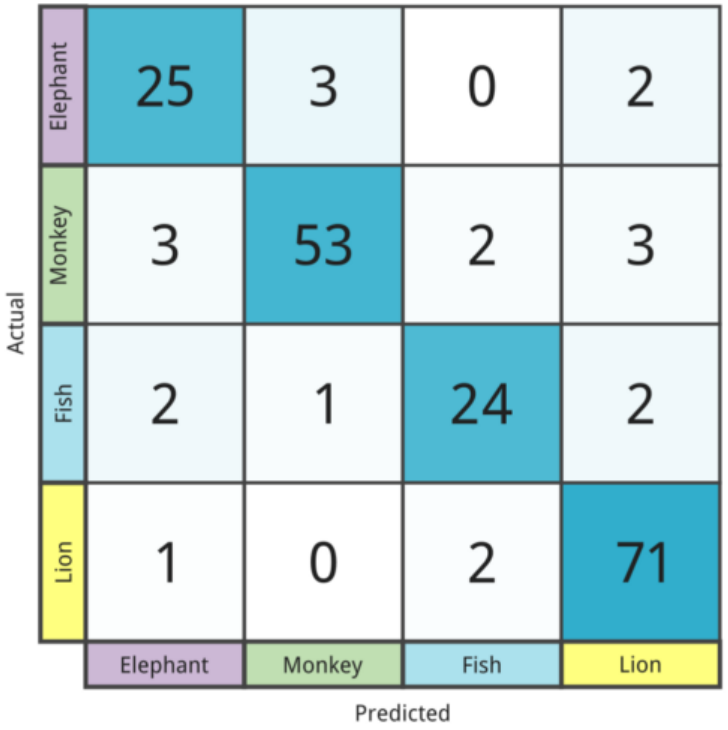
- Actual VS. Predicted Classes

  Predicted:
  the class with the highest probability

Binary

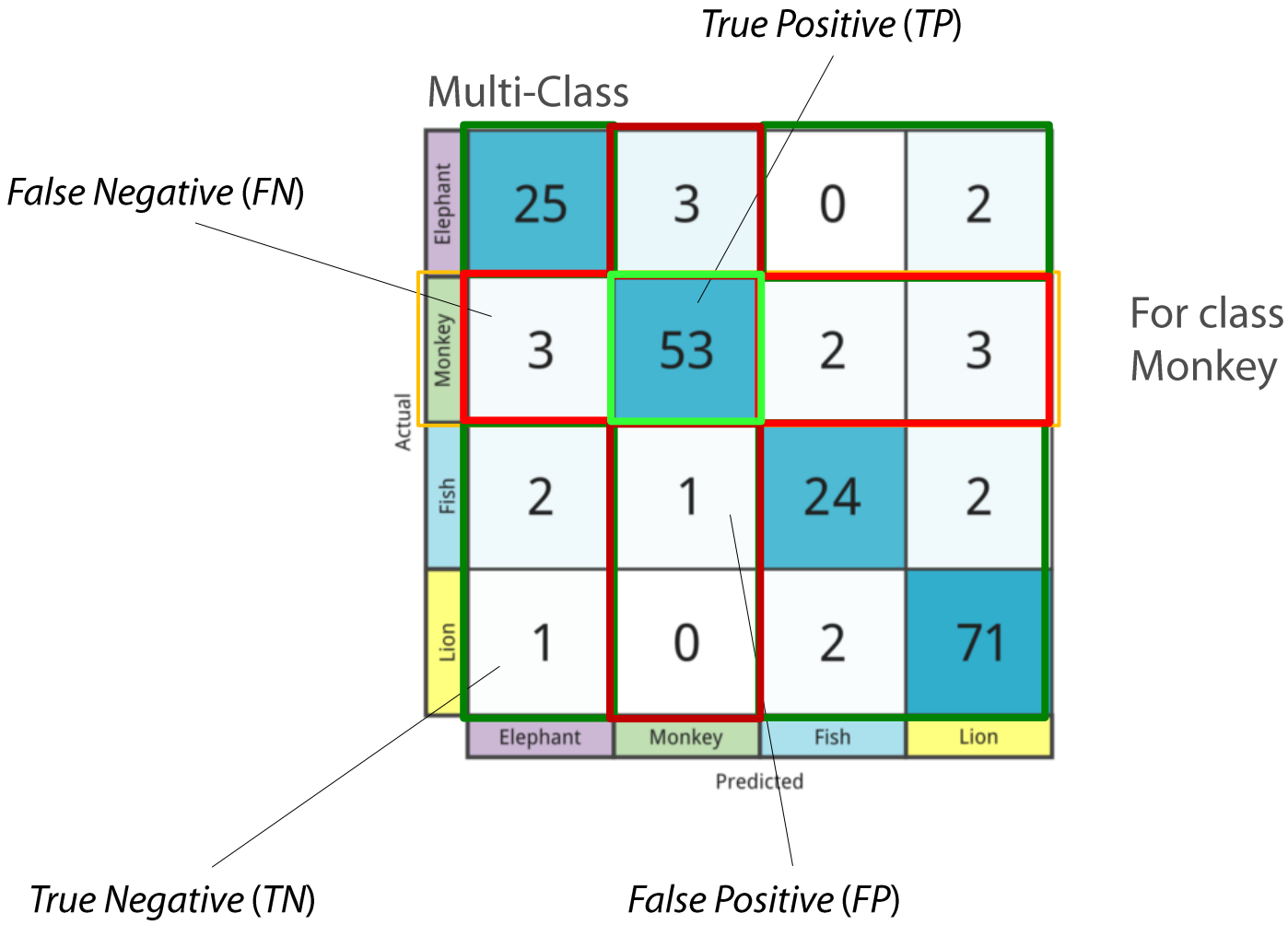


Multi-Class



Classifier: Confusion Matrix

- Actual VS. Predicted Classes



Classifier: Metrics

- **Accuracy**

$$\text{accuracy} := \frac{TP + TN}{TP + TN + FP + FN}$$

- **Recall**

$$\text{recall} := \frac{TP}{TP + FN}$$

also called 'sensitivity'

- **Precision**

$$\text{precision} := \frac{TP}{TP + FP}$$

- **F_1**

$$F_1 := 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

typically preferred when
when positive and negative cases are highly unbalanced

Receiver Operating Characteristic (ROC)

Consider a binary classifier that produces a probability distribution
Where do we put the decision threshold γ ?

as estimated by the classifier

$$y \in \{0, 1\} \qquad \tilde{y} = 1 \;\text{ if }\; \tilde{p}(y(\boldsymbol{x}) = 1) > \gamma$$

Consider the probability values:

'true' distribution

$$p(y \mid \tilde{y}) = \begin{cases} p(y = 1 \mid \tilde{y} = 1) & \text{true positive} \\ p(y = 0 \mid \tilde{y} = 1) & \text{false positive} \end{cases}$$

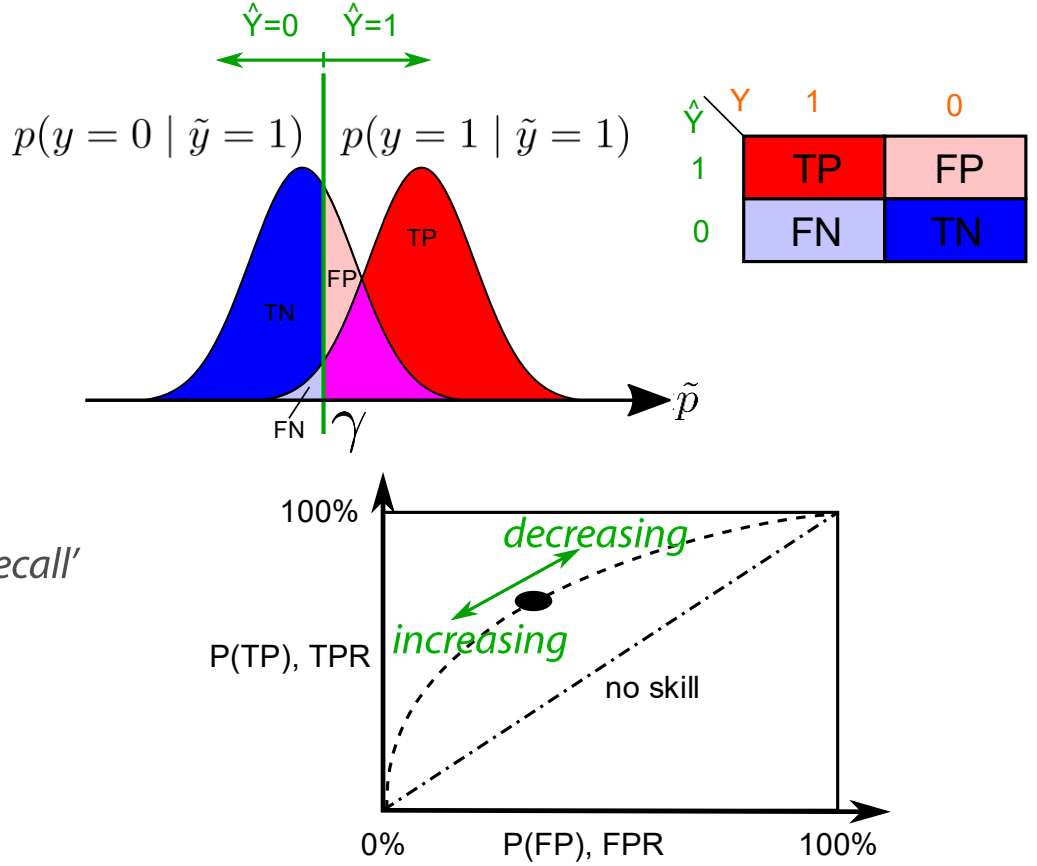
which could be estimated as

same as 'recall'

$$\text{true positive rate (TPR)} := \frac{TP}{TP + FN}$$

$$\text{false positive rate (FPR)} := \frac{FP}{FP + TN}$$

these values depend on the threshold γ we choose



[Image adapted from https://en.wikipedia.org/wiki/Receiver_operating_characteristic]

Area Under Curve (AUC)

A random classifier is right / wrong an equal number of times, regardless of γ

$$\mathrm{TPR} = \mathrm{FPR}, \quad \forall \gamma$$

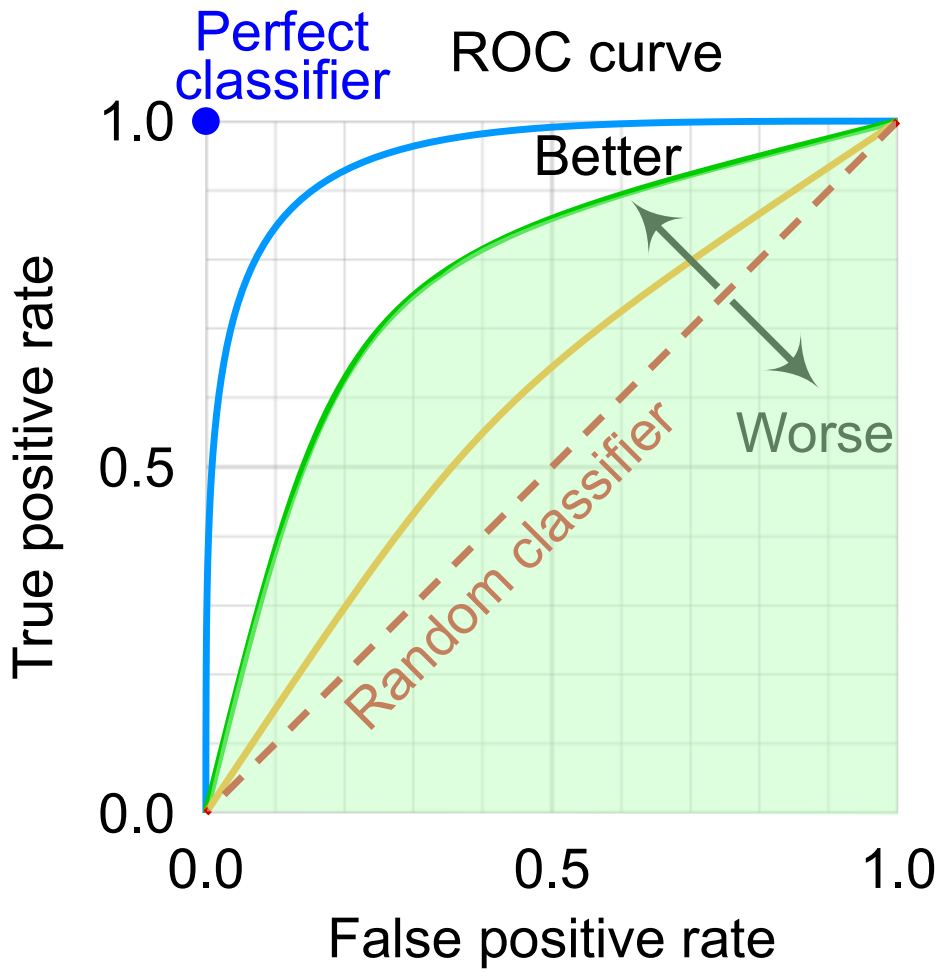
whereas a good classifier should have

$$\mathrm{TPR} \geq \mathrm{FPR}, \quad \forall \gamma$$

The Area Under Curve of the ROC measures the overall efficiency of a classifier

For a random classifier: $\mathrm{AUC} = 0.5$

For a perfect classifier: $\mathrm{AUC} = 1.0$



[Images from https://en.wikipedia.org/wiki/Receiver_operating_characteristic]