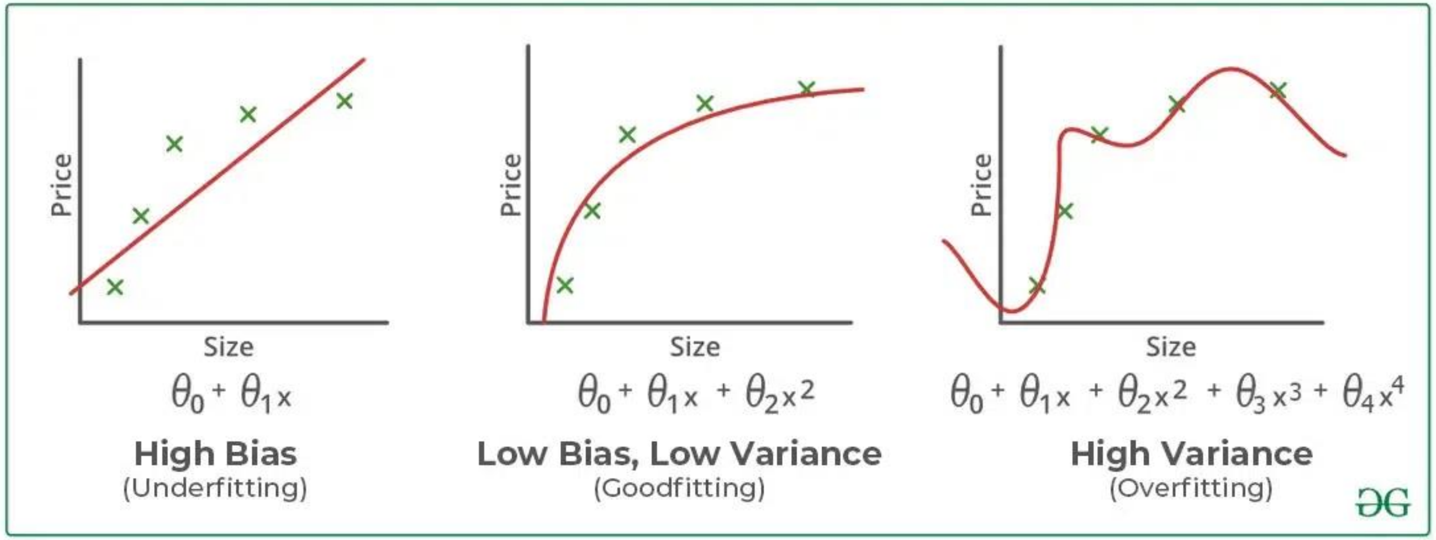
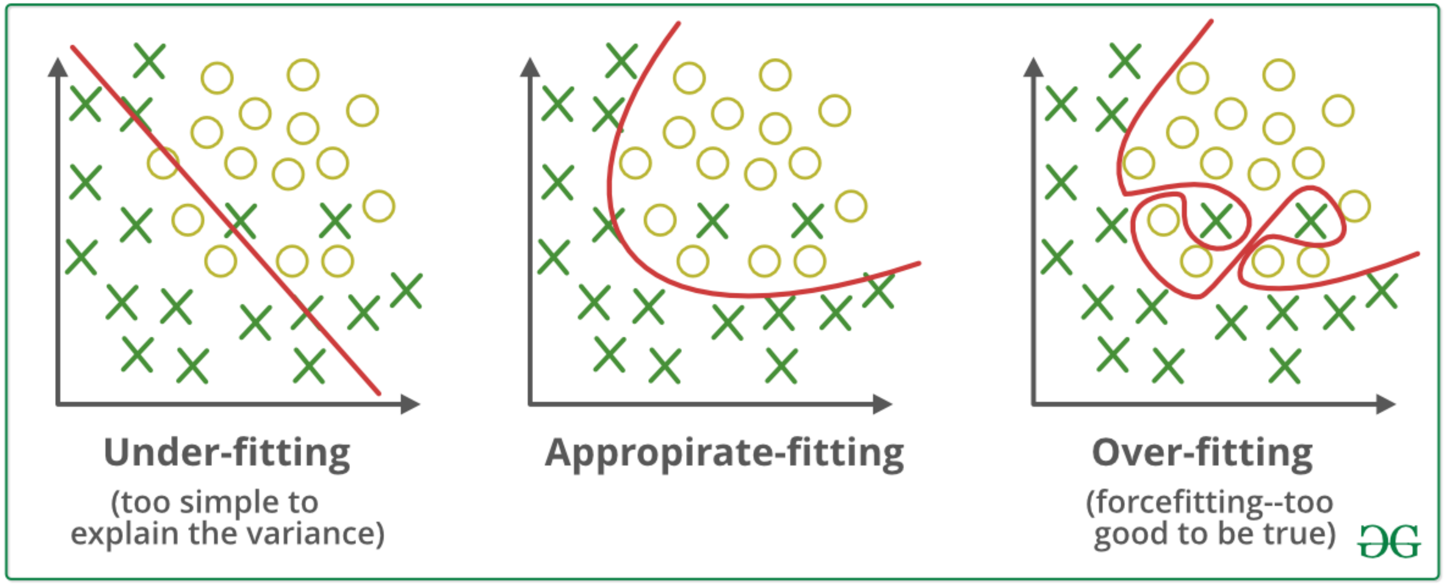
THE UNIVERSITY of TENNESSEE KNOXVILLE

Lecture 11: Testing, Hyperparameter Optimization, Bagging and Boosting

Instructor: Sergei V. Kalinin

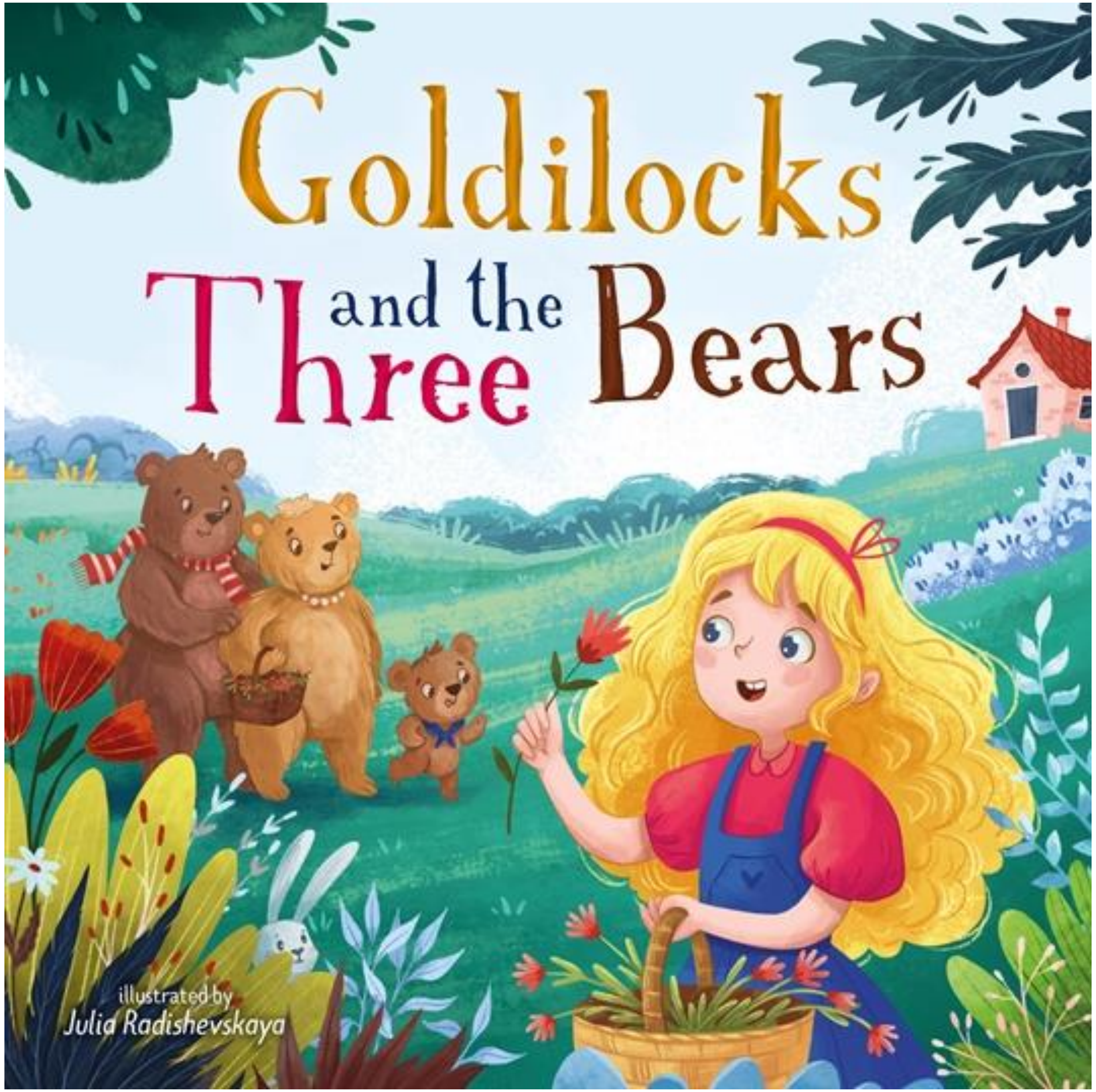
Overfitting and Underfitting





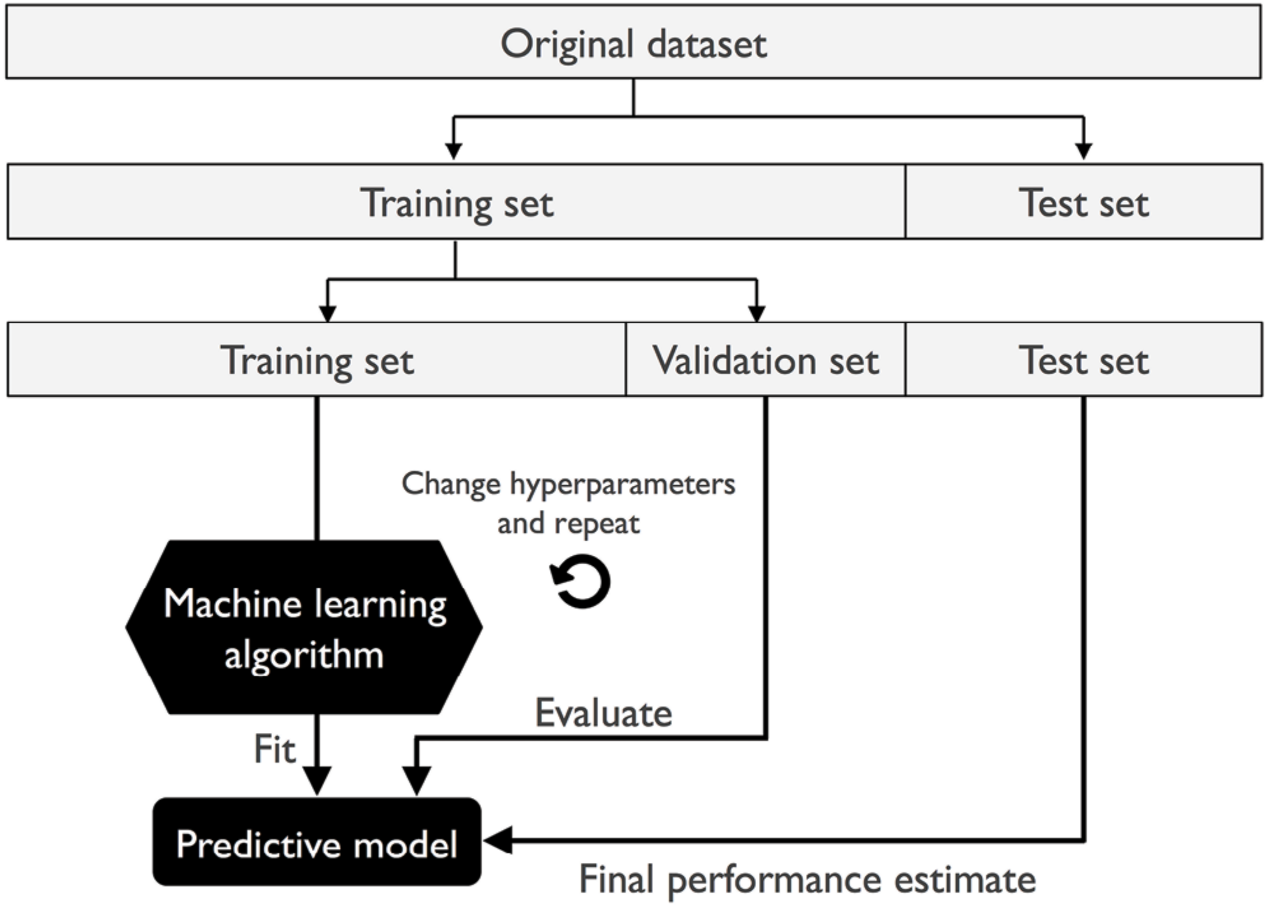
https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

Overfitting and Underfitting



Training, testing, and validating

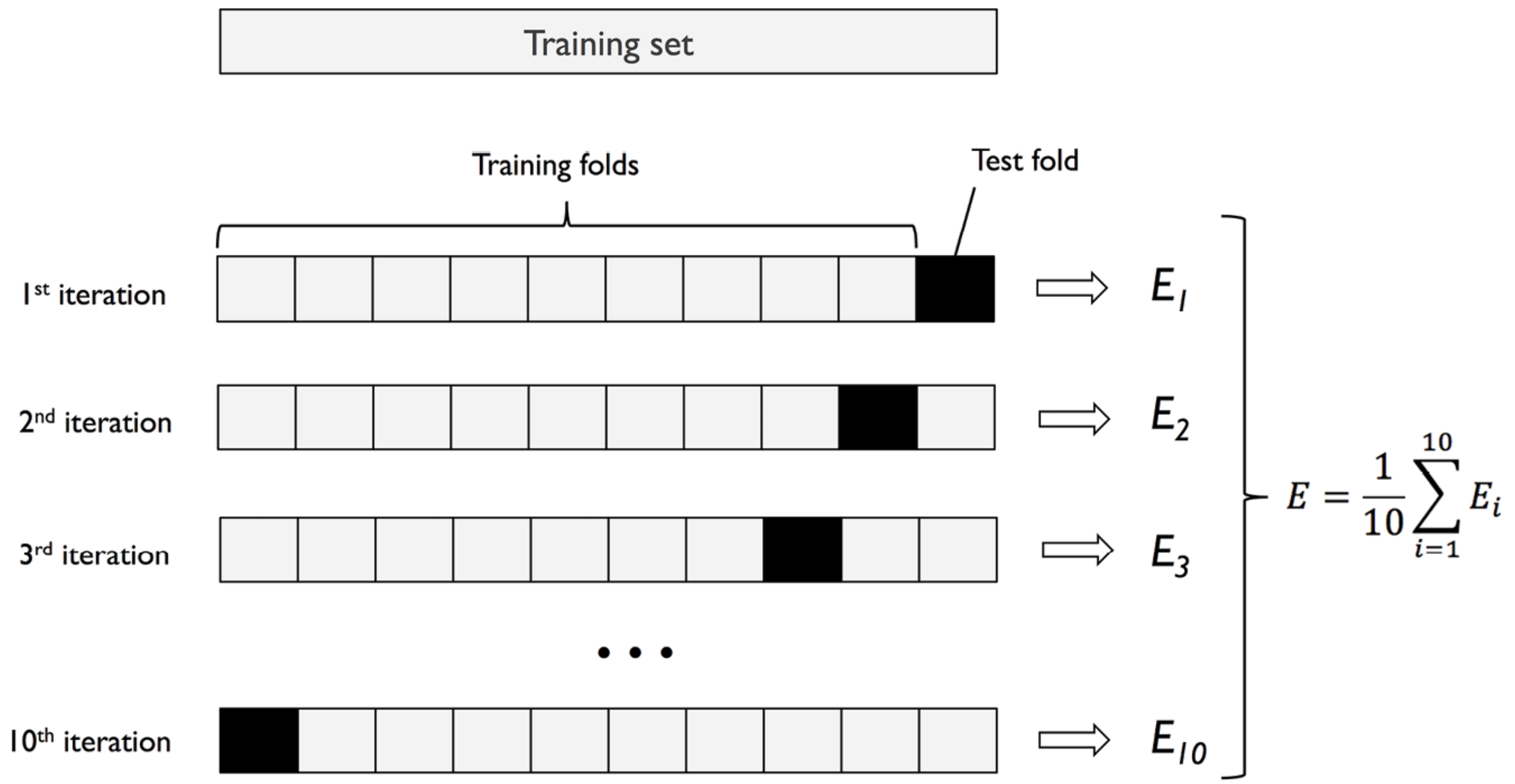
How can we be certain that model that is trained on data we have will perform well in production?



From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

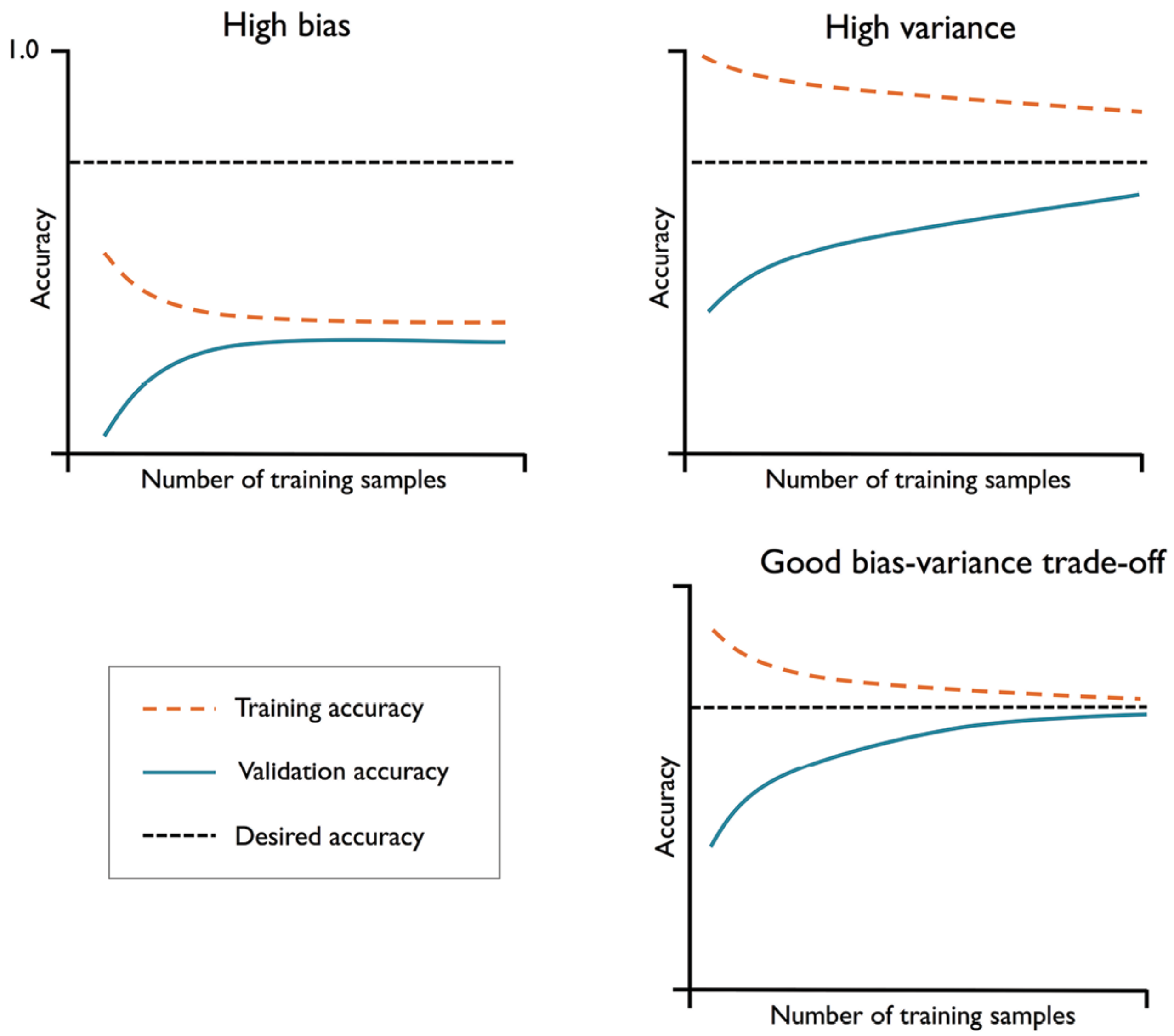
- In real world, we make decisions based on:
  - prior knowledge,
  - intuition,
  - simulations,
  - data,
  - advice
  - ....
- Now imagine we have data only!

k-Fold cross-validation



From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

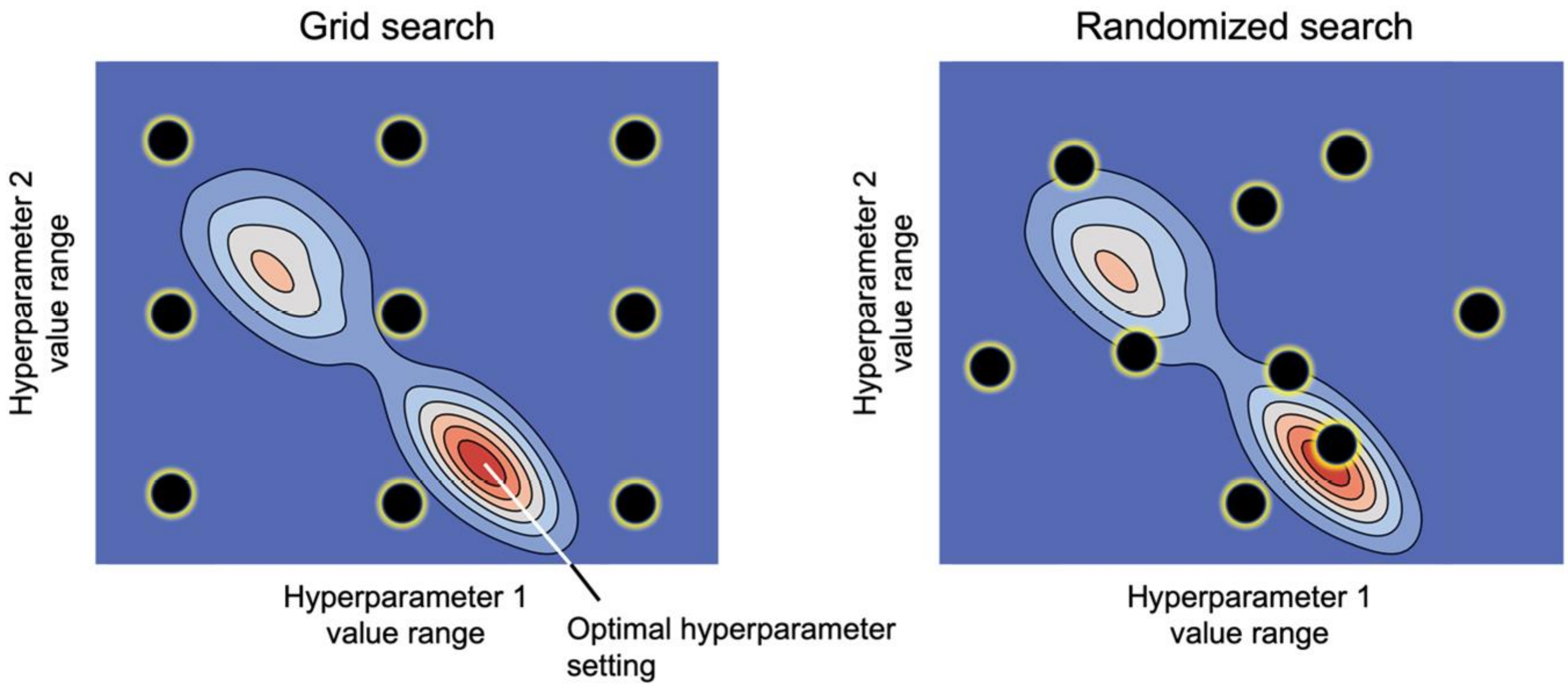
Bias-variance trade-off



From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Colab Part 1

Finding the right hyperparameters



From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Automated tuning hyperparameters

Parameter scape halving:

1.Draw a large set of candidate configurations via random sampling

2. Train the models with limited resources, for example, a small subset of the training data (as opposed to using the entire training set)

3. Discard the bottom 50 percent based on predictive performance

4. Go back to *step 2* with an increased amount of available resources

Hyperopt:

Hyperopt (https://github.com/hyperopt/hyperopt), implements several different methods for hyperparameter optimization, including randomized search and the **Tree-structured Parzen Estimators (TPE)** method. TPE is a Bayesian optimization method based on a probabilistic model that is continuously updated based on past hyperparameter evaluations and the associated performance scores instead of regarding these evaluations as independent events.

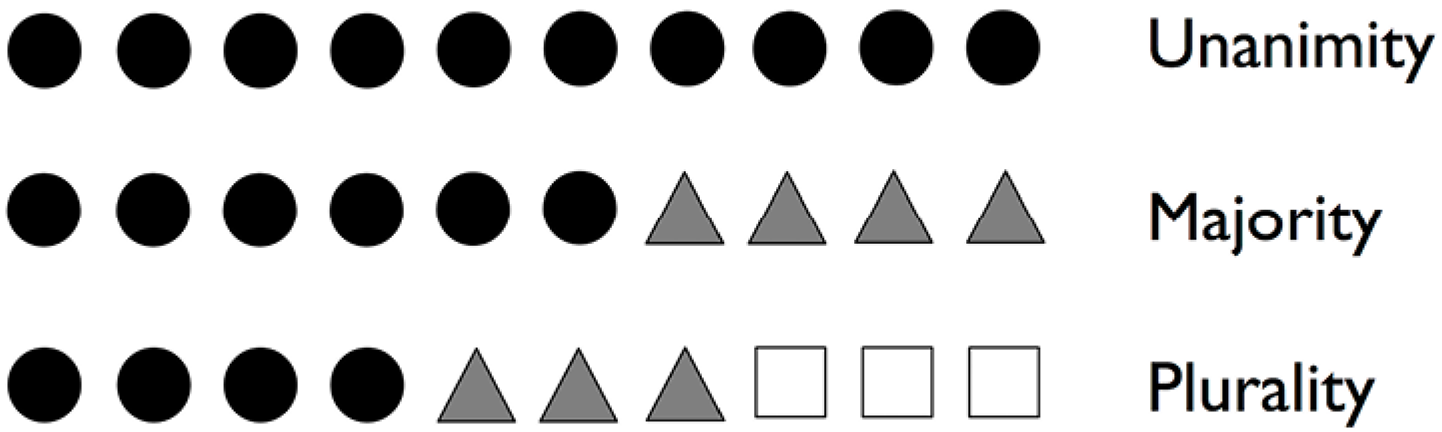
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Colab Part 2

Combining weak learners

https://en.wikipedia.org/wiki/Lemmings_(video_game)

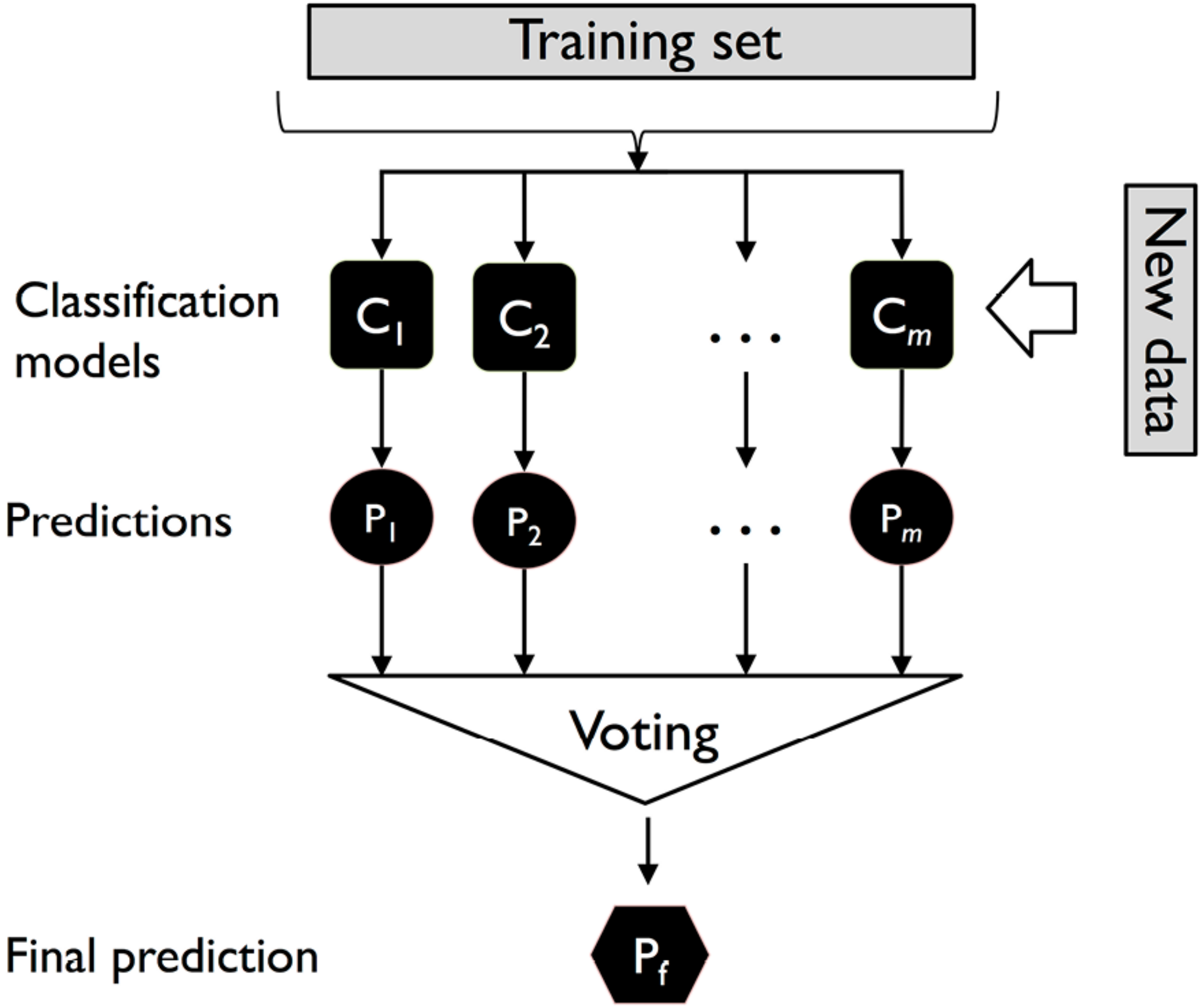
Combining weak learners



We can combine multiple weak learners into a strong learner

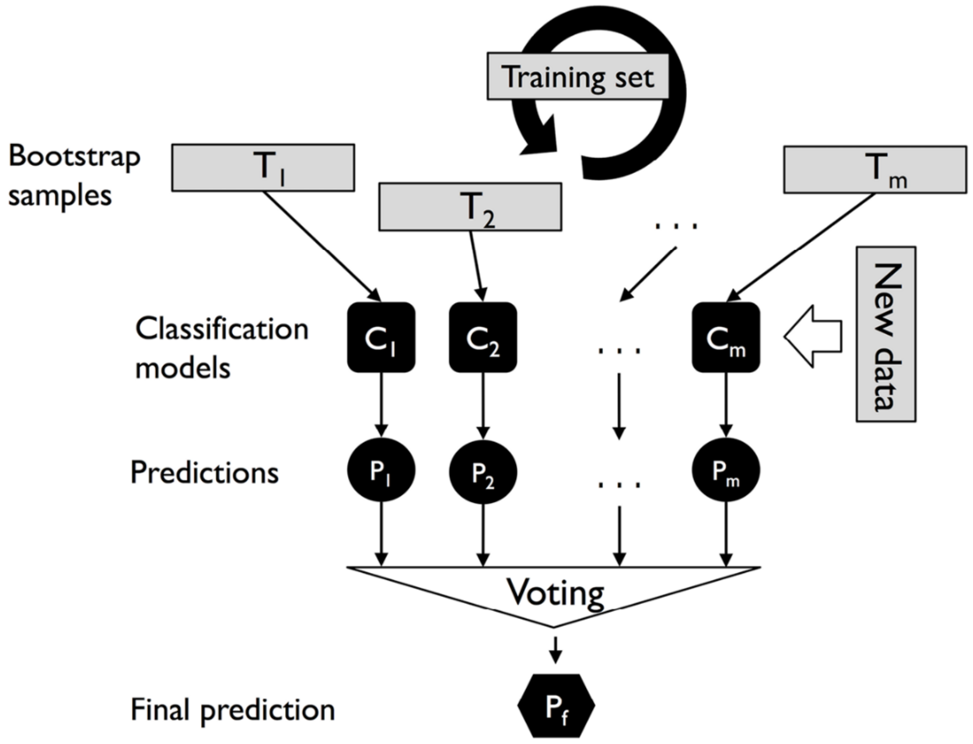
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Combining weak learners



From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

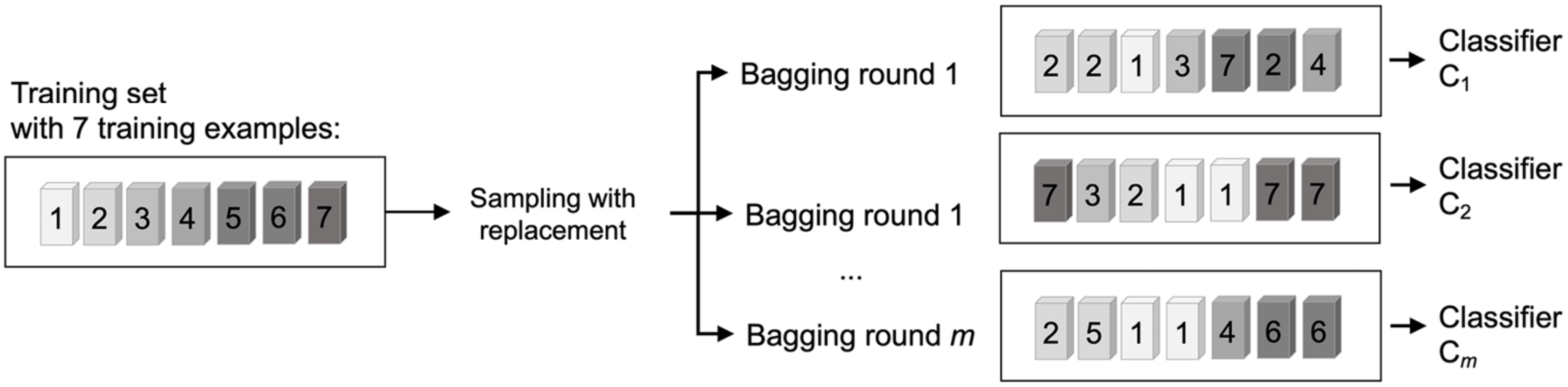
Bagging



- Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor. The aggregation averages over the versions when predicting a numerical outcome and does a plurality vote when predicting a class
- The multiple versions are formed by making bootstrap replicates (samples of the data, with repetition) of the learning set and using these as new learning sets.
- Bagging can give substantial gains in accuracy.
- Bagging can improve accuracy if prediction is unstable

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

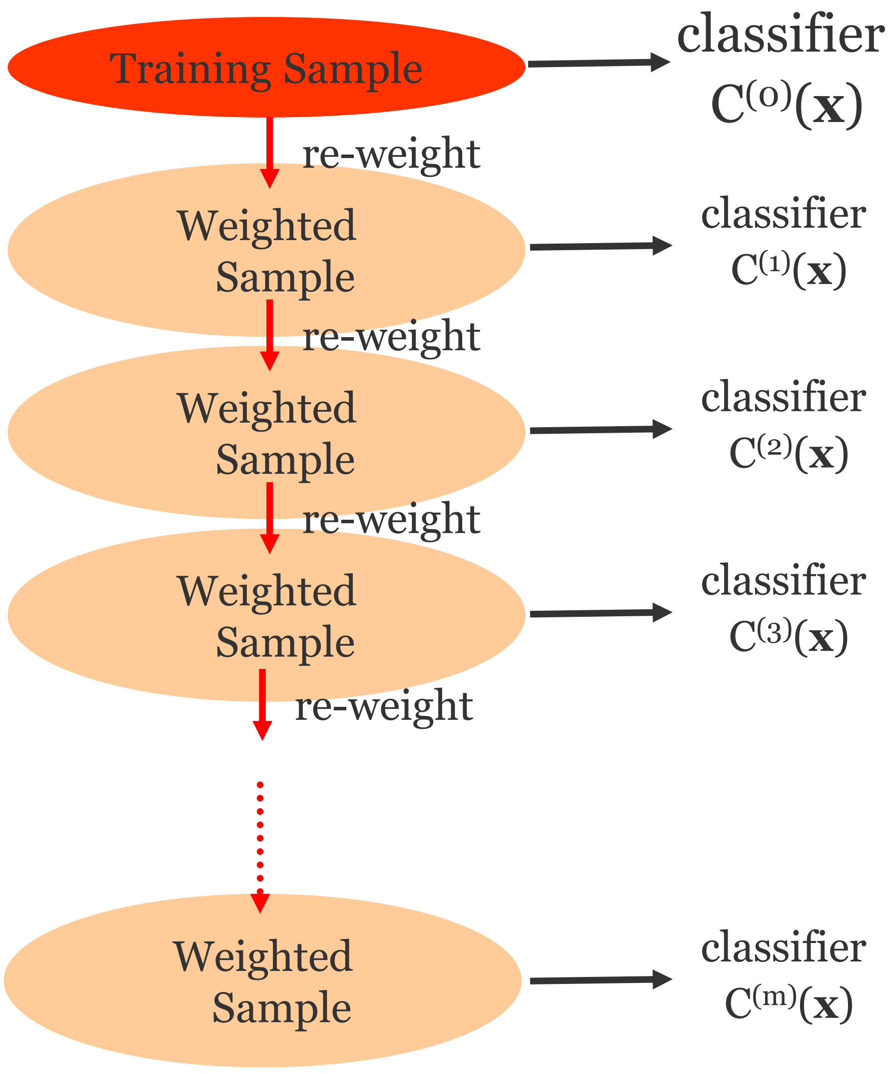
Bagging



- Draw 100 bootstrap samples of data
- Train trees on each sample → 100 trees
- Average prediction of trees on out-of-bag samples

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Adaptive Boosting (AdaBoost)



AdaBoost re-weights events misclassified by previous classifier by:

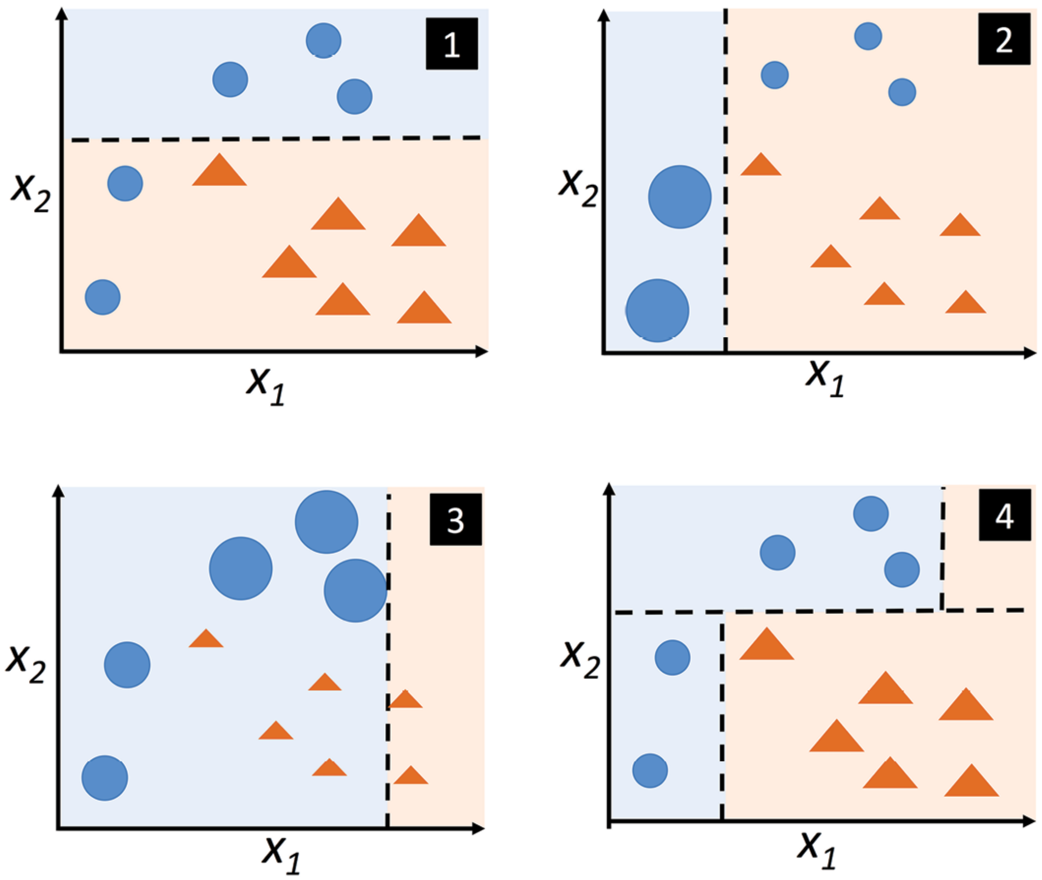
$$\frac{1-f_{err}}{f_{err}} \quad \text{with:}$$

$$f_{err} = \frac{\text{misclassified events}}{\text{all events}}$$

AdaBoost weights the classifiers also using the error rate of the individual classifier according to:

$$y(x) = \sum_{i}^{N_{Classifier}} \log\left(\frac{1-f_{err}^{(i)}}{f_{err}^{(i)}}\right) C^{(i)}(x)$$

AdaBoost



1. Draw a random subset (sample) of training examples, d_1, without replacement from the training dataset, D, to train a weak learner, C_1.

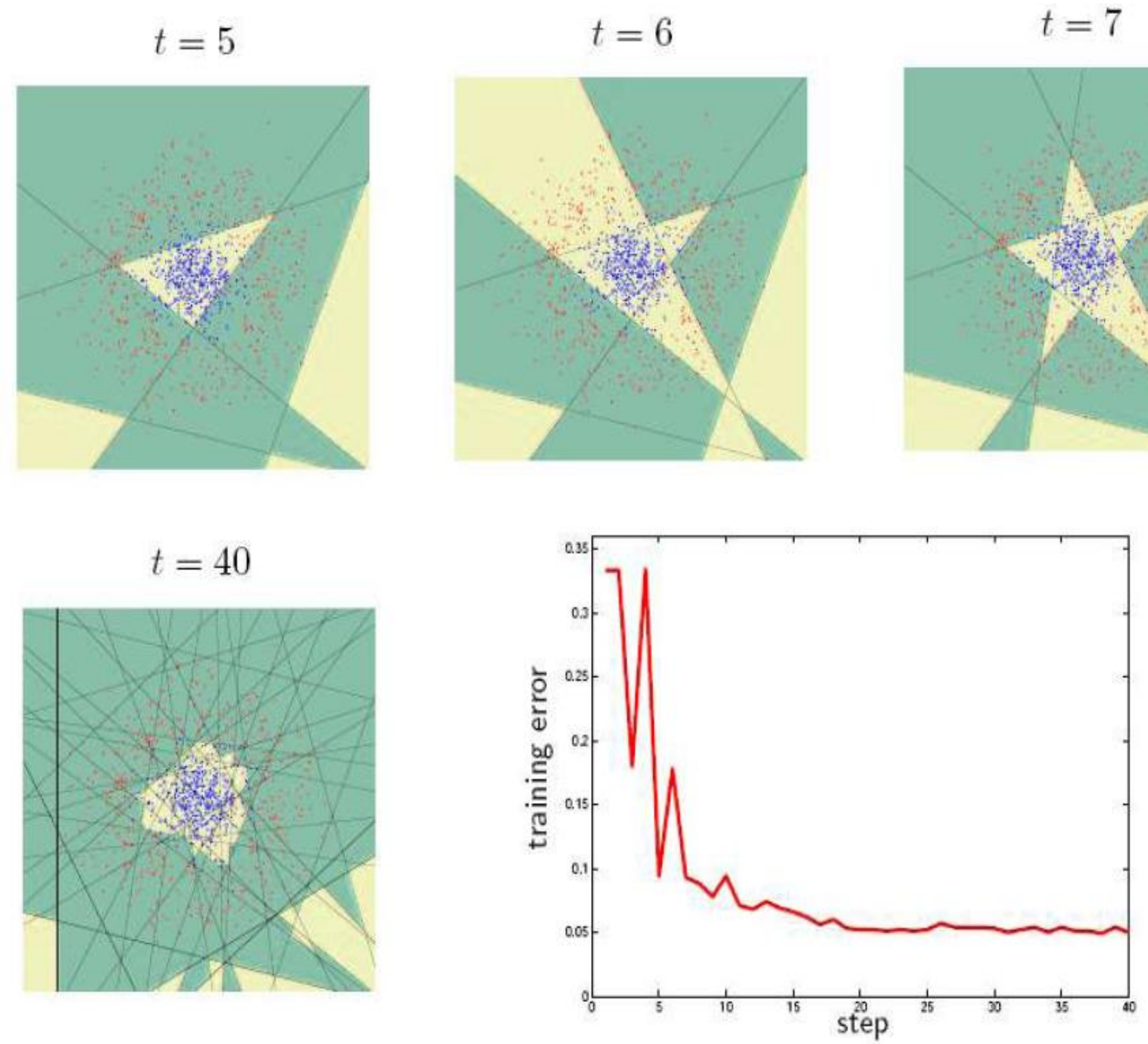
2. Draw a second random training subset, d_2, without replacement from the training dataset and add 50 percent of the examples that were previously misclassified to train a weak learner, C_2.

3. Find the training examples, d_3, in the training dataset, D, which C_1 and C_2 disagree upon, to train a third weak learner, C_3.

4. Combine the weak learners C_1, C_2, and C_3 via majority voting.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

AdaBoost On a linear Classifier



Boosted classifiers

1. Give events that are "difficult to categorize" more weight and average afterwards the results of all classifiers that were obtained with different weights

2. See each Tree as a "basis function" of a possible classifier:
   1. boosting or bagging is just a mean to generate a set of "basis functions"
   2. linear combination of basis functions gives final classifier

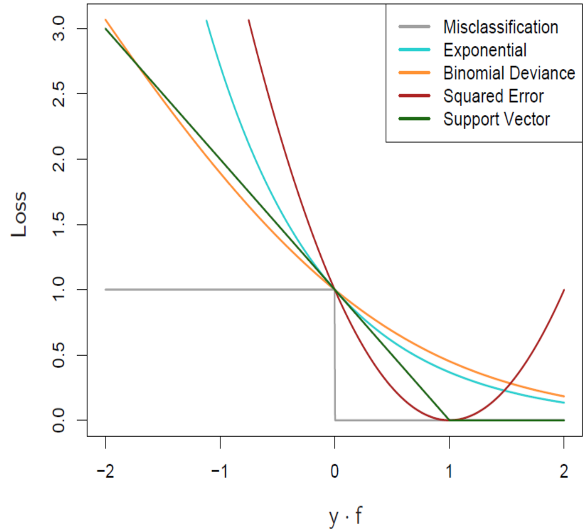
3. Every "boosting" algorithm can be interpreted as optimizing the loss function in a "greedy stagewise" manner, *i.e.* from the current point in the optimization – *e.g. building of the decision tree forest*- chooses the parameters for the next boost step (weights) such that one moves a long the steepest gradient of the loss function

AdaBoost: Exponential loss: $\exp(-y_0 y(\alpha,x))$

- theoretically sensitive to outliers

Binomial log-likelihood loss: $\ln(1 + \exp(-2y_0 y(\alpha,x))$

- more well-behaved loss function



Note of warning: you cannot do better then data

Colab Part 3