Logistic Regression

- Probabilistic linear classifier
- Logistic (sigmoid) function $f(x)=1/(1+e^{-x})$
  - Where $x = w_0 + \sum_i w_i x_i$
  - $f(x) = P(C=1 \mid X)$

- $w_0 + \sum_i w_i x_i = 0$ defines a hyperplane where $P(C=1 \mid X) = 0.5$ and $P(C=0 \mid X) = 0.5$

  and $w_0 + \sum_i w_i x_i$ is proportional to the distance from the hyperplane

- Learning
  - no closed form solution - optimization, e.g., with gradient descent
  - definition of a cost function (several options); $-y \log (y') - (1-y) \log (1-y')$; y in {0,1}
  - updating of weights (according to optimization results); $w_j = w_j - \alpha \sum_i (y'_i - y_i) x_{ij}$
    for all instances, multiple times

SVM

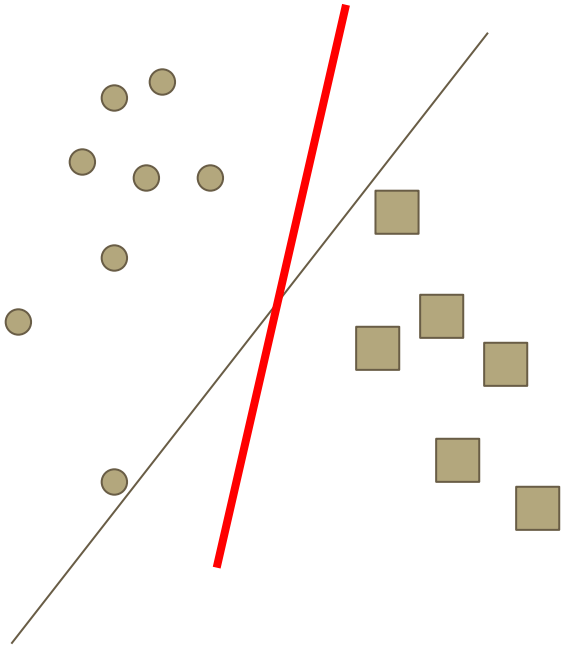
- Linear binary classifier (not probabilistic)
- Extension of linear classifiers to model non-linear decision boundaries
  - Transformation of the feature space using synthetic features of higher order

$$y' = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2 + w_5x_1x_2$$

- But this brings problems
  - Computational complexity (a lot more parameters to learn, transformation operations)
  - Overfitting
- SVM algorithm deals with these (max. margin & SV, kernel trick & SV)

SVM - max. margin

- Model (linear, *hyperplane*) for separation of data by using the *maximal margin* principle (MAX: robustness, SV: stability)



$$\overline{W} \cdot \overline{X} + b = 0$$

$$\overline{W} \cdot \overline{X_i} + b \geq 0 \quad \forall i : y_i = +1$$

$$\overline{W} \cdot \overline{X_i} + b \leq 0 \quad \forall i : y_i = -1$$

- Learning: maximal margin (optimal hyperplane) optimization problem
- Soft margin to allow misclassifications
  - Distance on the wrong side: ξ_i
  - Parameter *C* (misclassification cost) - set with experimentation!
  - Penalty: $C \cdot \xi^r_i$

SVM - kernel trick

- Use of higher dimensions for linearly non-separable data
  - https://www.youtube.com/watch?v=3liCbRZPrZA

- Learning (optimization) involves dot products in the term to maximize:

$$L_D = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \overline{X_i} \cdot \overline{X_j}$$

Dot product is needed,
(not feature values)

~similarity

classification too:

We can avoid
representing W

$$F(\overline{Z}) = \text{sign}\{\overline{W} \cdot \overline{Z} + b\} = \text{sign}\{(\sum_{i=1}^{n} \lambda_i y_i \overline{X_i} \cdot \overline{Z}) + b\}$$

SVM - kernel trick, here it is

- We do not need the feature values, just dot products
- Transformation would mean:

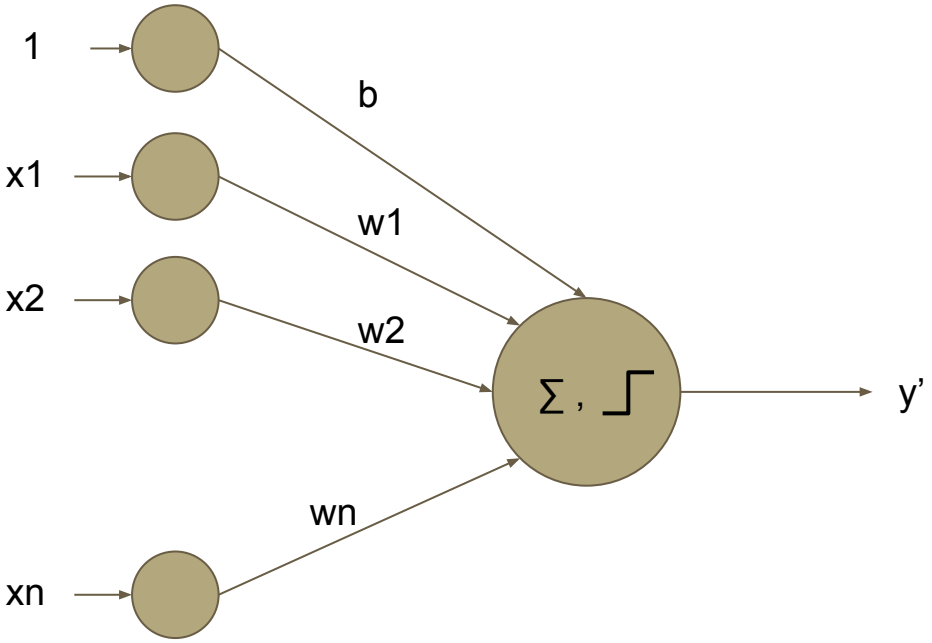
  $\Phi(x_i) \cdot \Phi(x_i)$

  calculation of transformations, then the lengthy dot products...

- Instead, we can use a function such that: $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_i)$
  - And $K(x_i, x_j)$ is in original space!
    - EXAMPLE !
  - We can only calculate kernels (polynomial, Gaussian RBF, ...)
  - Simetric, positive semi-definite; similarity ; even for strings, graphs
  - The mapping Φ can now be only implicitly used

Neural networks - perceptron

- Inspired by (simulation of) the human nervous system



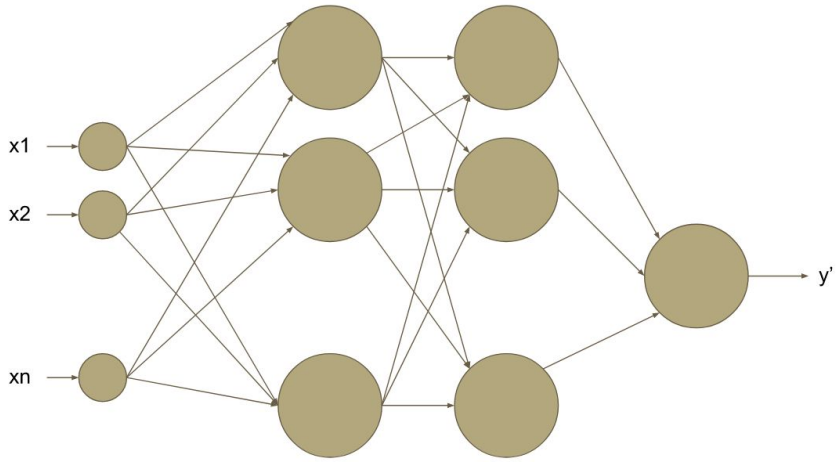
$$y' = sign(\sum_i w_i x_i + b)$$

Learning (iterative process):

- Initialize weights
- For each training item (**x**,**y**)
  - y' = f(w,x)
  - update all weights
    $w_i' = w_i + \eta(y_i - y'_i)x_i$
- Until convergence

- Can learn (converge) in linearly separable situations
- Finds (some!) linear separation

Neural networks with hidden layers



- Very powerful in capturing arbitrary functions
  - having non-linear activation functions; careful selection to facilitate learning
- Automatic generation of (higher-level) features!
  - last level is similar to logreg on generated (relevant) high-level features, not all quadratic, cubic, ... which easily go into hundreds of thousands.
- Drawbacks
  - computationally demanding learning (recently alleviated)
  - more layers - more power - more prone to overfitting
  - black-box models

Neural network - use (forward propagation)

Use of a neural network

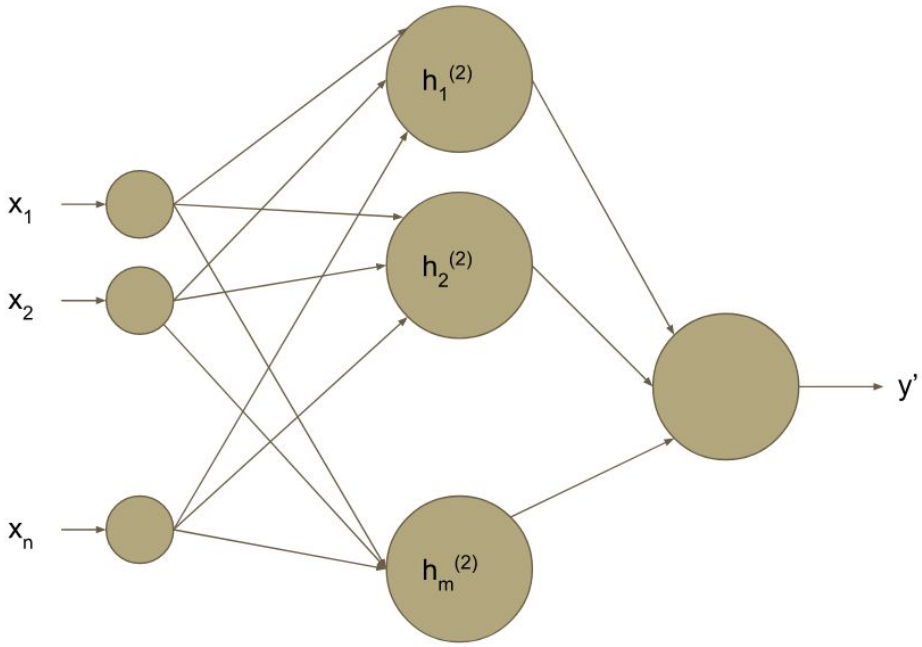
$$h_1^{(2)} = f(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + \ldots + w_{1n}^{(1)}x_n)$$

$$h_2^{(2)} = f(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + \ldots + w_{2n}^{(1)}x_n)$$

...

$$h_m^{(2)} = f(w_{m1}^{(1)}x_1 + w_{m2}^{(1)}x_2 + \ldots + w_{mn}^{(1)}x_n)$$

$$y' = f(w_{11}^{(2)}h_1^{(2)} + w_{12}^{(2)}h_2^{(2)} + \ldots + w_{1m}^{(2)}h_m)$$



Neural networks - learning

- Two things to learn:
  - Structure: expert knowledge and experimentation
  - Parameters/weights : backpropagation (and other optimization approaches)
    - Gradient descent (consequence: step → sigmoid; error 0/1 → $(y-y')^2$)
      - Optimum can be local !
    - Can be done in a batch or online mode
    - Overfitting problem - stop on check with holdout, ...
    - Computationally demanding
    - Nice explanation of the procedure in the Weka book CHECK!
      - Activation function selection was not random ;)

Ensemble methods

- Combine results of multiple classifiers
  - Different learners
  - Different training data subsets
  - Combined predictions
    - averaging
    - weighted voting
    - model of combination
- Helps tackling error components
  - Bias
    - Model assumptions (e.g., linear separation)
    - Consistently incorrect for some instances
  - Variance
    - Data variations leading to very different models (~ overfitting)
    - Inconsistently classified data
  - Noise
    - Intrinsic error in target class
    - Some algorithms are more affected, some less

Bagging

- Single learning algorithm
- *k* data samples with replacement
- *k* learned (same kind) classifiers
- Majority vote
- Reduces variance (makes sense for low bias learners, e.g. deep trees)
- Models are independent, can be built in parallel

Boosting

- Single learning algorithm
- Weighted training instances
  - Adapted learning
  - Weighted data sampling
- Iterative reweighting according to classifier performance
- Focus on misclassified instances in next iteration (increased weights)
  - Various increase approaches and termination criteria
- Aggregation of weighted (according to performance) predictions
- Reduces overall bias (to be used with simple, high bias, models)
- Reduction of variance depends on intensity of reweighting scheme
  - No reweighting in iterations == bagging
- Sensitive to noise (training can focus on bad data!)
- Models depend on previous ones, sequential process

Stacking

- Combination of predictions with another machine learned model
- Two level classification, two data subsets
- *k* classifiers (bagged, boosted or from different learners) learned on the first subset
- Their outputs on second subset are *k* new features
- Second level classifier is trained on
  - new feature space (of size *k*), or
  - combined feature space (old+new)

Random forest

- Similar as bagging with decision trees, but promotes more diverse trees
  - Decision trees in bagging tend to be similar
- Randomness at splits:
  - A random subset of attributes
  - Often advised: $\log_2(\#all\_atribs)+1$
- Usually no or minimal pruning
- Also bootstrapped data samples (as in bagging)
- Majority vote
- Efficient (less attributes considered at splits)
- Resistant to noise, outliers and overfitting

Random forest

- Similar as bagging with decision trees, but promotes more diverse trees
  - Decision trees in bagging tend to be similar
- Randomness at splits:
  - A random subset of attributes
  - Often advised: $\log_2(\#all\_atribs)+1$
- Usually no or minimal pruning
- Also bootstrapped data samples (as in bagging)
- Majority vote
- Efficient (less attributes considered at splits)
- Resistant to noise, outliers and overfitting

Gradient boosted trees

- Construct a (regression) tree and let the residuals (y-F(x)) become a new target for another model in the iteration
- Next model learns the residuals of the first one
  - Using original features and the new target
- Taking the two models together we get a better prediction
- Repeat *m* times, until stopping criterion...
- In fact: a gradient of a differentiable loss function is usually modelled instead of actual residuals (added parameter: step size)
- Sensitive to noise
- Sequential (cannot run in parallel)

Gradient boosted trees

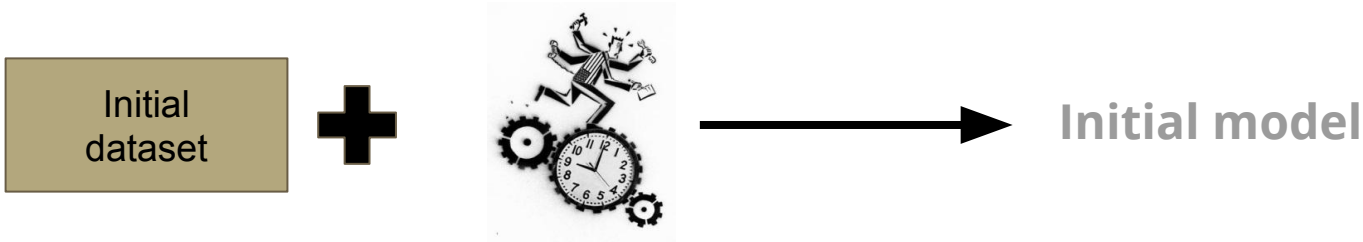
- Construct a (regression) tree and let the residuals (y-F(x)) become a new target for another model in the iteration
- Next model learns the residuals of the first one
  - Using original features and the new target
- Taking the two models together we get a better prediction
- Repeat *m* times, until stopping criterion...
- In fact: a gradient of a differentiable loss function is usually modelled instead of actual residuals (added parameter: step size)
- Sensitive to noise
- Sequential (cannot run in parallel)

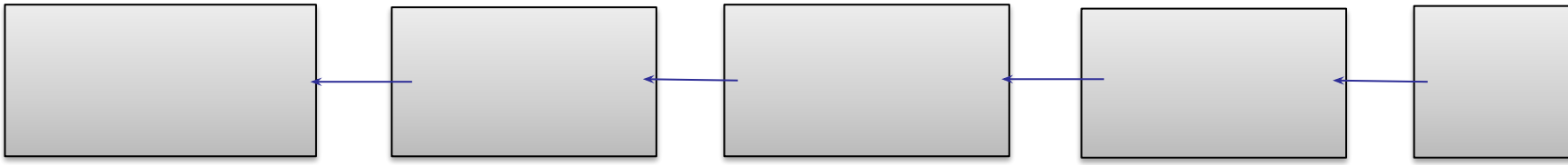
→ **ESA challenge example**

Active learning

- Labels are sometimes hard or expensive to get
  - Time restrictions in dynamic settings
- AL aims at getting the most of information with the least amount of labels
- Components
  - Querying system : selects the instances to be labelled
  - Oracle : provides labels
- Querying strategies
  - Highest uncertainty regions (danger: querying in low data quality areas)
  - Expected error of variance reduction
  - Representativeness
    - Equal representativeness of regions (weighted by density)
  - And many others and their mixtures...

Active learning

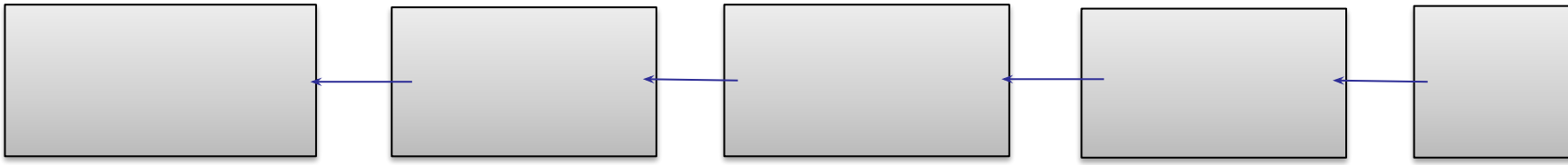


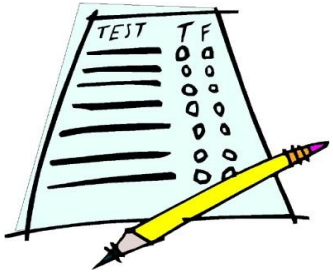


Active learning

Initial dataset

Initial model



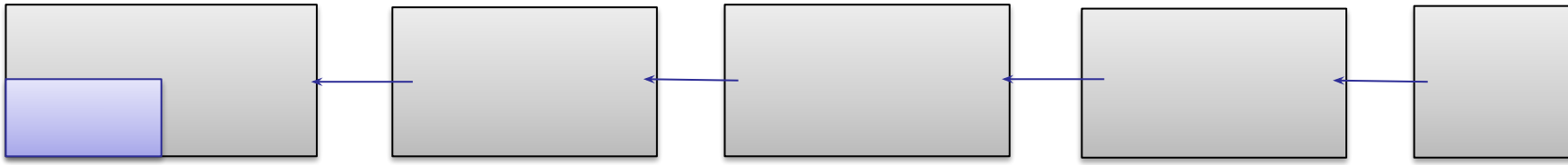


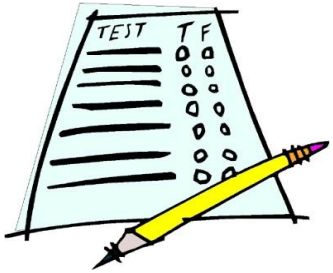
Initial model

Active learning

Initial dataset

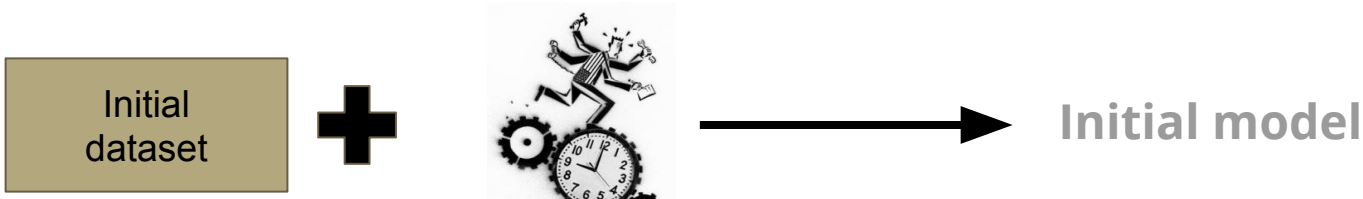
Initial model

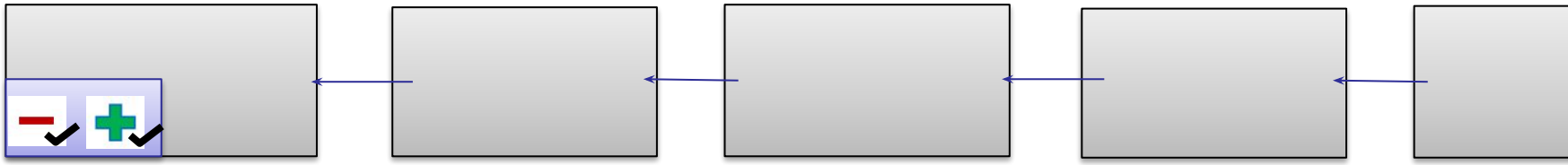


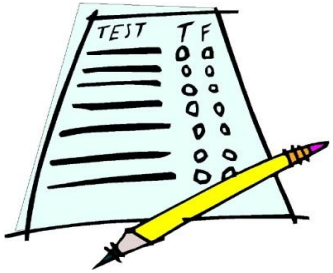


Initial model

Active learning

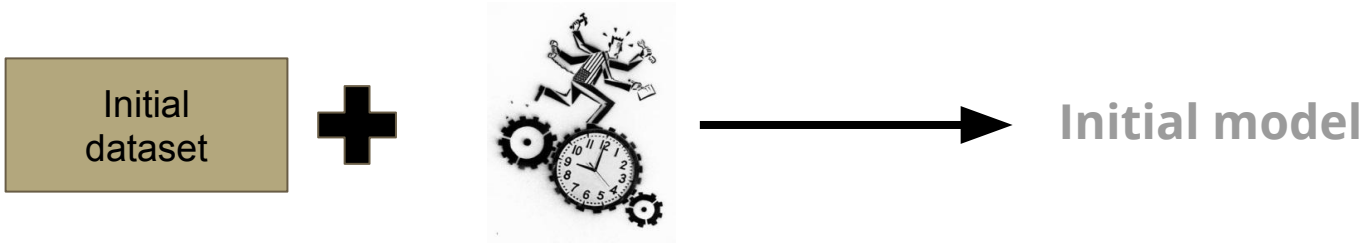


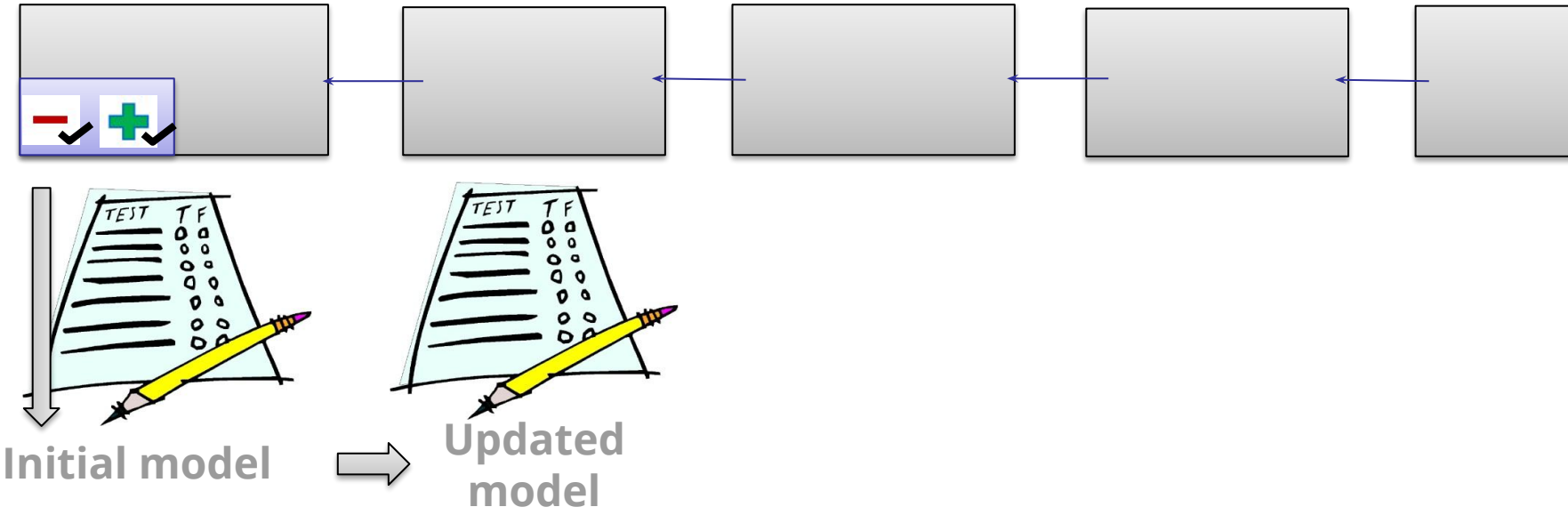




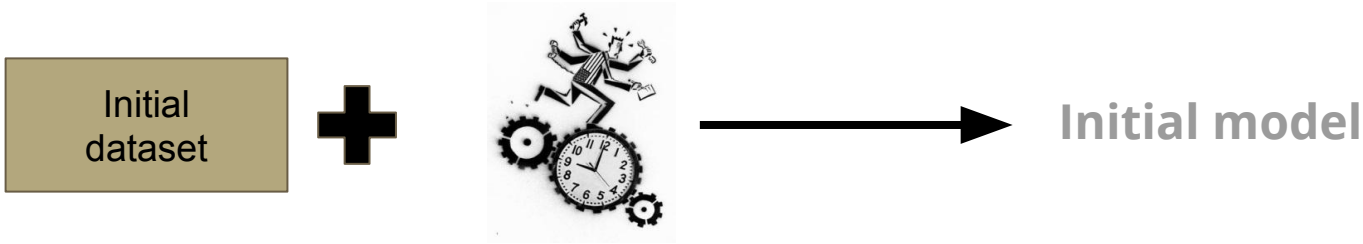
Initial model

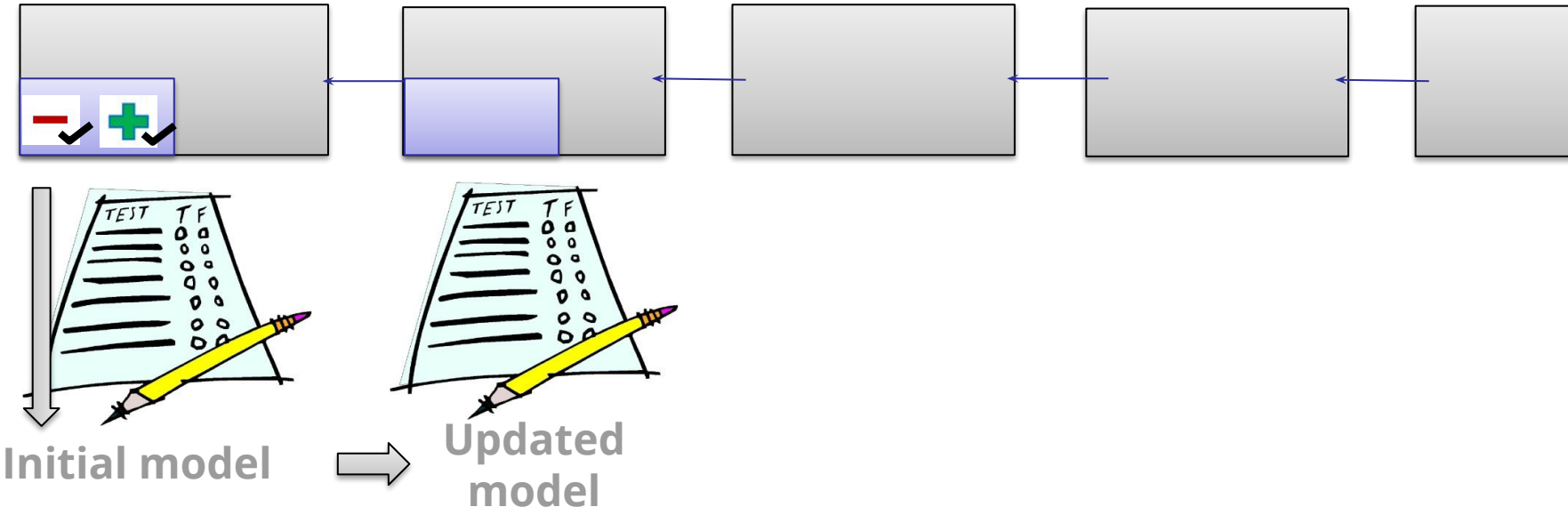
Active learning



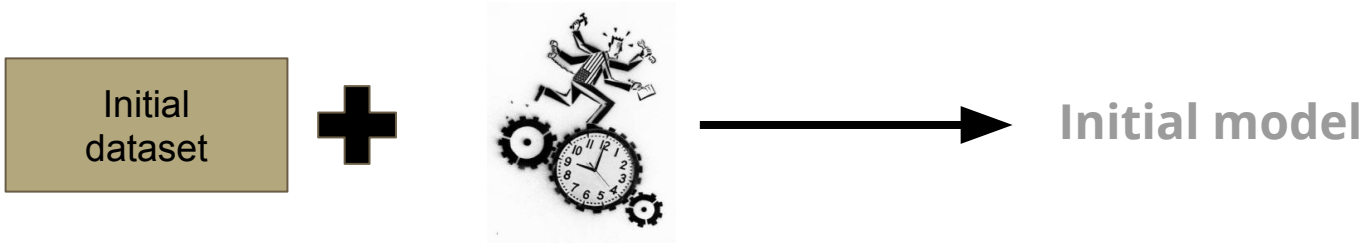


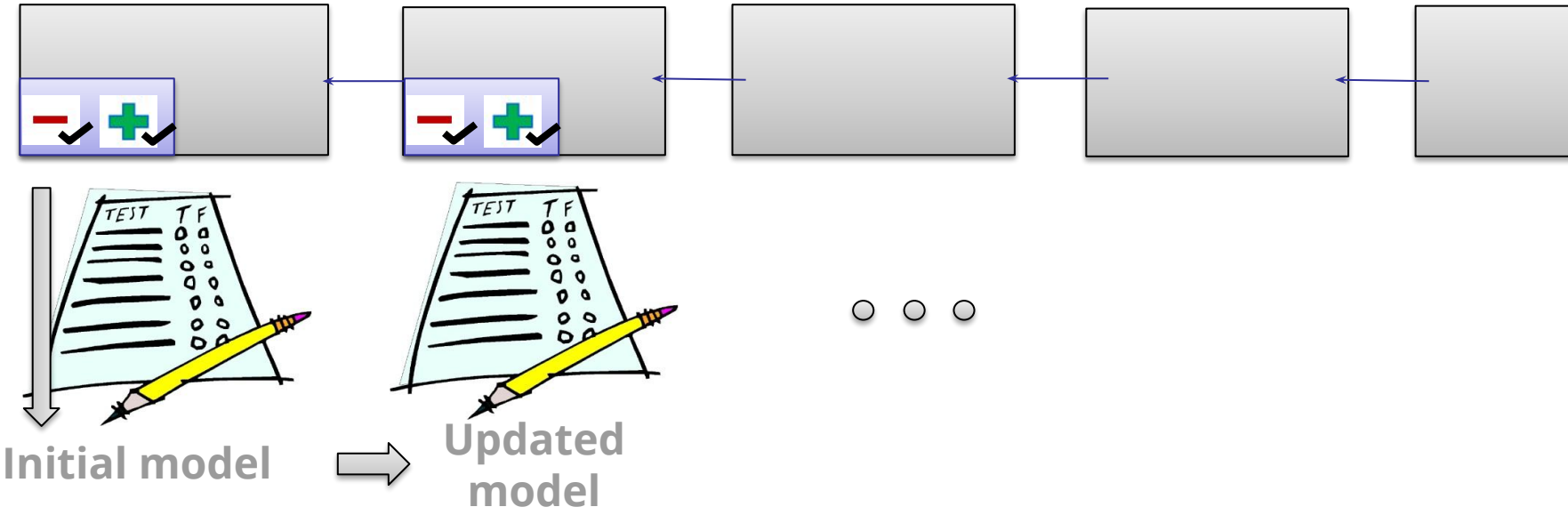
Active learning





Active learning





Active learning