error analysis of a classifier

- exact analytical results are usuall impossible for real world problems
- asymptotic results for $N \to \infty$ can be derived [may be unrealistic for small N]

worst case bounds for fixed N ("how bad can it be, when we got an unfortunate training set")

[often too pessimistic, because worst case rarely happens]

- empirical error estimation: make experiments with the trained algorithm (i.e. algorithm is now fixed)

generalization error for new data ($\hat{=}$ test error) $\Rightarrow$ average over all possible data

$$Err = \mathbb{E}_{x,y \sim p^*(x,y)} \left[p\left(\underbrace{error(x,y)}_{\hat{f}(x) \neq y} \;\middle|\; \underbrace{\text{fixed algorithm}}_{f(x) \text{ after training}} \right) \right]$$

training tries to minimize the <u>training</u> error

$$err = \mathbb{E}_{x,y \sim TS} \left[p\left(error \;\middle|\; \underbrace{\text{current alg}}_{\text{change the alg as long as err reduces}} \right) \right]$$

$\Rightarrow$ in general $err < Err$, sometimes $err \ll Err$

because we made err as small as possible ("training")

$Err - err =$ "optimism" of our predictor

big optimism $\hat{=}$ over-fitting (extreme case: memorize training data, but has no idea about new data)

we need to ensure that Err is small enough for the predictor to be applicable in practice four levels of empirical error analyses:

① (luxury solution, for very critical applications): defines test procedures independently of training data and algorithms, example: certification for autonomous cars

VDA Leitinitiative ~~www.vdat~~ www.vdali.de

② (very good, for scientific benchmarks): benchmark website, which offers TS for some problem, and ~~the~~ a set of test X's (Y's secret)

- people ~~of~~ train their algorithm on TS and apply to test set, upload $\hat{Y}$
- benchmark managers compare $\hat{Y}$ with secret Y^* $\Rightarrow$ leader board

example: CREMI Neuro segmentation Challenge

very popular in the life sciences (medicine, biology, neuroscience, ...)

③ ~~m~~ (standard) provide TS and test sets with revealed ground truth

$\Rightarrow$ use to design better algorithms (people could cheat, but rarely do)

example: MNIST data set (handwritten digits by US Postal service for automated zip-code detection/reading)

④ (fall back: if you do not have a test set): split ~~the~~ TS into a training and a test subset, clever version <u>m-fold cross-validation</u>

alg: ⓪ split the TS into M subsets of size $\frac{N}{M}$ (roughly)
"folds"

① for each $m = 1 \ldots M$

in every iteration, train from scratch on TS \ fold_m

ⓐ use subset m as <u>test set</u>, train on remaining data

$\Rightarrow$ test error ~~as~~ $\hat{Err}_m = \frac{\#\text{errors}}{N_m}$ ← size of subset m

② compute test error $\boxed{\hat{Err} = \frac{1}{M} \sum_{m=1}^{M} \hat{Err}_m}$

Improvements for the Nearest Neighbor classifier

- for large data sets (N big) and/or high-dimensional features (D big), NN is slow
  complexity of "naive" implementation: $O(N \cdot D)$ steps
    - distance comp. takes D steps (look at every feature)
    - finding the neighbor N steps

  reduce N and D

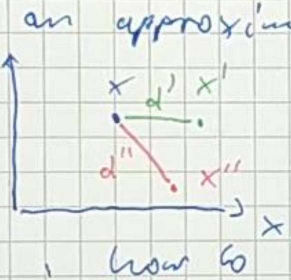
  ① reduce D:
  - feature selection: drop all features that are not very influential on the outcome
    - backward:
      - start with all features
      - try, how much results worsen, when each feature is dropped in turn $\Rightarrow$ D-times cross-validation
      - remove feature with lowest performance drop
      - repeat (drop as many features as possible)
    - forward:
      - start with the "best" single feature
      - add features one at a time, until good performance
  - feature computation: $\tilde{x}_i = f(x_i)$ $\quad f: \mathbb{R}^D \to \mathbb{R}^{D'}$ $\quad D' < D$
    - by hand: BMI (1-dimensional) $= f(\underbrace{\text{weight, height}}_{\text{2-dimensional}})$
    - learn informative features $\Rightarrow$ later (chapter "unsupervised learning")

  ② reduce N: only keep very informative / typical training instances:
  - exact algorithm: compute the Voronoi diagram of all training data, $O(N^{D/2})$
    - prediction (position of decision boundary) unchanged, if drop instance ~~with~~ not at boundary

- greedy approximation: • maintain active set of instances, initially empty
  • for each $i = 1, \ldots, N$: if instance i is incorrectly classified by the current active set => add (x_i, y_i) to active set, otherwise, drop it
- clustering algorithm: - group similar instances together ("cluster")
  - choose one representative per cluster

} later in chapter "unsupervised l."

- relax criterion: choose an approximate nearest-neighbor



$$d' \le d'' \le \begin{cases} d' + \varepsilon & \text{bounded additive error} \\ d'(1+\varepsilon) & \text{multiplicative error} \end{cases}$$

large field of research, "approximate NN search": how to guarantee that x'' is not too far off and the search is much faster than $O(N)$

- clever data structure(?): in 1-D, a binary search tree can find an instance in $O(\log N)$ time

  generalization to higher dimensions: k-D tree (or variants thereoff)

  search complexity: $O(D \cdot \min(\underset{\uparrow \text{ sequential search}}{N}, 2^D \underset{\uparrow \text{ tree search}}{\log N}))$

  in practice: only ~~worthwhile~~ faster for $D \le 20$

  improved variants $D \le 30 \ldots 50$

- two-stage procedure: - use approximate alg. to filter out far-neighbors
  e.g. locality-preserving hashing
  - use exact alg. on remaining near neighbor candidate set
  $O(N')$ $N' \ll N$

- nearest neighbor alg. is <u>not</u> consistent, ~~e.g.~~ i.e. does not converge to the optimal Bayes classifier even when $N \to \infty$

⇒ make better use of the TS via $\underline{\text{k-nearest neighbor alg.}}$
M-nearest neighbor alg.

- instead of looking for nearest neighbor, find the M nearest neighbors
- count fraction of class k instances in this set $\hat{p}_k = \frac{M_k}{M} = \hat{p}(Y=k \mid X)$

M_k : # of $Y=k$ in M-neighbor set

- return the vector of posteriors $(\hat{p}_1, \ldots, \hat{p}_C)$ or $\hat{Y} = \arg\max_k \hat{p}_k$

theorem: M-nearest neighbor rule is consistent if $\underline{M \to \infty}$, as $N \to \infty$,

ensures that $\hat{p}(y|x) \to p^*(y|x)$

and $\underbrace{\frac{M}{N} \to 0}$ as $N \to \infty$ (ex: $M = \log N$)

ensures that the radius of the M-neighbor set in feature space → 0, i.e. converges around query point X

- nearest neighbor alg. only makes sense, if $d(X, X')$ distance for neighbor search represents the semantics of our application: if $d(X, X') \approx 0$ ⇒ events X, X' should be similar ~~for~~ from application perspective. This is difficult for more realistic applications.

⇒ metric learning, similarity learning ⇒ own research field