Probabilistic Active Learning in Datastreams

Daniel Kottke, Georg Krempl, Myra Spiliopoulou

Pseudocode and Efficiency

A naive implementation of the algorithm presented in the paper results in a computational complexity of $\mathcal{O}(w \log(w))$, as sorting is computationally expensive. A common data structure to efficiently store sorted representations are B-Trees, which need $\mathcal{O}(\log w)$ to add, find or remove values (see Alg. 1). The downside of B-Trees is that the rank (position in a sorted list) of elements is not directly accessible. Hence, we use the threshold θ directly. A new incoming usefulness value can either leave the threshold as it was or change the threshold to the next higher value (resp. the next lower one).

As already shown, the threshold index is determined by Eq. 1. Hence, the threshold index (and the threshold) might change solely by adding new instances. Therefore, we decrease the threshold during the insertion phase (ll. 23 - 32) if $\text{THRESIDX}(|Q|, b) - \text{THRESIDX}(|Q| - 1, b) = 1$ (see line 26). If the new value u_i is smaller than the threshold while the queue size is not reached, we have to increase the threshold by one (see l. 29).

$$\text{THRESIDX}(|Q|, b) = \lfloor |Q| \cdot (1 - b) \rfloor \tag{1}$$

After the queue has been filled, we have to distinguish other cases, because the threshold index is constant now but the deletion may cause threshold shifts. We distinguish two cases: First, if the value that is removed was the threshold or less and the new value is greater than the removed one, the threshold moves right (see line 18). The second case is exactly the opposite: if the removed value was the threshold or greater and the new value was less than the old one, the threshold must be moved left (see line 20). For every other case, the threshold stays constant.

The resulting algorithm runs in $\mathcal{O}(\log(w))$ time (per execution step). This is due to all operations being done in constant time, except for the B-Tree operations of insertion, deletion and finding min, max, successors and predecessors, which require $\mathcal{O}(\log(w))$).

Algorithm 1 Probabilistic Active Learning in Streams (with B-Trees)

```
b ∈ [0,1]; w, w_tol ∈ ℕ {Predefined budget, IQF window size, balancing window size}
C ← {} {Generative Classifier}
Q ← {} {Queue for IQF algorithm}
T ← BTree() {B-Tree to store sorted list}
θ ← null {Threshold value}
i ← 1, c_acq ← 0 {Instance counter, counter of acquired labels}
while Stream delivers new instance x_i do
  {determine spatial usefulness value}
  p̂ ← P_C(+|x_i);    n ← KFE_C(x_i)
  u_i ← pgain(p̂, n)
  {determine BIQF threshold}
  Q.push(u_i)
  T.insert(u_i)
  if |Q| > w then
    u_old ← Q.pop()
    T.remove(u_old)
    if u_old ≤ θ ∧ u_i > u_old then
      θ ← T.getNext(θ)
    else if u_old ≥ θ ∧ u_i < u_old then
      θ ← T.getPrev(θ)
    end if
  else
    if |Q| = 1 then
      θ ← u_i
    else
      if thresidx(|Q|, b) − thresidx(|Q| − 1, b) = 1 then
        θ ← T.getNext(θ)
      end if
      if u_i < θ then
        θ ← T.getPrev(θ)
      end if
    end if
  end if
  θ_bal ← θ − (T.getLast() − T.getFirst()) / w_tol · (b · (i − c_acq))
  if u_i ≥ θ_bal then
    C.retrain(x_i, getLabel(x_i))
    c_acq ← c_acq + 1
  end if
  i ← i + 1
end while
```