Online algorithms and machine learning

**Decision lists I:** Recall that a decision list is a series of if-then-else rules of the form “If \(x_i = b_i\) then [prediction]" where \(b_i \in \{0,1\}\). Can you think of a dataset that is not consistent with a decision list?

**Solution:** Sure. Just make an XOR on 2 variables.

**Decision lists II:** In class we saw a “batch” (offline) algorithm for learning decision lists over \(n\) Boolean features. This algorithm, given a labeled dataset \(S\), produced a consistent DL if one exists. Here, let’s consider the problem in the online setting. Examples (e.g., email messages) arrive one at a time. When each one arrives we must make a prediction (positive or negative). Then we are told if we were correct or not, and charged 1 mistake if we were incorrect.

Claim: there is an algorithm such that if the target function is a decision list (i.e., there really is some decision list \(f\) that is correct on all the examples) will make at most \(O(n^2)\) mistakes.

**Hint:** the algorithm will use “lazy” decision lists.

**Solution:** Here is what we can do. We will create a hypothesis (a predictor) which is like a decision list but may have several rules at the same level. The semantics is that if any rule at the top level fires, you pick one arbitrarily to use; else you go to the second level and see if any rule there fires and if so you pick one arbitrarily to use, and so on. We begin with all of the if-then rules at the top level. Any time we make a mistake, we move the rule that we used to make our prediction down by one level. Notice that the rule that is at the top level in the target function will never fire incorrectly and so will never be moved down below the top level. This also implies that the rule that is at level 2 in the target function will never be moved below level 2 (because once it reaches level 2, by induction it is below all rules that were above it in the target function \(f\) and so it will never fire incorrectly). More generally, the rule at level \(i\) in the target function will never be moved below level \(i\). Since each mistake moves one rule down by one level, and no rule is moved past level \(n\), this means that the total number of mistakes must be \(O(n^2)\).
**Paging.** In class, we discussed the marking algorithm for paging, but didn’t have time to talk about the proof of why it has competitive ratio $O(\log k)$. Present the proof for the case of $N = k + 1$.

**Algorithm “Marking”:**

- Assume the initial state is pages $1, \ldots, k$ in fast memory. Start with all pages unmarked.
- When a page is requested,
  - if it’s in fast memory already, mark it.
  - if it’s not, then throw out a random unmarked page. (If all pages in fast memory are marked, unmark everything first. For analysis purposes, call this the end of a “phase”). Then bring in the page and mark it.

We can think of this as a 1-bit randomized LRU, where marks represent “recently used” vs “not recently used”.

We will show the proof for the special case of $N = k + 1$. For general $N$, the proof follows similar lines but just is a bit more complicated.

**Proof.** (for $N = k + 1$). In a phase, you have $k + 1$ different pages requested so OPT has at least one page fault. For the algorithm, the page not in fast memory is a *random* unmarked page. Every time a page is requested: if it was already marked, then there is no page fault for sure. If it wasn’t marked, then the probability of a page fault is $1/i$ where $i$ is the number of unmarked pages. So, within a phase, the expected total cost is $1/k + 1/(k - 1) + \ldots + 1/2 + 1 = O(\log k)$. 

Name: ____________________________  2 Andrew ID: ____________________________