Measuring the “on-lineness” of data streams

Manfred K. Warmuth    Jiazhong Nie

University of California - Santa Cruz

Dec. 10, 2015 —- Nips workshop on Easy Data

Includes some earlier work with
Corrie Scalisi, Robert Gramacy, Scott Brandt and Ismail Ari
Goals

- Design on-line algorithms in domains that are outside of the reach of theory

- Design good comparators that exploit the on-lineness of the data
1. Disk spindown problem

- When to spin down the disk on your laptop?
- Best time-out time/user/usage dependent
Non-convex loss

If idles times expected to be
- short, then long timeout better
- long, then short timeout better
2. Caching

Want to build combined caching policy from 12 base policies (our experts):

- LRU, RAND, FIFO, LIFO, LFU, MFU, SIZE, GDS, GD*, GDSF, LFUDA
Characteristics Vary with Time

![Graph showing various data stream characteristics over time]
Best Policy Varies with time
Permuting trick for disk spindown data

on-line :-)

not on-line :-(
Permuting caching data

highly on-line data

some caching policies already on-line
Using a comparators to measure on-lineness of data

Properties

- Should exploit on-lineness of data
- Might be too expensive to compute in practice, but can serve as a goal to compare against
- Might rely on information not available to the on-line algorithm
Idea 1: Use dynamic programming to compute BestShift($K$) curve

- Partition of the timeline into $K$ segments
- BestFixed in each segment

\begin{itemize}
  \item \begin{tabular}{|c|c|c|}
    \hline
    2 & 4 & 7 \\
    \hline
  \end{tabular}
\end{itemize}

![Diagram showing average energy vs. total # of shifts with a point representing a partition]
Dynamic programming: $O(KN^2T)$

where $K$ # of partitions, $N$ # of discrete idle times, $T$ # of trials

[H]
BestShift curves

BestShift(K) on Cello-2 Data
50 experts exponentially spaced between 0 and 10
BestShift(K)
BestShift(K): Randomized
Optimal

on-line

BestShift(K) on Intel dataset
50 experts exponentially spaced between 0 and 10
BestShift(K)
BestShift(K): Randomized
Optimal

not on-line
Comparators for caching

- **BestFixed**: a posteriori best of 12 policies on entire request stream
- **BestRefetching**($R$):
  minimum number of misses with at most $R$ refetches
  in any sequence of switching policies
Refetches & Policy Switches

**Comparator:** All sequences of the form

\[
\begin{array}{cccccc}
\text{GDSF} & \text{LFUDA} & \text{SIZE} & \text{GDSF} & \text{LRU} \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
r(t_1) & r(t_2) & r(t_3) & r(t_4) & t
\end{array}
\]

We plot miss rate v.s. refetches:
BestRefetching($R$)

Dynamic programming: $O(RN^2T)$
Our theoretically sound algorithms become heuristics

- Use loss and share updates on non-convex losses

- Build a merged cache that does not correspond to the mixture
Spindown results

on-line :-)  not on-line :-(
Caching - we “Tracks” best policy

Miss-rates under FSUP with Master

Requests Over Time t

Miss-rates

lru
fifo
mruc
lifo
size
lfu
mfu
rand
gds
gdsf
lfuda
gd
roll
Measuring the “on-lineness” of data streams

$8.5\% = \text{LRU missrate}$
$2.0\% = \text{Obligatory missrate}$

Refetches as $\%$ of Total Requests

Missrates $\%$

BestRefetching(R)
Rank Ideal
Rank 60$\%$ Ideal
Rank 40$\%$ Ideal
BestFixed = SIZE
AllVC
SMoLRU Master and Comparator Missrates

59.8% = LRU missrate
15.3% = Obligatory missrate

- BestRefetching(R)
- Rank Ideal
- Rank 60% Ideal
- Rank 40% Ideal
- BestFixed = SIZE
- AllVC

Missrates % vs. Refetches as % of Total Requests
Idea 2: Split into even/odd requests

Requests:

Pair1
R1  R2  R3  R4  R5  R6  R7  R8  R9  R10
Pair2
Pair3
Pair4
Pair5

Training
Testing

- Best partition based on training set
- Performance based on test set
Miss Rate of Testing Requests

No overfitting to random data: testing miss rate goes up immediately

![Graph showing the miss rate against refetch rate for different data types (random permuted data train, random permuted data test, original data train, original data test). The graph indicates that the miss rate increases immediately when overfitting occurs.]
Don’t be afraid
to use your algorithms
as heuristics in domains
where the theory breaks down