Machine Learning for Data Management Systems

Bloom Filters; QD-Trees

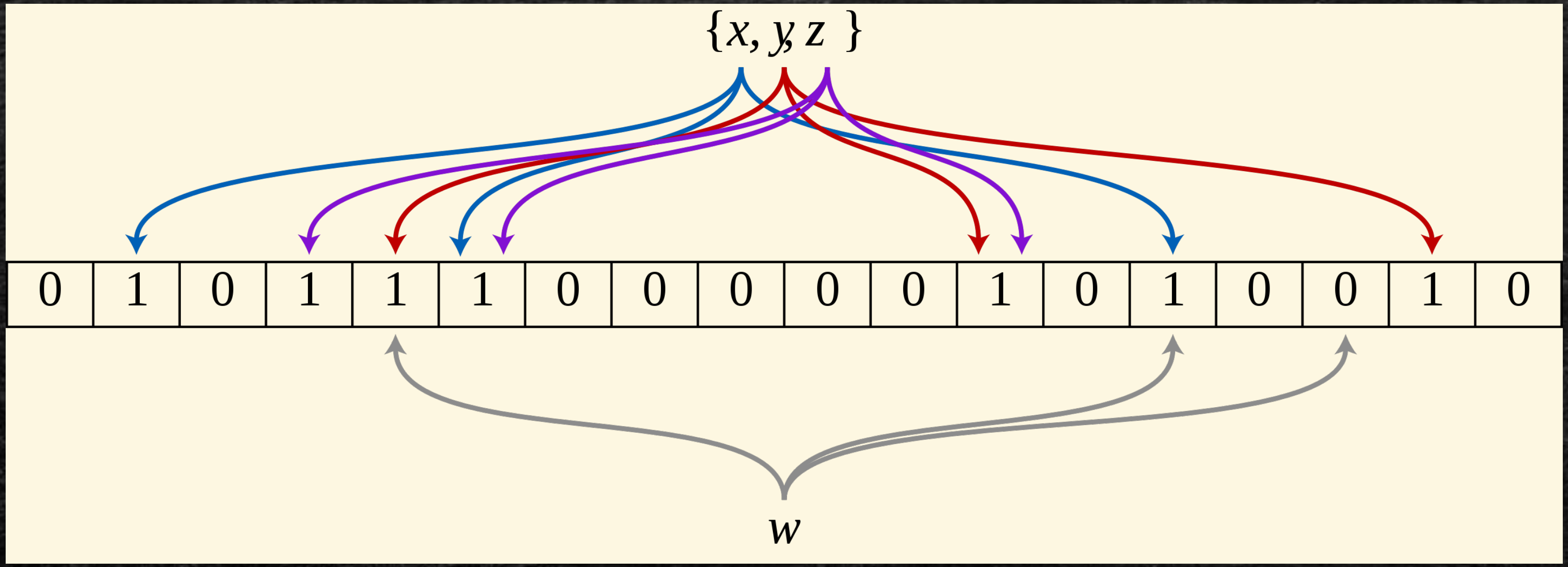
Amol Deshpande
February 9, 2023

Outline

- **Bloom Filters**
- Storage Layout Problem and Prior Work
- QD-Tree

Bloom Filters

Simplest one: Use a single hash function -- for each *key*, set *h(key)* = 1
Standard practice: Use 3 or more hash functions
No false negatives → if the *key* exists, we will return 1
Need to bound "false positives"



No updates allowed
Other data structures (e.g., counting bloom filter) that allow it

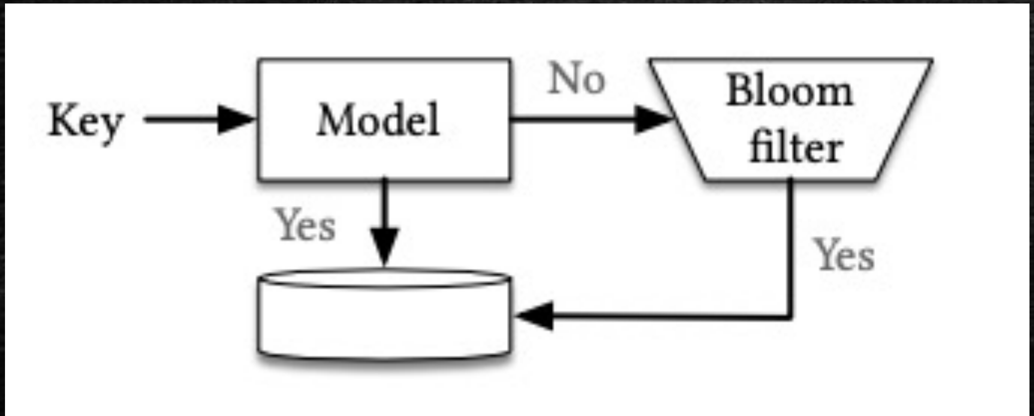
Bloom Filters

- Widely used in many contexts
  - e.g., Indexes use them to filter searches quickly (like log-structured merge trees)
  - Network problems (e.g., routing, etc)
  - Data streams
  - Can improve "joins" using them

- But, standard Bloom Filters too big
  - 2.23 GB for a FPR of 0.01, for 1 billion records

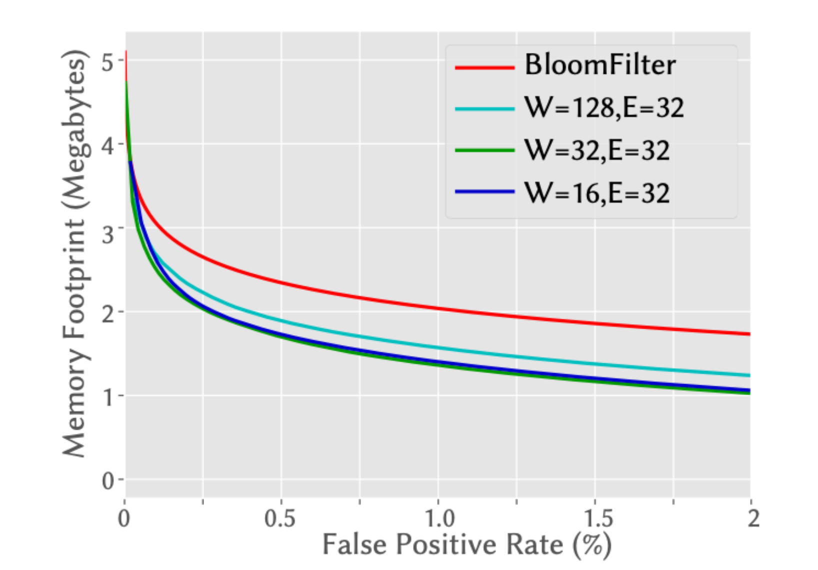
Learned Bloom Filters

- Train a classification model instead
  - Positive examples we have, but Negative examples are needed
  - Some positive examples may get classified as negative
- Approach:
  - Check the model first -- if positive, return TRUE
  - If not, consult an auxiliary Bloomfilter to confirm
    - Hopefully this BF is much smaller



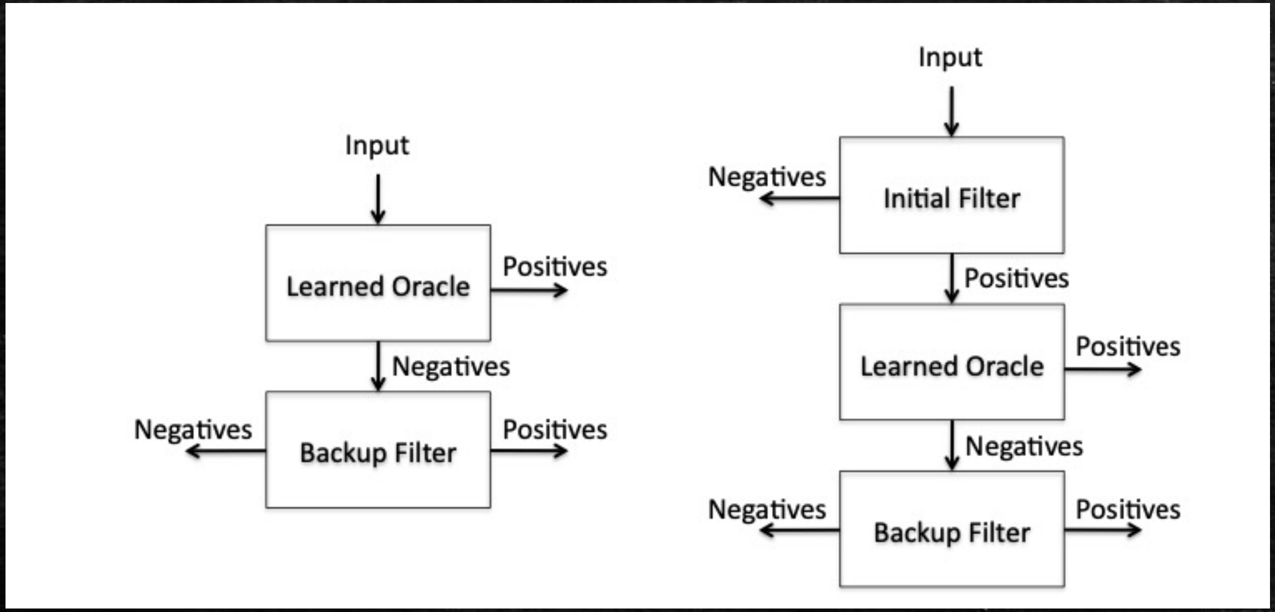
Learned Bloom Filters

- Results:
  - 1.7 M URLs; normal BF for 1% FPR = 2.04 MB
  - Model
    - 6-dimensional GRU with a 32-dimensional embedding for each character
    - 0.0259MB
  - Spillover Bloom Filter = 1.31



Sandwiched Bloom Filter

- Mitzenmacher; NeurIPS 2018
- Theoretical analysis of learned bloom filters
- Proposed adding another BF on top (model == learned oracle)
- Better theoretical guarantees



Follow-up Work

- Metalearning Bloom Filters
  - Uses a Recurrent Neural Network for one-shot (one-pass) learning
  - Maintains a small memory (analogous to Bloom Filter Bitmap) that's updated with new data items

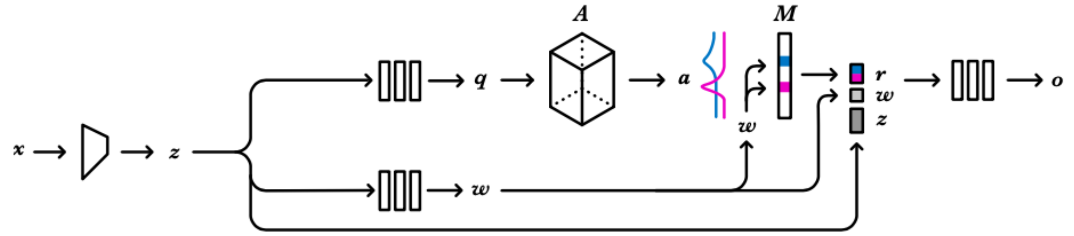


Figure 1. Overview of the Neural Bloom Filter architecture.

- Partitioned Bloom Filters; ICLR 2020
  - Maintain different auxiliary BFs for different score-ranges of the model
  - Figure out the "threshold" (to accept a key) automatically

Outline

- Bloom Filters
- **Storage Layout Problem and Prior Work**
- QD-Tree

Storage Data Layouts

- Most big datasets split by columns/rows
  - Usually by date (horizontal), and then columns (vertical)

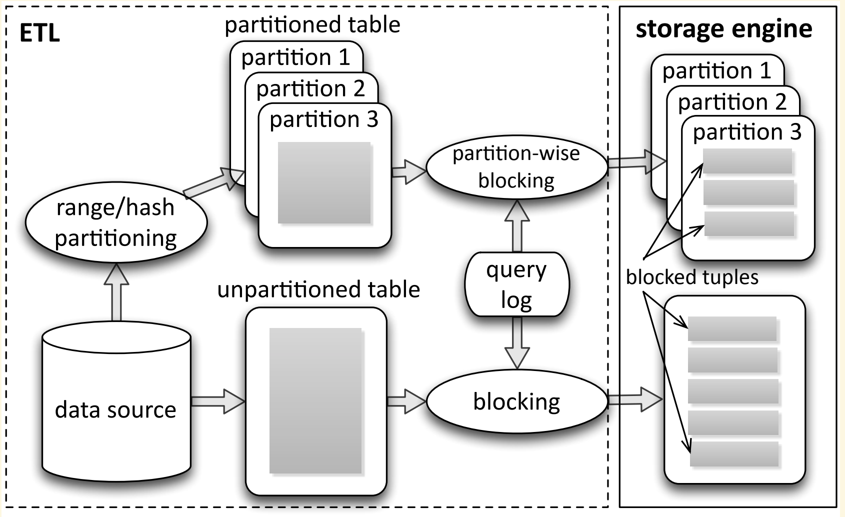


Figure 1: Blocking at Data Loading

Sun et al.; SIGMOD 14; Fine-grained data partitioning for aggressive skipping

Storage Data Layouts

- Most big datasets split by columns/rows
  - Usually by date (horizontal), and then columns (vertical)

	time	id	event	category	publisher	revenue
t_1	08:01:01	102	click	jeans	groupon	0.0
t_2	08:01:01	103	click	shirts	google	-0.5
t_3	08:01:01	104	click	shirts	groupon	0.0
t_4	08:01:02	105	buy	jeans	google	12.0
t_5	08:01:03	106	click	jeans	google	-0.5
t_6	08:01:04	107	buy	shoes	shoedeal	30.0

(a) tuples

	features	weight
F_1	*event='buy'*	50
F_2	*product='jeans'*	20
F_3	*publisher='google'* *revenue < 0*	10

(b) features

	vector (F_1, F_2, F_3)
t_1	(0,1,0)
t_2	(0,0,1)
t_3	(0,0,0)
t_4	(1,1,0)
t_5	(0,1,1)
t_6	(1,0,0)

(c) vectors

	blocking
P_1 (1,1,0)	t_1 (0,1,0) t_4 (1,1,0)
P_2 (0,1,1)	t_2 (0,0,1) t_5 (0,1,1)
P_3 (1,0,0)	t_3 (0,0,0) t_6 (1,0,0)

(d) blocks

Figure 2: Example of Blocking

Sun et al.; SIGMOD 14; Fine-grained data partitioning for aggressive skipping

Storage Data Layouts

- Most big datasets split by columns/rows
  - Usually by date (horizontal), and then columns (vertical)
- Sun et al., approach
  - Pick some “filters” that would be useful for skipping (i.e., commonly seen in queries)
  - Try to combine tuples into blocks that have similar feature vectors
    - Use bottom-up clustering for this
  - Try to maximize the zeros (more zeros ➔ more skipping)
  - A block tagged (1, 1, 0) ➔ has tuples with features 1 and 2, but no tuples with feature 3
    - Serves as a “semantic” description of the block
  - ”Complete-ness” not guaranteed
    - A tuple with features (1, 0, 0) could be in a few different blocks

Sun et al.; SIGMOD 14; Fine-grained data partitioning for aggressive skipping

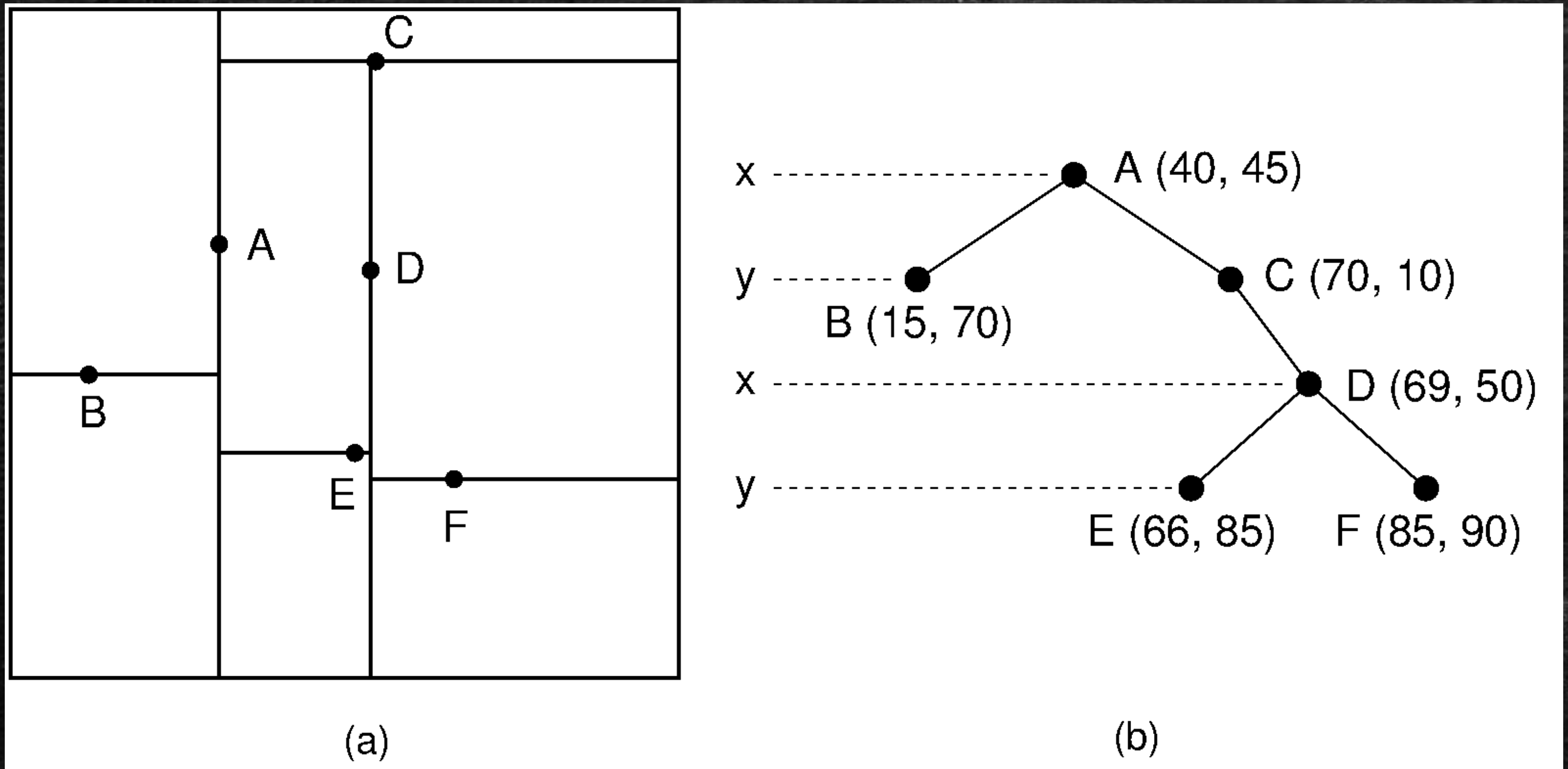
Outline

- Bloom Filters
- Storage Layout Problem and Prior Work
- **QD-Tree**

Main Ideas

- Aim for complete and semantic partitioning
- Roughly based on k-d-trees
  - Partition the multi-dimensional space recursively
  - In essence, build out a “decision tree” that partitions the tuples
  - Each leaf partition has a clear semantic description
- Question:
  - How to build this tree?

k-d-trees



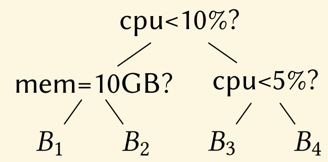


Figure 2: An example qd-tree with four leaf blocks.

Qd-Trees

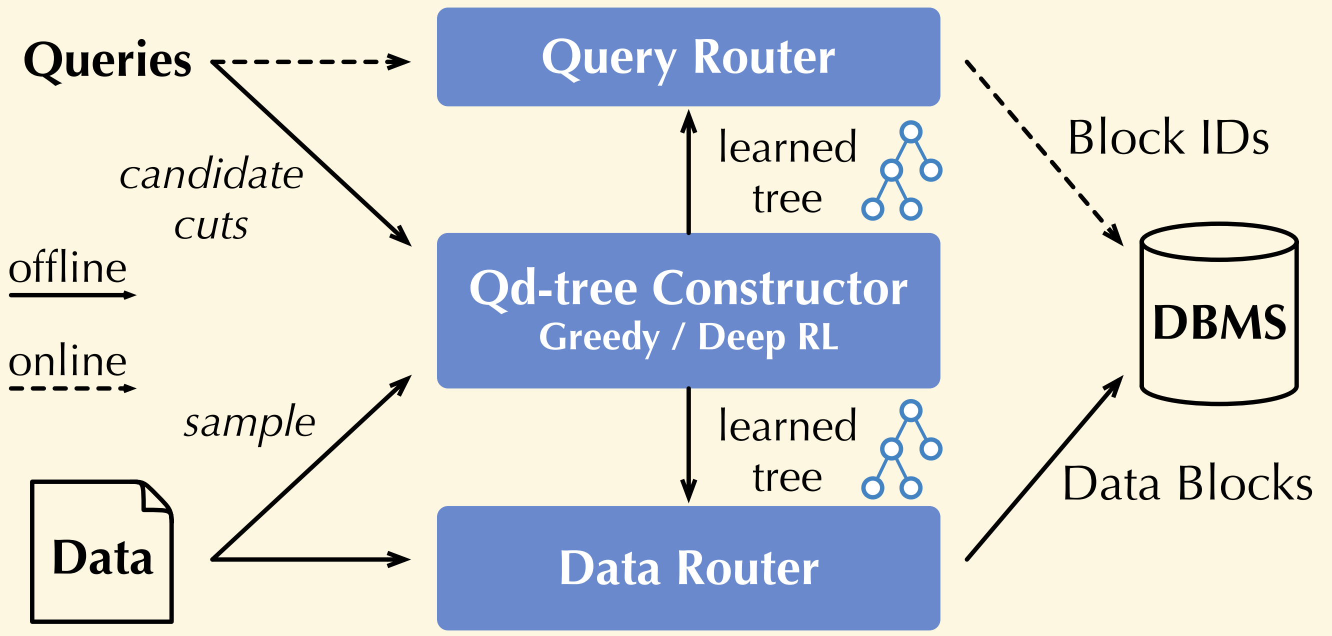


Figure 1: System Architecture.

Qd-Trees

Fields of node n	Definition
`n.range`	Hypercube describing the node's subspace. $2N$-dimensional array.
`n.categorical_mask`	Map: categorical column i's name → $\|Dom_i\|$-dim of bits. 0 means that value is not present.

Table 1: Semantic description of a qd-tree node.

Choosing Cuts

```
SELECT ... FROM R
WHERE (R.a < 10 OR R.b > 90) AND (R.c IN (0,4))
```

candidates: R.a < 10, R.b > 90, R.c in [0, 4]

Numerical columns

root.range: $[0, MAX_{cpu}), [0, MAX_{mem}),$

↓

left.range: $[0, 10\%), [0, MAX_{mem})$
right.range: $[10\%, MAX_{cpu}), [0, MAX_{mem})$

Categorical columns

root.categorical_mask: $(priority \rightarrow [1, 1, 1])$

↓

left.categorical_mask: $(priority \rightarrow [1, 1, 1])$
right.categorical_mask: $(priority \rightarrow [1, 0, 1])$

Advanced Cuts (similar to features)

- AC_0: `c_nationkey = s_nationkey`
- AC_1: `l_shipdate < l_commitdate`
- AC_2: `l_commitdate < l_receiptdate`

Greedy Algorithm

- NP-Hard Problem
- Adapts standard decision tree algorithm
- Start with root = all tuples
- Iterate:
  - Pick the leave with the best cut among all leaves (with size > 2b)
    - Best = highest benefit = most data skipped for queries
    - Can cache this from iteration to iteration
  - Add that cut and recurse
- Optimal algorithm based on Dynamic Programming
  - Too expensive

Greedy Algorithm

- Doesn’t look past the next cut
- Overall “goodness” of the tree can’t be estimated easily early on

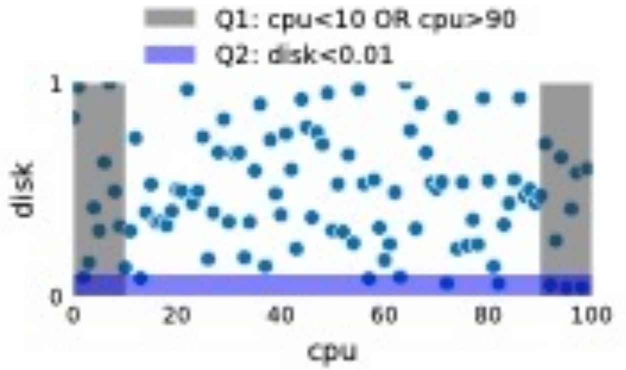


Figure 3: A dataset with disjunctive queries. Regions selected by Q1/Q2 are shown in grey/blue. The candidate cuts are: {cpu<10, cpu>90, disk<0.01}. The first two cuts cannot skip any query, so Greedy opts for the third cut, resulting in a scan ratio of 50.5%. Woodblock is not limited by the forms of queries; it produces a layout with a scan ratio of 10.4%, a 4.8× improvement. Discussion in Section 5.1.

Deep Reinforcement Learning

- Earlier work: decision trees for network packet classification (Liang et al.; SIGCOMM 2019)

- Define a Markov Decision Process
  - State space: set of all possible "subspaces" of the overall data space (i.e., any possible node in the decision tree)
  - Action: how to split the node
  - So we are going to learn how to split a given subspace

- Set up RL using two deep networks
  - policy network: $\pi_\theta : S \to A,$ takes a state (node) and outputs an action (stochastically)
  - value network: $V_\theta : S \to \mathbb{R},$ estimates reward for a given node
  - Shared weights
  - Use proximal policy optimization for updates

From Neurocuts Paper

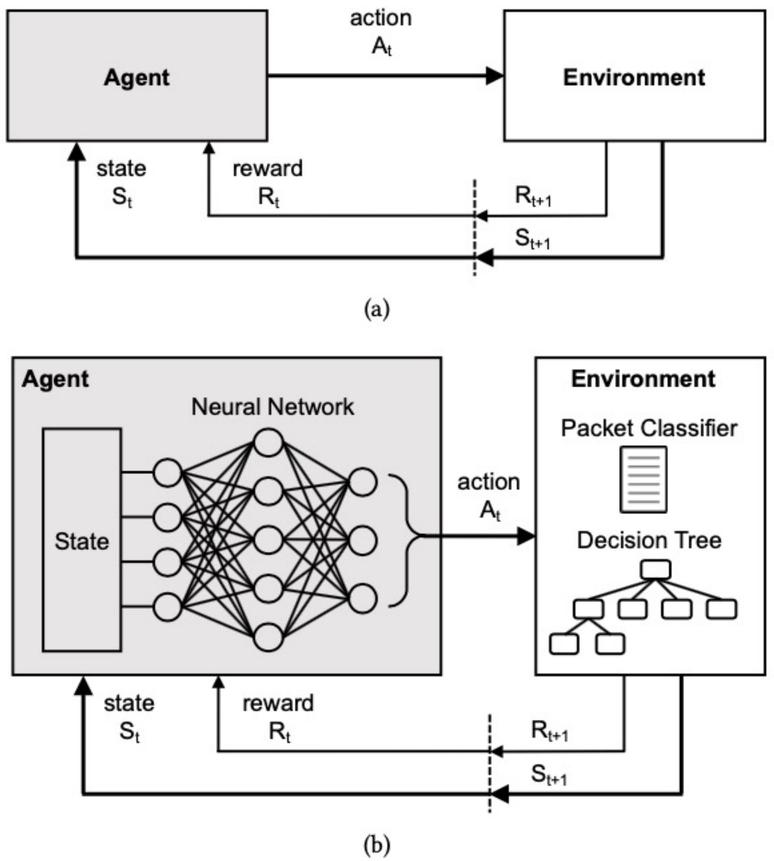


Figure 4: (a) Classic RL system. An agent takes an action, A_t, based on the current state of the environment, S_t, and applies it to the environment. This leads to a change in the environment state (S_{t+1}) and a reward (R_{t+1}). (b) NeuroCuts as an RL system.

Some Details

- Stopping condition based on size of the node
- All rewards computed on a sample (approx. 1%): could be still quite expensive
- Implementation:
  - Uses 2 fully connected layers, 512 units each, with ReLU activation
  - Different output layers for the two networks
  - State (node) defined by a concatenation of range predicates and categorial masks
- Extensions
  - Advanced multi-attribute predicate-based cuts
  - Allow for data overlap or duplication

Thoughts

- Solving an NP-Hard problem using RL
- Should we expect this to work?
- What if we were to sample a bunch of trees and try to average based on that?
  - Perhaps that's what RL is doing
- Failure scenarios?
  - Do we expect RL to converge to a bad local optimum

Literature Survey Assignment

- Individual; Due March 25, 2023
- Pick any one of these topics and explore state of the art
  - e.g., learned bloom filters, storage layouts, multi-dimensional indexes, etc.
  - See later papers as well (from the schedule)