Main Questions

How do we analyze and optimize learned Bloom filters?
How do we compare them to standard Bloom filters?

Standard Bloom Filters

- Given a set \( S = \{x_1, x_2, x_3, \ldots, x_n\} \) on a universe \( U \), want to answer membership queries of the form: Is \( y \in S \)?
- Data structure should be:
  - Fast (Faster than searching through \( S \))
  - Small (Smaller than explicit representation).
- To obtain speed and size improvements, allow some probability of error.

False positives: \( y \in S \) but we report \( y \notin S \)
False negatives: \( y \notin S \) but we report \( y \in S \)

Learned Bloom Filters

- Given a set \( S = \{x_1, x_2, x_3, \ldots, x_n\} \) on a universe \( U \), want to answer membership queries of the form: Is \( y \in S \)?
- Data structure should be:
  - Fast (Faster than searching through \( S \))
  - Small (Smaller than explicit representation).
- To obtain speed and size improvements, allow some probability of error.

Learned Bloom Filter Analysis

- Items that the oracle says are very likely positives (above threshold) are treated as positives. This yields some false positives.
- Items that the oracle says are negatives might include false negatives. A backup (standard) Bloom filter catches false negatives from the oracle and returns them as positives. This might create additional false positives.

Sandwiched Learned Bloom Filter Analysis

- Take derivatives to optimize:
  \[
  b_i = F_e \log_{1+}\left(\frac{F_e}{F_e}\right)
  \]

- Optimal configuration has small, fixed-sized backup filter
- All remaining bits go to the initial filter
- Better to get rid of false positives early
- Can improve false positives by an order of magnitude

Many Open Problems

- Further optimizations?
- Best implementation? Computational tradeoffs?
- What applications have sets amenable to learning?
- Other algorithms/data structures that can benefit from learning?