My Goals

• Find ways to include machine learning in “traditional algorithms”
  • Machine learning algorithm can just be a black box

• Ideally with provable statements/guarantees.
  • Guarantees can be based on assumptions about machine learning algorithm performance.
Analyzing Learned Bloom Filters

Michael Mitzenmacher
Bloom Filters: Approximate Membership Queries

• 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:
  $$\text{Is } y \in S.$$  

• 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: $y \notin S$ but we report $y \in S$
  • False negatives: $y \in S$ but we report $y \notin S$
Bloom Filters

Start with an $m$ bit array, filled with 0s.

Hash each item $x_j$ in $S$ $k$ times. If $H_i(x_j) = a$, set $B[a] = 1$.

To check if $y$ is in $S$, check $B$ at $H_i(y)$. All $k$ values must be 1.

Possible to have a false positive; all $k$ values are 1, but $y$ is not in $S$.

$n$ items $\quad m = cn$ bits $\quad k$ hash functions
False Positive Probability

- $\Pr($specific bit of filter is 0$)$ is
  
  $$p' \equiv (1 - 1/m)^{kn} \approx e^{-kn/m} \equiv p$$

- If $\rho$ is fraction of 0 bits in the filter then false positive probability is
  
  $$ (1 - \rho)^k \approx (1 - p')^k \approx (1 - p)^k = (1 - e^{-k/c})^k$$

- Approximations valid as $\rho$ is concentrated around $E[\rho]$.
  - Martingale argument suffices.

- Find optimal at $k = (\ln 2)m/n$ by calculus.
  - So optimal fpp is about $(0.6185)^{m/n}$

\[ n \text{ items} \quad m = cn \text{ bits} \quad k \text{ hash functions} \]
Example

$\frac{m}{n} = 8$

Opt $k = 8 \ln 2 = 5.45...$

Hash functions

False positive rate

$0.1$

$0.09$

$0.08$

$0.07$

$0.06$

$0.05$

$0.04$

$0.03$

$0.02$

$0.01$

$0$

$n$ items

$m = cn$ bits

$k$ hash functions
Learned Bloom Filters

- Google Brain suggests can do better than standard Bloom filters
  - In a data dependent way
  - Assuming you can “learn” the set from the Bloom filter
- Use machine learning to develop a small-size oracle that provides probability an element is in a set
  - Oracle should (hopefully) give few false positives
  - And you need a backup to catch any false negatives.
Extreme Example

• Suppose that your set is a collection of closed intervals (universe is 32 bit integers).

• This has a very small representation.
  • 2 integers per interval.
  • Maybe it can be learned.

• You could just have a “Bloom filter” that checks if the input is in one of these intervals.
  • Could be much smaller and more accurate than a regular Bloom filter.
  • Depends on specific structure of the data.

• Basis of learning: most data has some structure so there exists a small, mostly accurate representation.
Learned Bloom Filter Setup (Google Brain)

• Bloom filters as binary classification problem
• Use set of elements as a collection of positive examples.
• Use set of non-elements as a collection of negative examples.
  • Can be chosen non-elements, or random non-elements.
• Derive neural network with sigmoid activation to produce a “probability estimate” than an input is in the set, minimizing log loss.
• Choose a threshold.
  • Elements evaluated above the threshold are assumed positive; below the threshold are assumed negative.
• May be false positives, and false negatives.
Learned Bloom Filter Setup (Google Brain)

Items that the oracle says are very likely positives are treated as positives. Might be some false positives.

Items that are negatives might include possible false negatives! So we have a backup Bloom filter to catch oracle negatives in the set.

This might create additional false positives.
False Positives vs. “False Positives”

• Both Bloom filters and learned Bloom filters have false positives
  • But terms means different things.

• For a Bloom filter, the false positive probability is the probability *any* non-set item yields a false positive
  • And corresponds to the rate of false positives for any collection of distinct non-set items.
False Positive Probability

• Pr(specific bit of filter is 0) is
  \[ p' = (1 - 1/m)^{kn} \approx e^{-kn/m} \equiv p \]
• If \( \rho \) is fraction of 0 bits in the filter then false positive probability is
  \[ (1 - \rho)^k \approx (1 - p')^k \approx (1 - p)^k = (1 - e^{-k/c})^k \]
• Approximations valid as \( \rho \) is concentrated around \( \mathbb{E}[^{\rho}] \).
  • Martingale argument suffices.
• Find optimal at \( k = (\ln 2)m/n \) by calculus.
  • So optimal fpp is about \( (0.6185)^{m/n} \)
False Positives vs. “False Positives”

• Both Bloom filters and learned Bloom filters have false positives
  • But terms means different things.

• For a learned Bloom filter, the false positive probability (from the learned oracle) is an empirical estimate, assuming your future data looks like the training data used to create the filter.
  • Big assumption, not suitable for many applications.
  • Proposed solution – if the false positives become too high, retrain. (Not a great solution?)
Example

• Universe is $[1,1000000]$. 
• Bloom filter holds 500 random keys from $[1000,2000]$. 
• So learned Bloom filter might accept all keys from $[1000,2000]$. 
  • Can’t “learn” random behavior in this range.
• False positive probability depends on query range/distribution. 
  • If queries are uniform over whole space, very low. 
  • If queries are uniform over $[1,100000]$, now false positives about ten times larger.
Learned Oracle Backup Filter

Input

Learned Oracle

Positives

Negatives

Backup Filter

Positives

Negatives

Let $m$ be size of set being stored.

Let $x$ be size of the learned oracle.
Let $FP$ be false positive rate of learned oracle.
Let $FN$ be fraction of false negatives of learned oracle.

Let $bm$ be size of the backup filter.

Let false positive rate of a Bloom filter with $z$ bits per item be $a^z$. 
A Basic Comparison

• So we want to compare a standard Bloom filter with $x+bm$ bits for $m$ elements to a learned Bloom filter as in the last slide.
• False positive probability of a standard Bloom filter = $a^{(b+x/m)}$.
• False probability of the learned Bloom filter is $FP+(1 - FP)a^{(b/FN)}$.
• Learned Bloom filter is better when

$$ \frac{x}{m} < \log_a(FP+(1-FP)a^{(b/FN)}) - b $$
A Better Construction?

• Bloom filter at the end catches false negatives.
  • So you have to have that.
• Maybe a Bloom filter at the beginning could throw out false positives from the learned Bloom filter.
  • Sandwiched learned Bloom filter.
• Is having a Bloom filter at the beginning worth it?
Comparison

Learned Oracle

backup filter

Input

Positives

Negatives

Initial Filter

Learned Oracle

Input

Positives

Negatives

Backup Filter
Sandwiched Learned Bloom Filter Setup

An initial Bloom filter for the set can remove false positives before the learned oracle passes them on.

Items that the oracle says are very likely positives are treated as positives. Might be some false positives.

Items that are negatives might include possible false negatives! So we have a backup Bloom filter to catch oracle negatives in the set. This might create additional false positives.
Sandwiched analysis

• Sandwiched learned Bloom filter with \( bm \) Bloom filter bits for \( m \) elements. Use \( b_1m \) for initial filter, \( b_2m \) for backup filter.

• False positive probability of sandwiched Bloom filter = \( a^{b_1} \) \((FP+(1-FP)a^{b_2/FN})\).
  • Passes through first Bloom filter with prob. \( a^{b_1} \)
  • Then false positive from learned filter or backup filter possible.
Sandwiched analysis

• Sandwiched learned Bloom filter with $bm$ Bloom filter bits for $m$ elements. Use $b_1m$ for initial filter, $b_2m$ for backup filter.

• False positive probability of sandwiched Bloom filter = $a^{b_1} (FP+(1-FP)a^{b_2/FN})$.
  • Passes through first Bloom filter with prob. $a^{b_1}$
  • Then false positive from learned filter or backup filter possible.
Sandwiched Analysis

• False positive probability of sandwiched Bloom filter
  \[ = a^{b_1} \left( FP + (1 - FP) a^{b_2/FN} \right). \]
  
• Equivalent expression:  \[ a^{b_1}FP + (1 - FP) a^{b/FN} a^{b_1(1 - 1/FN)}. \]

• Take derivative with respect to \( b_1 \), other values are constants.
Sandwiched Analysis

As $\alpha, F_p, F_n$ and $b$ are all constants for the purpose of this analysis, we may optimize for $b_1$ in the equivalent expression

$$F_p \alpha^{b_1} + (1 - F_p) \alpha^{b/F_n} \alpha^{b_1(1-1/F_n)}.$$

The derivative with respect to $b_1$ is

$$F_p (\ln \alpha) \alpha^{b_1} + (1 - F_p) \left(1 - \frac{1}{F_n}\right) \alpha^{b/F_n} (\ln \alpha) \alpha^{b_1(1-1/F_n)}.$$

This equals 0 when

$$\frac{F_p}{(1 - F_p) \left(\frac{1}{F_n} - 1\right)} = \alpha^{(b-b_1)/F_n} = \alpha^{b_2/F_n}. \quad (1)$$

This further yields that the false positive rate is minimized when $b_2 = b_2^*$, where

$$b_2^* = F_n \log_\alpha \frac{F_p}{(1 - F_p) \left(\frac{1}{F_n} - 1\right)}.$$
Analysis Outcome

• Backup Bloom filter has a *fixed size* (given other parameters).

• Intuition: better to head off false positives up front; just use the backup Bloom filter to get back false negatives.

• Same bits get improvements of factor of 2-10 in false positives for some standard parameters. (See paper.)
Other Structures

• Bloom filter construction useful, but...
  • Anything else?

• Bloomier filters?
  • Bloom filters are for set membership, in or out.
  • Bloomier filter returns a function value.
  • e.g. stores pairs \((x,y)\), where \(x\) in \(S\) and \(y\) in \(R\) for a set \(S\) of keys and a (limited) range \(R\) of values
    • Use null value for elements not in \(S\)
  • May yield false positives – e.g. non-null values – for keys not in \(S\), but should never return a wrong value for elements of \(S\).
  • Can we make a learned Bloomier filter?
Learned Oracle returns a value for an input; should be null for elements outside of $S$.

Use a Bloom filter to catch any false negatives from the Bloomier filter. Any non-hit on the Bloom filter ends with the value from the learned oracle.

Backup Bloomier filter gives correct values for false negatives. May also yield additional false positives.
Bloomier Filter Analysis

• Read the paper.
Scheduling with Predictions and The Price of Misprediction

Michael Mitzenmacher
Scheduling

• Suppose we have only two types of jobs, short and long.
• There are n jobs, you choose execution order.
• Goals is to minimize average waiting time.
• Assume no preemption.
• Natural schedule is *shortest job first*.
  • Easily proven optimal.
Scheduling with Prediction

• Now suppose only have predictions of whether jobs or short or long.
• Natural strategy: *shortest predicted job first*.
• Performance depends on how often jobs are misclassified.
Price of Misprediction

• Natural comparison points:
  • No information = random order
    • If an adversary presented jobs you would randomize rather than take given order
  • Full information = shortest job first

• Let us call the “price of misprediction” the ratio of performance (expected waiting time) between predicted information and full information.
  • Another variation of competitive ratio.
  • Similar to regret.
  • Can you appropriate metric in context.
Exercise

• Derive price of misprediction for short/long job scenario.
  • Short jobs have time a, long jobs have time b.
  • Assume predictions are independent, but probability of short being classified as long may be different than long classified as short.
  • Not a hard calculation.
  • In the paper.
Harder Exercise

• Suppose job times are independent, exponentially distributed, with mean 1.
• Suppose prediction for a job with actual time x is exponentially distributed with mean x.
• What is the price of misprediction?
Harder Exercise

• Suppose job times are independent, exponentially distributed, with mean 1.
• Suppose prediction for a job with actual time $x$ is exponentially distributed with mean $x$.
• What is the price of misprediction?
• This is much harder. (Some fun integrals.)
Harder Exercise

• Suppose job times are independent, exponentially distributed, with mean 1.
• Suppose prediction for a job with actual time $x$ is exponentially distributed with mean $x$.
• What is the price of misprediction?
• This is much harder. (Some fun integrals.)
• Key insight: linearity of expectations says you only need to consider a pair of jobs.
Harder Exercise

• Suppose job times are independent, exponentially distributed, with mean 1.
• Suppose prediction for a job with actual time x is exponentially distributed with mean x.
• What is the price of misprediction?
• This is much harder. (Some fun integrals.)
• Key insight: linearity of expectations says you only need to consider a pair of jobs.
• Answer turns out to be 4/3.
General Model

• How should we model this generally?

• Joint probability density function
  • Let $g(x,y)$ be the probability density for: a job requires time $x$, and is predicted to require time $y$.
  • Can think of it as the real job time induces a distribution on the output of the learning model, or the predicted job time induces a distribution on the real job time.
  • Also convenient to let
    
    $$f_s(x) = \int_{y=0}^{\infty} g(x,y)dy$$  
    Service density

    $$f_p(x) = \int_{x=0}^{\infty} g(x,y)dx$$  
    Predicted service density
General Formula

• Price of misprediction

\[
\frac{\int_{y=0}^{\infty} f_p(y) \left( \int_{x=0}^{\infty} \int_{z=0}^{y} g(x, z) dz \ dx \right) dy}{\int_{x=0}^{\infty} f_s(x) \left( \int_{z=0}^{\infty} zf_s(z) dz \right) dx}
\]
Queueing Setting

Poisson arrivals at rate $\lambda$, $\lambda < 1$

Service times have mean 1
Example: M/M/1 queue
Service times exponential
First In First Out (FIFO)
Queuing Setting, Known Job Times

• Service times governed by a general distribution
• If you know the job times, optimal strategies for minimizing average waiting time are:
  • Shortest job first (SJF) [when non-preemptive]
  • Shortest remaining processing time (SRPT) [when preemptive]
• Known formulae for average time in system in equilibrium for these strategies.
Queuing Setting, Predicted Job Times

• Service times/predicted service times governed by a general distribution with probability density $g(x,y)$

• If all you know are the predicted job times, natural strategies for minimizing average waiting time are:
  • Predicted shortest job first (PSJF) [when non-preemptive]
  • Predicted shortest remaining processing time (PSRPT) [when preemptive]

• Can we derive formula for average time in system?

• How good are they?
Review: Standard Queues

- Poisson arrivals at rate $\lambda$
- Service time distribution $S$ (usually mean 1)
- Let $\rho = \lambda E[S] = \text{load at the queue}$
- Then waiting time $W$ given by $W = \frac{\rho E[S^2]}{2E[S](1-\rho)}$
SJF vs SPJF

For SJF, let $\rho_x = \lambda \int_{t=0}^{x} tf_s(t) \, dt$ : server load of jobs with service time up to $x$. Let $W_x$ be the expected waiting time for job of size $x$.

$$W_x = \frac{\rho E[S^2]}{2E[S](1 - \rho_x)^2}$$

$W = \int_{x=0}^{\infty} f_s(x)W_x \, dx$

For SPJF, let $\rho_y' = \lambda \int_{t=0}^{y} \int_{x=0}^{\infty} x \, g(x, t) \, dx \, dt$ : server load of jobs with predicted service time up to $x$. Let $W_y'$ be the expected waiting time for job of predicted size $y$.

$$W_y' = \frac{\rho E[S^2]}{2E[S](1 - \rho_y')^2}$$

$W = \int_{y=0}^{\infty} f_p(y)W_y' \, dy$
Price of Misprediction

\[
\frac{\int_{y=0}^{\infty} \frac{f_p(y)}{(1 - \rho'_y)^2} \, dy}{\int_{x=0}^{\infty} \frac{f_s(x)}{(1 - \rho_x)^2} \, dy}
\]
Experimental Results 1

• Goal: Show equations are accurate
• Poisson arrivals at rate $\lambda$
• Exponential service times, mean 1
• When service time is $x$, predicted service time is exponential with mean $x$.
• Take average of 1000 trials over 1000000 time steps (first 100000 discarded).
### Experimental Results 1

<table>
<thead>
<tr>
<th>λ</th>
<th>SJF Eqns</th>
<th>SJF Sims</th>
<th>SPJF Eqns</th>
<th>SPJF Sims</th>
<th>FIFO Eqns</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>1.7127</td>
<td>1.7128</td>
<td>1.7948</td>
<td>1.7949</td>
<td>2.00</td>
</tr>
<tr>
<td>0.6</td>
<td>1.9625</td>
<td>1.9625</td>
<td>2.1086</td>
<td>2.1087</td>
<td>2.50</td>
</tr>
<tr>
<td>0.7</td>
<td>2.3122</td>
<td>2.3121</td>
<td>2.5726</td>
<td>2.5730</td>
<td>3.33</td>
</tr>
<tr>
<td>0.8</td>
<td>2.8822</td>
<td>2.8828</td>
<td>3.3758</td>
<td>3.3760</td>
<td>5.00</td>
</tr>
<tr>
<td>0.9</td>
<td>4.1969</td>
<td>4.1987</td>
<td>5.3610</td>
<td>5.3609</td>
<td>10.00</td>
</tr>
<tr>
<td>0.95</td>
<td>6.2640</td>
<td>6.2701</td>
<td>8.6537</td>
<td>8.6541</td>
<td>20.00</td>
</tr>
<tr>
<td>0.98</td>
<td>11.2849</td>
<td>11.2734</td>
<td>16.9502</td>
<td>16.9782</td>
<td>50.00</td>
</tr>
<tr>
<td>0.99</td>
<td>18.4507</td>
<td>18.4237</td>
<td>29.0536</td>
<td>29.1162</td>
<td>100.00</td>
</tr>
</tbody>
</table>

*Table 1. Results from simulations and equations for Shortest Job First (SJF) and Shortest Predicted Job First (SPJF).*
**Experimental Results 1**

<table>
<thead>
<tr>
<th>( \lambda )</th>
<th>SRPT Eqns</th>
<th>SRPT Sim</th>
<th>SPRPT Eqns</th>
<th>SPRPT Sim</th>
<th>FIFO Eqns</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>1.4254</td>
<td>1.4251</td>
<td>1.6531</td>
<td>1.6588</td>
<td>2.00</td>
</tr>
<tr>
<td>0.6</td>
<td>1.6041</td>
<td>1.6039</td>
<td>1.9305</td>
<td>1.9397</td>
<td>2.50</td>
</tr>
<tr>
<td>0.7</td>
<td>1.8746</td>
<td>1.8757</td>
<td>2.3539</td>
<td>2.3684</td>
<td>3.33</td>
</tr>
<tr>
<td>0.8</td>
<td>2.3528</td>
<td>2.3519</td>
<td>3.1168</td>
<td>3.1376</td>
<td>5.00</td>
</tr>
<tr>
<td>0.9</td>
<td>3.5521</td>
<td>3.5486</td>
<td>5.04808</td>
<td>5.0973</td>
<td>10.00</td>
</tr>
<tr>
<td>0.95</td>
<td>5.5410</td>
<td>5.5466</td>
<td>8.3221</td>
<td>8.4075</td>
<td>20.00</td>
</tr>
<tr>
<td>0.98</td>
<td>10.4947</td>
<td>10.5003</td>
<td>16.6239</td>
<td>16.7852</td>
<td>50.00</td>
</tr>
<tr>
<td>0.99</td>
<td>17.6269</td>
<td>17.6130</td>
<td>28.7302</td>
<td>28.7847</td>
<td>100.00</td>
</tr>
</tbody>
</table>

*Table 2. Results from simulations and equations for Shortest Remaining Processing Time (SRPT) and Shortest Predicted Remaining Processing Time (SPRPT).*
Experimental Results 2

• Goal: Show predicted results work well, even with poor predictions.
• Poisson arrivals at rate $\lambda$
• Exponential service times, mean 1
• Also Weibull-$1/2$ service time distribution, mean 1.
  • A more “heavy-tailed” distribution.
• When service time is $x$, predicted service time is uniformly distributed on $[(1-\alpha)x,(1+\alpha)x]$.  
  • Right mean, but can be way off.
Experimental Results 2
Takeaways

• Equations are accurate. (Less than 1% error, except at 0.99 arrival rate.)
• SPRPT appears better than SPJF.
  • Even better on heavy-tailed distributions.
• SPRPT and SPJF seem to do very close to SRPT and SJF
  • Depends on quality of prediction
  • Intuition: just have to get the order mostly right to do well.
    • This is a really good problem for prediction.
• Even “bad” predictions can be much better than FIFO.
Future Work

• Fairness?
• More general models than joint distribution $g(x,y)$?
• Other queueing systems? [Note: I have some work in progress already...]
• Other scheduling-type problems?