Probabilistic Inference and Privacy:

What to do when you've asked too much.

Frank McSherry and Ollie Williams
Microsoft Research, SVC

Outline

1. Differential Privacy and Probabilistic Inference

2. Case Study 1: Logistic Regression

3. Case Study 2: Synthetic Data Sets (w/Kunal Talwar)

4. Privacy Integrated Queries and Probabilistic Inference

Differential Privacy

w/Cynthia Dwork, from work with Kobbi Nissim and Adam Smith.

Ensures: Any event S "equally likely" with/without your data.
Does not prevent disclosure. Ensures disclosure is not our fault.

Lots of really neat properties:

1. No computational / informational assumptions of attackers.
2. Agnostic to data type D. Could be PII, binary data, anything.
3. Formal. Not only gives guarantees, but a basis for extension.

   ...

n. Allows mechanisms to expose the conditional probabilities:

$$\text{Pr[outcome} \mid \text{data]}$$

This last one is the only one we are going to care about here.

Probabilistic Inference

Noisy observations z (aggregates) of hidden variables x (data):

$$\text{Pr[data} \mid \text{outcome]} \;\propto\; \text{Pr[outcome} \mid \text{data]} \times \text{Pr[data]} .$$

Noisy observations result in a posterior over possible data sets.

On their own, observations often maximum likelihood estimates.
But, better things to do than use them as the "correct answer":

1. Integrate multiple observations. (accuracy++)
2. Express/use confidence in answers. (helpful in (1.))
3. Use prior knowledge about data. (non-negativity)
4. Posteriors over unasked questions. (hooray!)

Ex 1: Logistic Regression

Optimize a vector w against $\{x_j\}$ under the objective function:

$$\sum_j \text{log}(\text{Logistic}(< w, x_j >)) \; .$$

Consider a very simple iterative algorithm for Logistic Regression:

```
// Finds the gradient of Sum_j Log(Logistic(<vector, x_j>)) at vector.
public void LogisticStep(PINQueryable<double[]> input, double[] vector)
{
  // the gradient of the log-likelihood
  var gradient = new double[vector.Length];
  foreach (var i in Enumerable.Range(0, vector.Length))
    gradient[i] = input.Sum(epsilon, x => Logistic(vector, x) * x[i]);

  // incorporate gradient into the vector
  foreach (var i in Enumerable.Range(0, vector.Length))
    vector[i] = vector[i] + lambda * gradient[i];
}
```

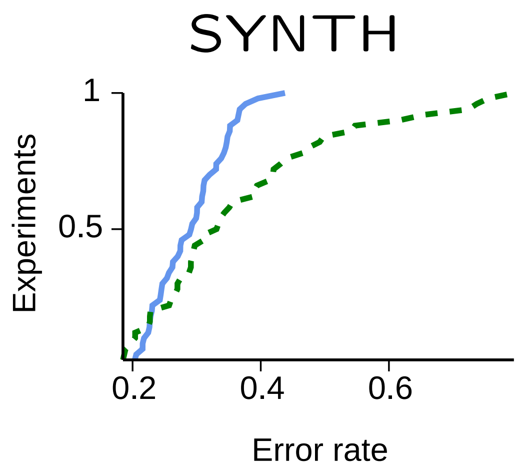
Performance can be OK. Pretty bad when epsilon is very small.

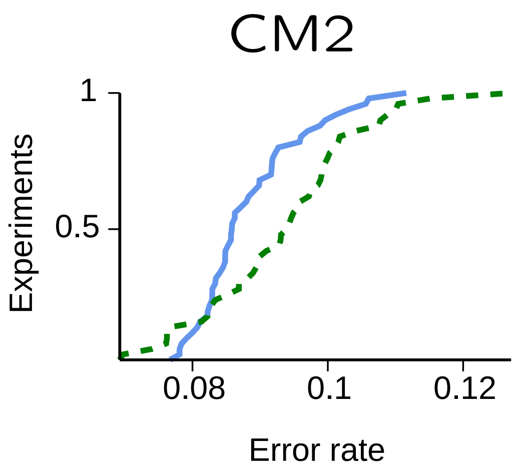
Logistic Regression (eps 0.1)

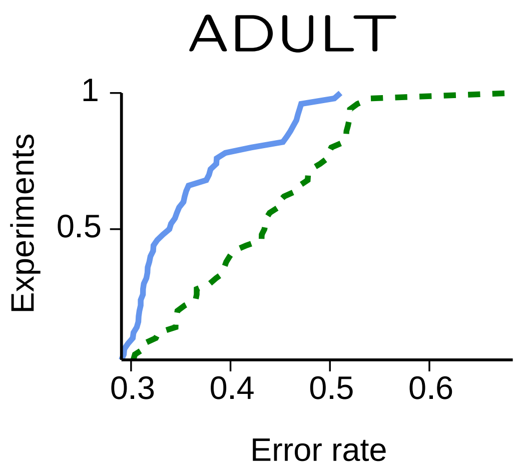
If we layer probabilistic inference on top of the observations,

	SYNTH	CM2	ADULT
Heuristic	37.40 ± 15.75	9.32 ± 1.18	43.15 ± 7.85
Inference	29.14 ± 5.54	8.84 ± 0.79	36.07 ± 6.32
Benchmark	16.40	5.40	26.09

Cumulative density functions for classification error rates:





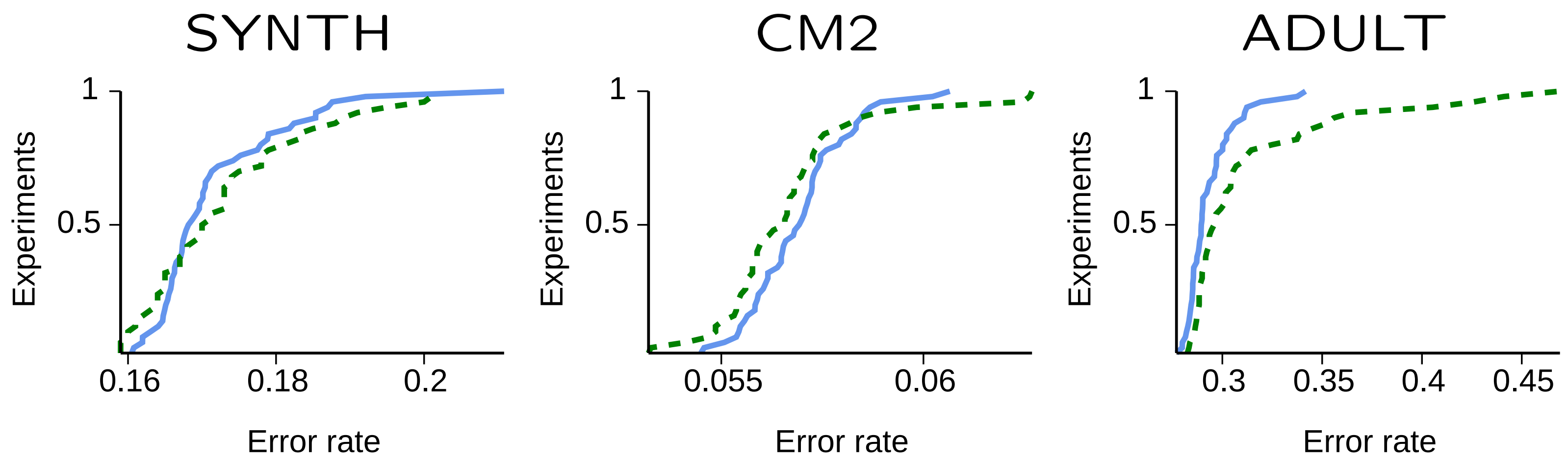


Logistic Regression (eps 1.0)

If we layer probabilistic inference on top of the observations,

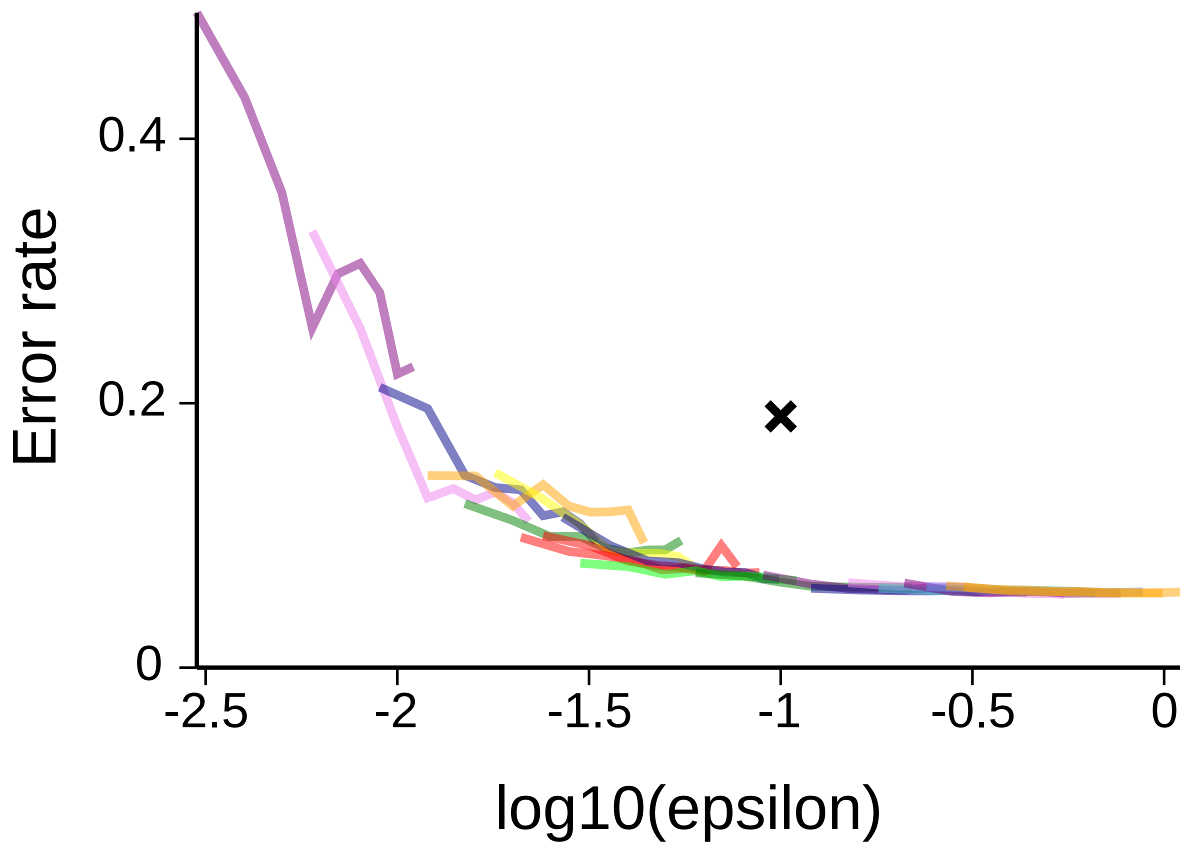
	SYNTH	CM2	ADULT
Heuristic	17.31 ± 1.12	5.67 ± 0.19	31.30 ± 4.16
Inference	17.16 ± 0.94	5.69 ± 0.13	29.36 ± 1.31
Benchmark	16.40	5.40	26.09

Cumulative density functions for classification error rates:



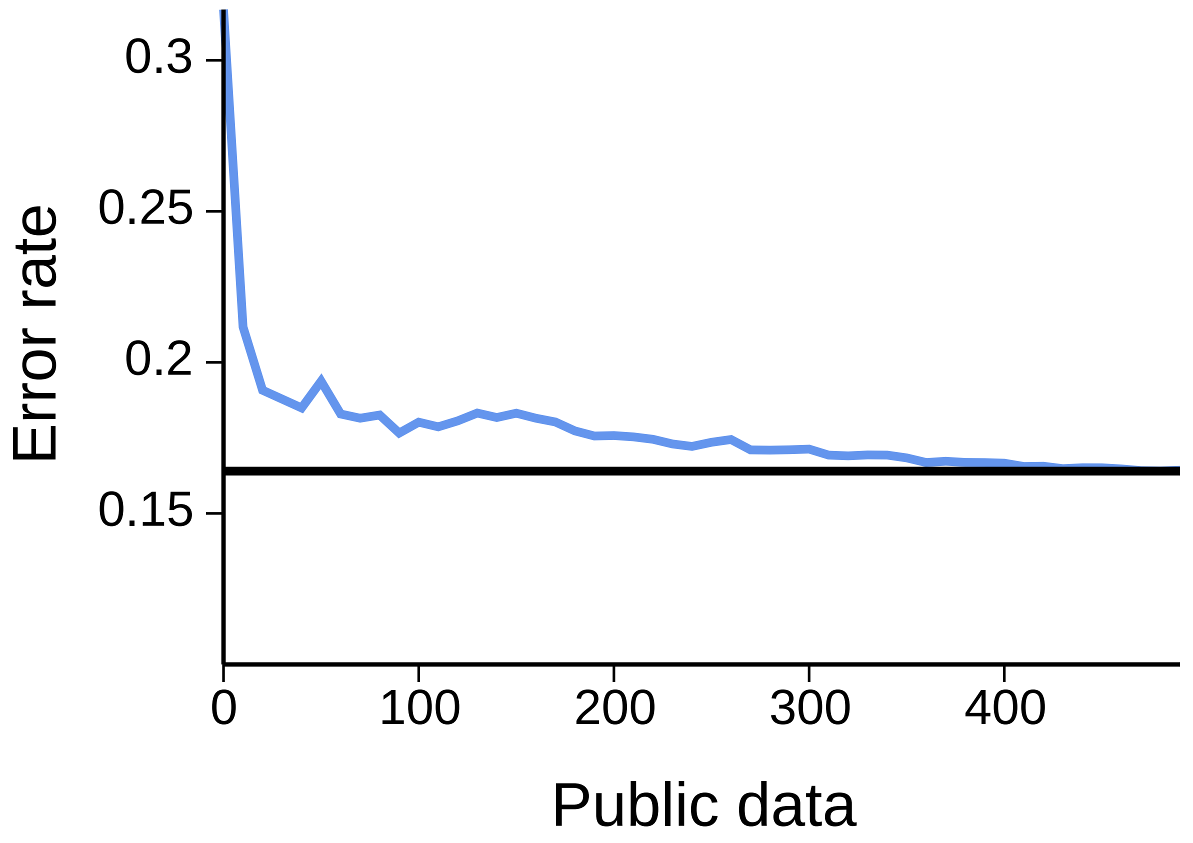
Configuring # of Iterations

The more iterations run, the less accuracy in each measurement!
Picking the right number of iterations could have been a problem.



Integrating Public Information

Publicly available data can be integrated naturally as a prior. Tossed a bunch of extra public examples in with SYNTH (@0.1):



Logistic Regression Wrap-up

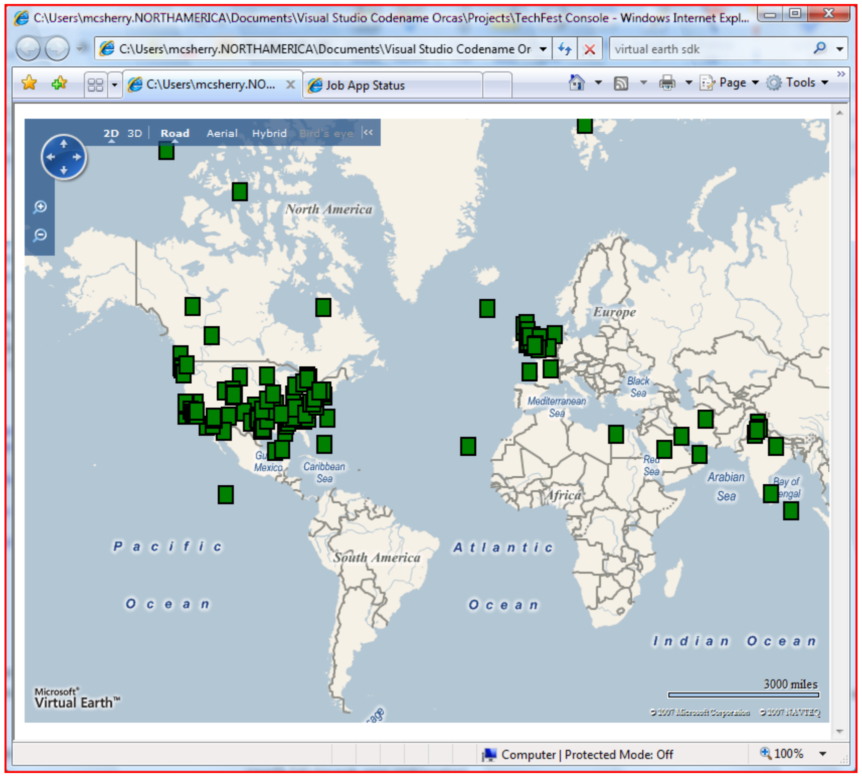
Probabilistic inference was certainly able to improve the heuristic.

1. Accuracy went up, using PI versus the last measurement.
2. Concentration went up too, as PI knew when it was right.
3. Prior knowledge was useful / useable.
4. We actually got the answer to an un-asked question.

Problems / Next steps:
Inference is hard. We used MCMC, but it takes hand-holding.
Tight integration can improve heuristic more (active learning).

Ex 2: MAP Synthesis

Ongoing project to synthesize data over "hierarchical" domains:


C:\Users\mcsherry.NORTHAMERICA\Documents\Visual Studio Codename Orcas\Projects\TechFest Console - Windows Internet Expl...
C:\Users\mcsherry.NORTHAMERICA\Documents\Visual Studio Codename Or
virtual earth sdk
C:\Users\mcsherry.NO...
Job App Status
Page
Tools
2D 3D Road Aerial Hybrid Bird's eye
North America
Europe
Black Sea
Mediterranean Sea
Caribbean Sea
Red Sea
Arabian Sea
Bay of Bengal
Africa
South America
Pacific Ocean
Atlantic Ocean
Indian Ocean
Microsoft Virtual Earth
3000 miles
Computer | Protected Mode: Off
100%

Intuitively, we measure counts at all powers-of-two granularities, then we try to find a data set that matches the observed counts.

A Measurement Framework

Imagine the data domain D can be hierarchically decomposed. We'll think about records as strings (sequences of characters).

For each prefix (sub-domain), we will take a noisy count. If it is "convincingly non-zero", we continue with children.

```
public static void Measure(PINQueryable<string> input)
{
    var measurement = input.Count(epsilon);
    if (measurement > threshold / epsilon)
    {
        var parts = input.Partition(characters, x => x[0]);
        foreach (var character in characters)
            Measure(parts[character].Select(x => x.Substring(1));
    }
}
```

Could take many measurements, from less to more accurate. Once recursion terminates, take one last accurate measurement.

Probabilistic Inference (1)

Recall that we know the exact conditional distribution of any noisy count measurement made of some data set A.

$$\text{Pr}[M(A) = x] \;\propto\; \text{exp}(-\epsilon \times |\text{Count}(A) - x|)\;.$$

For data set A the conditional probability of our observations is

$$\text{Pr}[obs|A] \;\propto\; \prod_i \text{exp}(-\epsilon \times |\text{Count}(A_i) - obs_i|)\;.$$

Our approach to synthesis is to find a data set A optimizing this. We take logs and switch to a minimization problem to simplify.

$$\text{Fit}(A) \;=\; \sum_i |\text{Count}(A_i) - obs_i|\;.$$

Probabilistic Inference (2)

Let $i \leq j$ mean that i is a prefix of j, and let $\mathsf{Fit}_i : \mathbb{R} \rightarrow \mathbb{R}$ be

$$\mathsf{Fit}_i(x_i) \quad = \quad \min_{\#(A_i)=x_i} \sum_{i \leq j} |\#(A_j) - obs_j| \ .$$

We can write Fit_i as a function of obs_i and Fit_{ic} for characters c:

$$\mathsf{Fit}_i(x_i) \quad = \quad |x_i - obs_i| + \min_{\sum x_{ic}=x_i} \sum_c \mathsf{Fit}_{ic}(x_{ic}) \ ,$$

The functions Fit_i can be computed using a dynamic program: Start at leaves (terminal observations) and work up to the root. From the root we can go back down, allocating counts optimally.

Probabilistic Inference (3)

Recall that we need to do the following a lot:

$$\mathsf{Fit}_i(x_i) \;=\; |x_i - obs_i| + \min_{\sum x_{ic}=x_i} \sum_c \mathsf{Fit}_{ic}(x_{ic}) \;.$$

The functions could be very complicated, and this would be hard. In fact, they are piecewise-linear and have non-decreasing slopes.

Minimizing Fit_i is just minimizing $|\mathsf{Fit}_i'|$. We will use Fit_i' instead.

1. Adding functions takes the union of their interval boundaries.

2. "Optimizing" functions is harder, but: all Fit_{ic}' must be equal. Fit_i' equals y at the sum of the x where the Fit_{ic}' equal y.

$$\mathsf{Fit}_i'^{-1}(y) = \sum_c \mathsf{Fit}_{ic}'^{-1}(y)$$

   Inverting step functions is easy (swap values and boundaries).

Synthetic Data wrap-up

It's up and running against enormous data sets. (DryadLINQ!)

Synthetic Search Query Logs:

Lots and lots of strings. And URLs (strings) that got clicked.

Synthetic frequency is key. False negatives $>>$ false positives.

Synthetic LEHD data:

Lots and lots of pairs of lat/lon coordinates (four values).
Viewed as strings via 4-dimensional quad-tree coordinates.

Sub-population (prefix) counts are important. Not leaves.

PINQ and Prob. Inference (1)

PINQ is a “privacy-safe” declarative programming language.
Privacy-naive programmers can write privacy-safe programs.

But:

1. It isn’t obvious that they will write productive programs.
2. It isn’t obvious that they know what to do with outputs.

Probabilistic Inference can be brought to bear on both of these.

1. Probabilistic inference supplies posteriors over questions.
2. Active learning selects queries to concentrate posteriors.

Can we automate these to support privacy/probability-naive users?

PINQ and Prob. Inference (2)

PINQ presently exposes a protected `PINQueryable<T>` type:

1. Supports transformations of protected data.
2. Supports DP aggregations of protected data.

Declarative nature of the observations provide simple posteriors:

$$\Pr[\text{ obs } | \text{ data }] \propto \Pr[\text{ obs } | \text{ Aggr(Trans(data)), epsilon }] .$$

Ad-hoc differentially-private computations would be much messier.

PINQ and Prob. Inference (3)

PINQ could expose a protected `PINQueryable<T>` type:

1. Supports transformations of protected data.
2. Supports DP aggregations of protected data.
3. Supports any LINQ query over inferred data.

Programmers write C#/PINQ programs; Maybe with AL help.
PINQ captures each (query, result) pair. Does Prob. Inference.

At any point (at zero-DP cost): posterior over any LINQ query:

1. Maximum Likelihood Estimates. (programs can run)
2. Posteriors supply confidence. (accurate enough?)

Probabilistic Inference wrap-up

Probabilistic Inference and Differential Privacy fit very well.

Use Probabilistic Inference whenever possible. Silly not to.