MSCS Thesis

Update 1

Andrea Klein

Problem Statement

- Goal: learn an unknown function f(I) = O between two probability distributions. Given a new sample from distribution I', predict the corresponding output distribution O' = f(I')
- Training data:
  - M pairs of <I, O> examples
  - I and O are length-eta samples from the input and output distributions, respectively
- Prediction for f(I') takes the form of a weighted sum of training output distributions; those with inputs similar to I' are weighted more heavily

Main Implementation Tasks Completed

1. Recreate 1-dimensional training distributions described in ICML proceedings
2. Sample distributions to create training data
3. Fit training samples to nonparametric density series estimators
4. Compute distances between training distributions and test distributions
5. Use distances to compute weights and build estimated output for test inputs

Toy Data

I have recreated Junier's training <input, output> distributions <p,q>, which were of the following form:

$$p(x) = \frac{1}{2} g(x; \mu_1, \sigma_1) + \frac{1}{2} g(x; \mu_2, \sigma_2)$$

$$q(x) = \frac{1}{2} g(x; 1 - \mu_1, \sigma_1) + \frac{1}{2} g(x; 1 - \mu_2, \sigma_2)$$

where:

Normal PDF

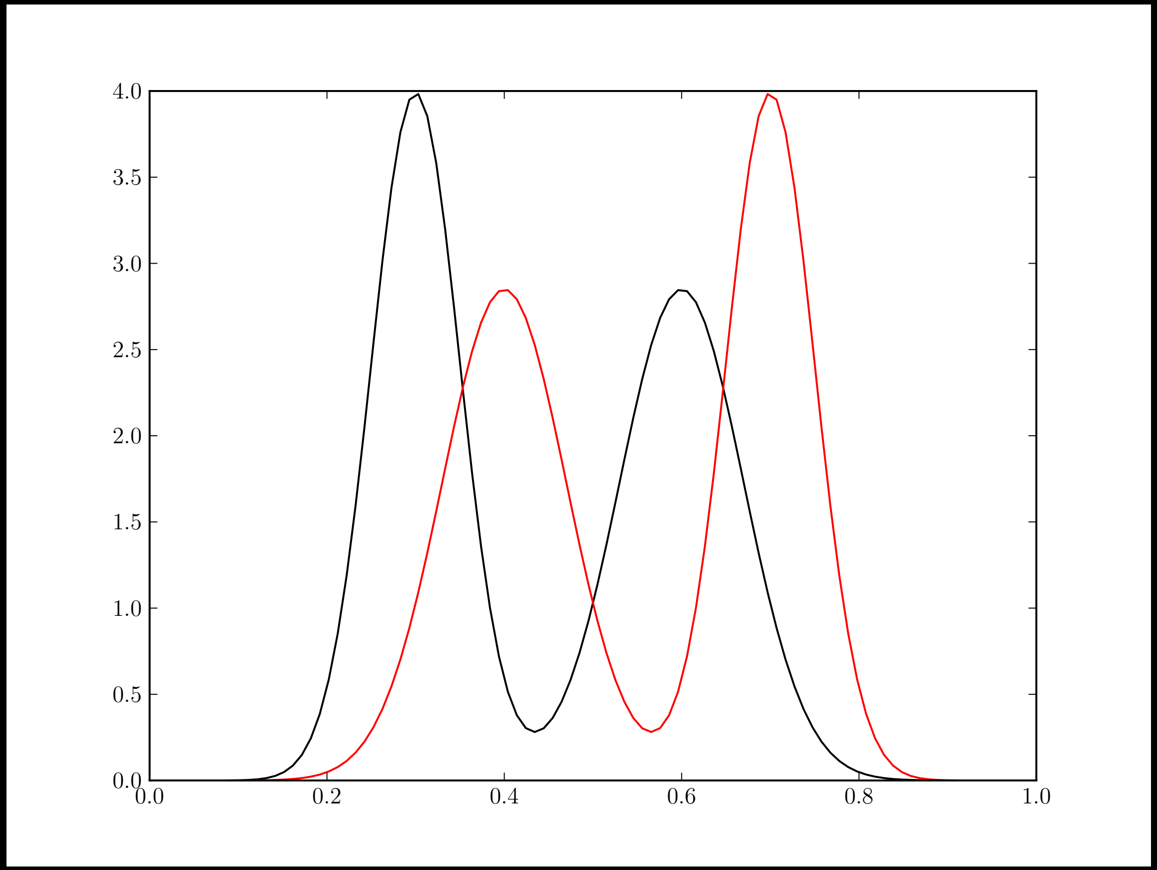
Normal CDF

$$g(x; \mu, \sigma) = \frac{\frac{1}{\sigma}\phi\left(\frac{x-\mu}{\sigma}\right)}{\Phi\left(\frac{1-\mu}{\sigma}\right) - \Phi\left(\frac{-\mu}{\sigma}\right)}$$

$$\mu_1, \mu_2 \sim \mathrm{Unif}[0,\ 1] \qquad \sigma_1, \sigma_2 \sim \mathrm{Unif}[.05,\ .1]$$

Toy Data

The functions look like this. Note that q(x) is just p(x) flipped about x = .5.



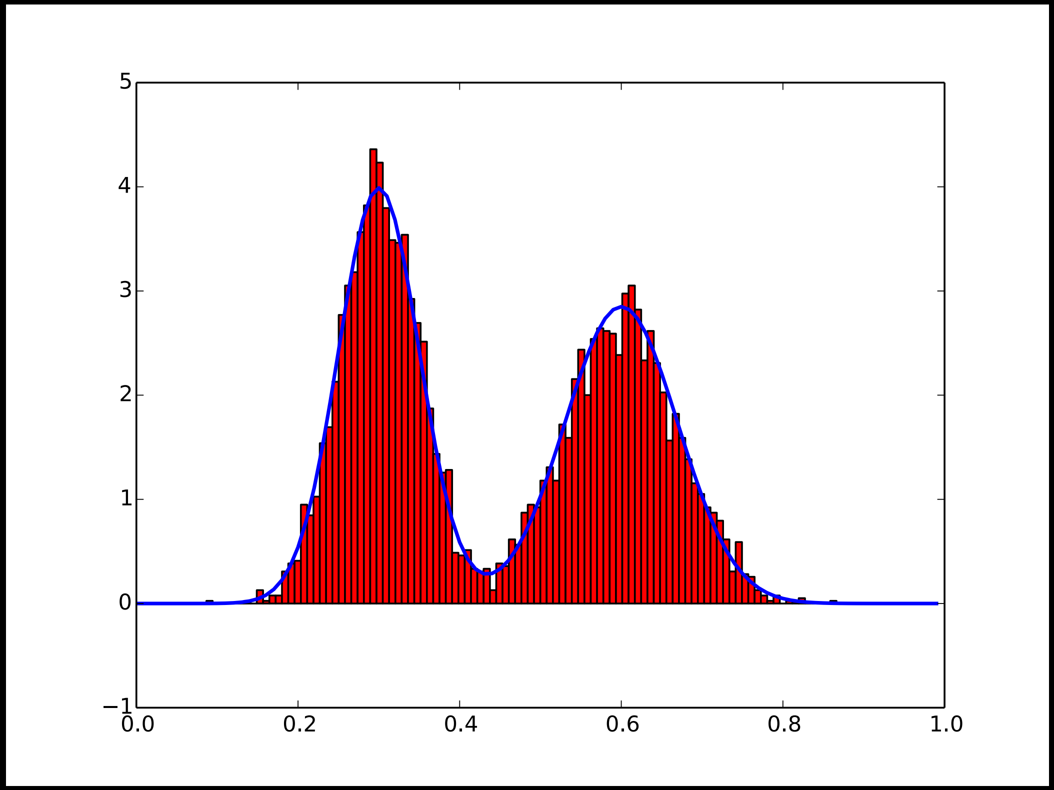
mu1, mu2 = .3, .6
sig1, sig2 = .05, .07

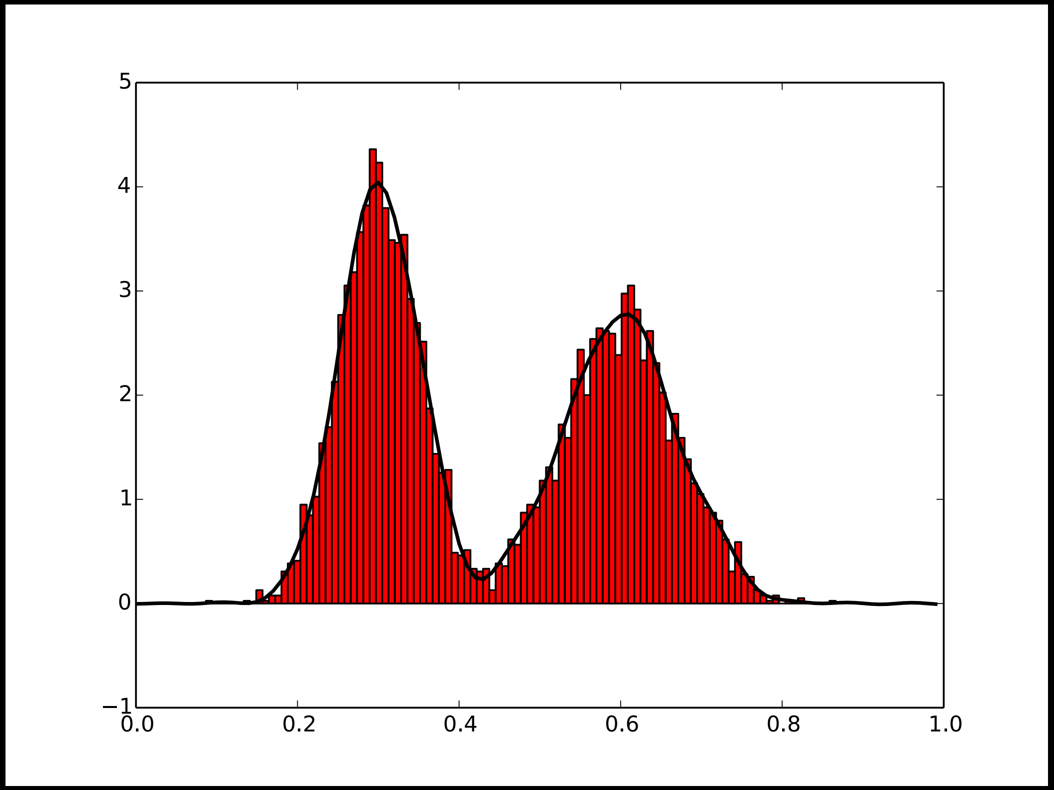
$$p(x) = \frac{1}{2}g(x; \mu_1, \sigma_1) + \frac{1}{2}g(x; \mu_2, \sigma_2)$$
$$q(x) = \frac{1}{2}g(x; 1-\mu_1, \sigma_1) + \frac{1}{2}g(x; 1-\mu_2, \sigma_2)$$

Toy Data

After creating the training distributions with randomly selected parameters, I draw samples from them (via rejection sampling) to create the actual training data (see example at left).*

In practice, you'd only have the samples, so it's necessary to approximate the parent distribution. I used a 20-term** orthogonal series estimator to recreate the distribution the sample came from (see right).



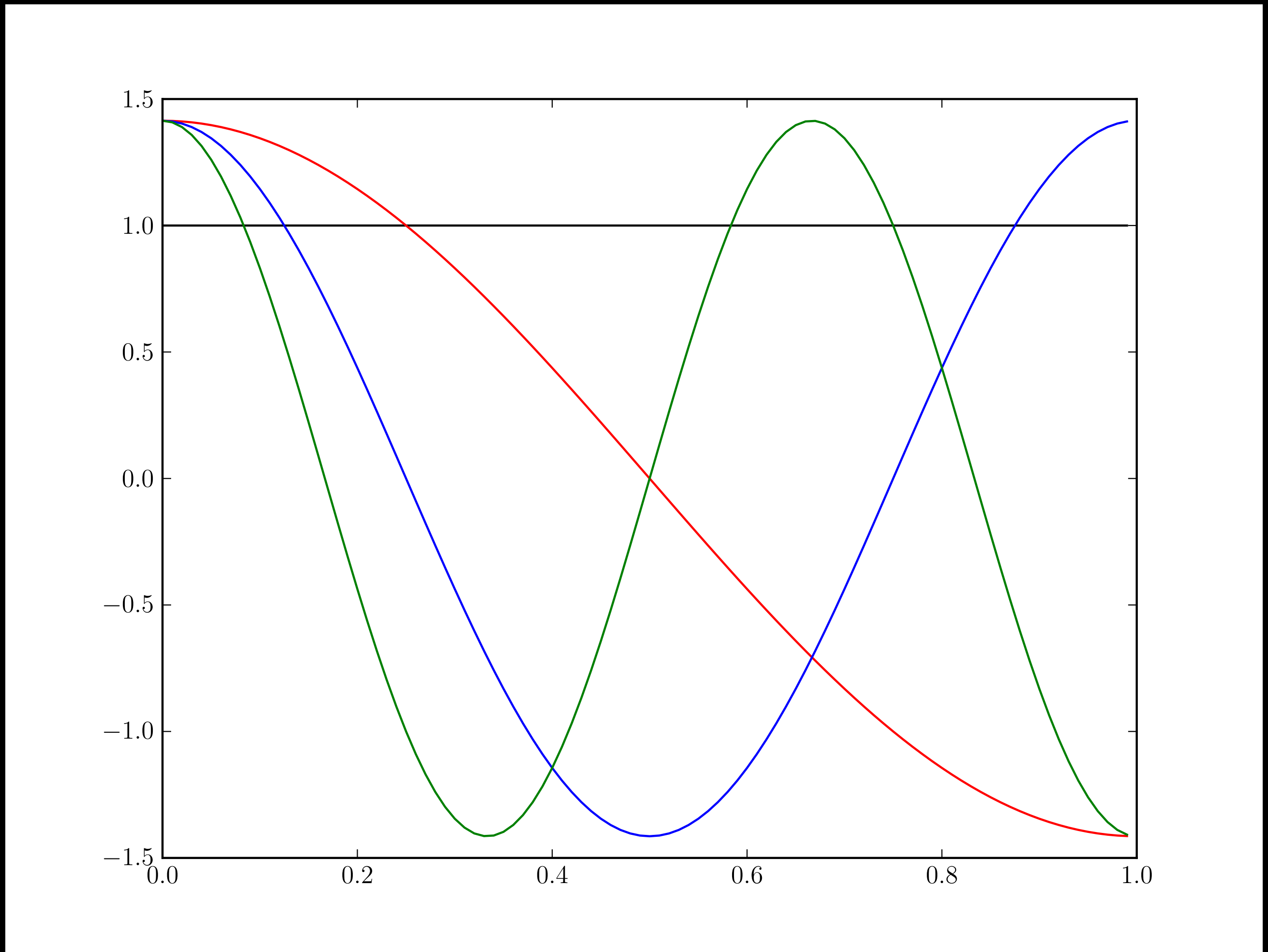


* I'll ignore the time complexity of data generation, since for real applications, the data will be provided

** Note that the accuracy of the estimator depends on both the # of samples provided and the # of terms used in the series

Distribution Approximation

I have adopted an orthogonal basis of cosine functions (phi) that span the space of square-integrable function f on [0, 1]. The first four are plotted below:



$$\{\phi_0(x) = 1, \phi_j(x) = \sqrt{2}\cos(\pi j x), j = 1, 2, \ldots\}$$

Distribution Approximation

The approximate distributions are built as a weighted finite sum of the basis functions, which can approximate any function in their space to arbitrary accuracy:

$$f_J(x) := \sum_{j=0}^{J} \theta_j \phi_j(x), 0 \le x \le 1$$

$$\theta_j = \int_0^1 \phi_j(x) f(x) dx.$$

The approximate function f_J is called an "orthogonal nonparametric series density estimator" (i.e. a Fourier series).

Distribution Approximation

The coefficients (theta) are really just expectation values. So I can compute the coefficient corresponding to a given basis function phi by evaluating phi on all the samples and taking the average:

$$\theta_j = \frac{1}{n} \sum_{l=1}^{\hat{n}} \phi_j(X_l).$$

To get all the thetas, you have to evaluate each phi on each sample from each distribution. So, to make a series with T terms for M training pairs with eta samples/distribution requires a one-time overhead of O(T * M * eta).

The Distance Function

The ICML proceedings use an L1 norm to compute the "distance" between two nonparametric estimators:

$$D(\tilde{P}_i, \tilde{P}_0) = ||\tilde{p_0} - \tilde{p}_i||_1 = \int |\tilde{p_0}(x) - \tilde{p}_i(x)|dx$$

Unfortunately, this requires explicitly computing the integrand and then numerically integrating it from 0 to 1.

The overhead is quite significant: for example, computing D(Pi, Po) from a fixed Po to 900 Pi values took an average of 0.46 seconds per pair, or 7:02 overall. This is unacceptably slow!

The Distance Function

Consequently, I've switched to the **L2 norm**. This allows me to:

1. work with a coefficient representation of the distribution estimators
2. simply compute their distance as the sum of squared differences of the coefficients

Computing the same 900 pairwise distances now takes 0.05 s, or an average of 5.55e-05 s per pair. That's ~10,000x faster.

Kernel Weights

Officially, the weights in the estimator are given in terms of a kernel function as follows:

$$W(\tilde{P}_i, \tilde{P}_0) = \begin{cases} \frac{K(\frac{D(\tilde{P}_i, \tilde{P}_0)}{b})}{\sum_{j=1}^{M} K(\frac{D(\tilde{P}_j, \tilde{P}_0)}{b})}, & \text{if } \sum_{j=1}^{M} K(\frac{D(\tilde{P}_j, \tilde{P}_0)}{b}) > 0. \\ 0, & \text{otherwise.} \end{cases}$$

bandwidth

The ICML proceedings use the triangle kernel. By working instead with a **Gaussian (RBF) kernel**:

1. Fewer "if" statements - don't have to check whether the input to the kernel is less than 1 or the output of the kernel is 0

2. Less similar training examples are penalized more heavily, reducing over-smoothing

Bandwidth

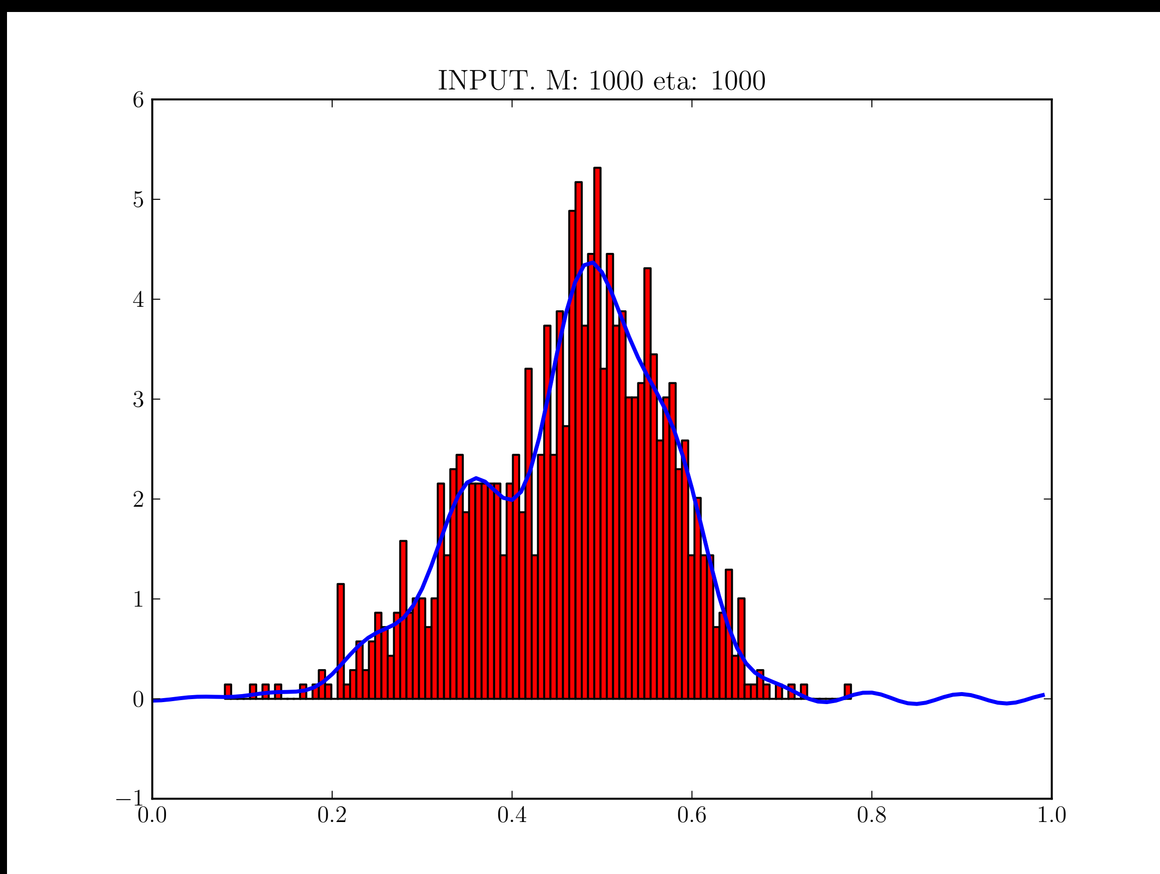
The bandwidth parameter b is the principal means by which to control over-smoothing.

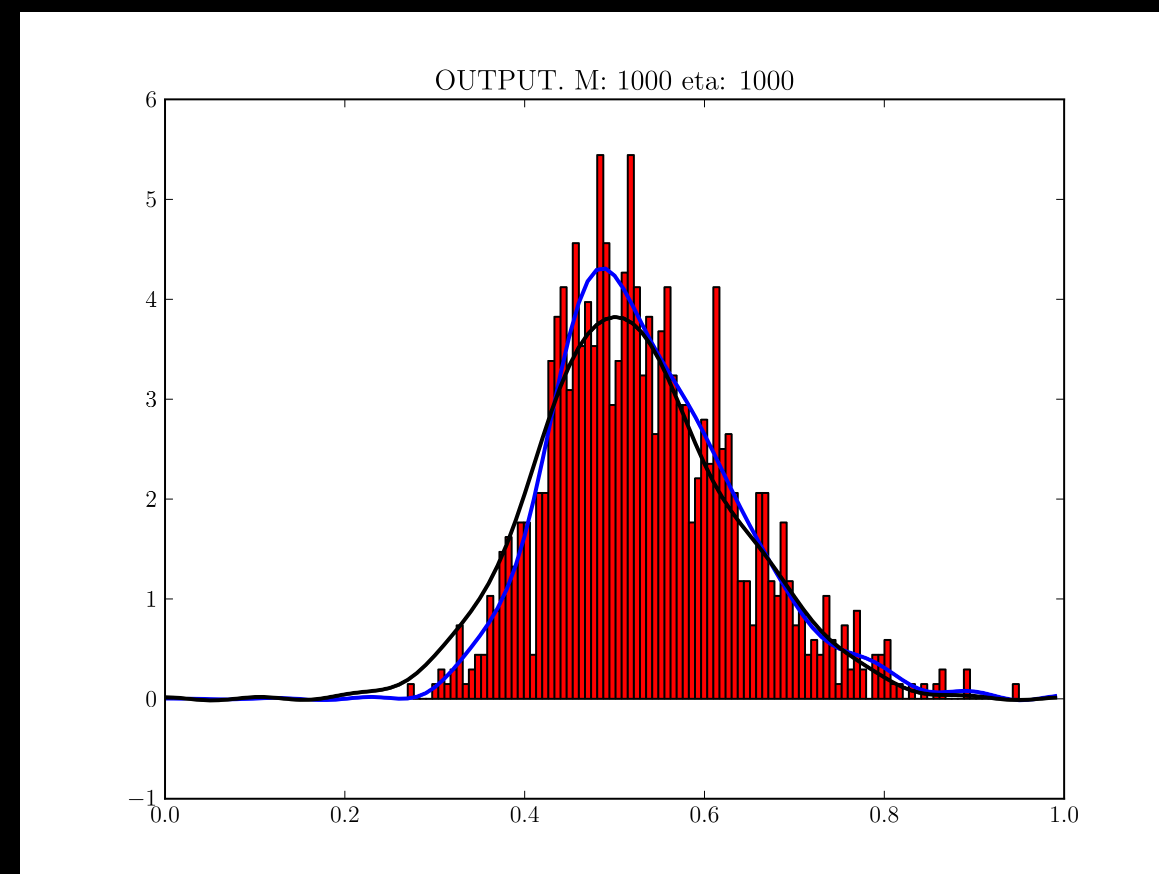
For now, I try b values in the set [.15, .25, .5, .75, 1, 1.25, 1.5]. (In the future, i.e. for unknown datasets, there will be a more sophisticated rule.)

In order to pick best b, I cross-validate b on a “holdout” training dataset consisting of about 10% of my total training data. The b with the lowest mean L2 error wins.

This step requires regressing on the holdout data and computing the L2 error on the resulting estimators. (Note that computing the weights + cross-validating must be repeated 5 times for different values of b.)

Estimator Performance





Example performance of a fully-trained estimator. "True" fits in blue; estimated output fit in black.