Final Project: Bayesian Optimization for ABC Methods

IL181.014 - Professor Carl Scheffler

Guy Davidson

December 21, 2018

Note: all plural in this paper is the academic plural, I did not collaborate with anyone on this work. Additionally, my apologies, but I failed to add figure captions. All figures are described in the body of the paper.

1 Introduction

Approximate Bayesian Computation methods are used to perform posterior inference in models with an intractable likelihood function and access to the generative process transforming model parameters to observations. (ABC, Tavare et al., 1997; for recent reviews, see Harting et al., 2011; Marin et al., 2011, and Lintusaari et al., 2016). Given observed data D, a prior $P(\theta)$ and a data generating simulator $M(\theta, \dot{})$, we perform inference by sampling $\hat{\theta} \sim P(\theta)$, generating data $\hat{D} = M(\hat{\theta}, \dot{})$, and evaluating the distance between the observed data and the simulated one, assigning probability proportionally, $P(\theta = \hat{\theta} - D) \propto \rho(D, \hat{D})$. To perform ABC rejection sampling, we only accept samples where the distance is sufficiently small, $\rho(D, \hat{D}) < \epsilon$, and noting that it appears reasonable to consider the set of parameters with the smallest observed distance as the maximum likelihood estimate for the true parameters. Since the probability of landing close to the target data diminishes rapidly as the dimensionality of the data increases, it is common to choose a summary function $S(D)$ and to compute distances over the summarized data, $P(\theta = \hat{\theta}) \propto \rho(S(D), S(\hat{D}))$. One alternative to ABC rejection sampling is ABC-MCMC (Marjoram et al., 2003), where we provide a transition function $q(\theta, \theta')$, and perform Markov Chain Monte Carlo-style transitions. From a current estimate of the parameters θ_t, we propose θ' by sampling from $q(\theta)$, and evaluate it as described previously, estimating $P(\theta' - D) \propto \rho(D, D')$. As in standard MCMC methods, the acceptance probability is a product of three terms $a = \min\left(1, \frac{P(\theta' - D}{P(\theta_t - D} \frac{P(\theta')}{P(\theta_t)} \frac{q(\theta_t, \theta')}{q(\theta', \theta_t)}\right)$. If we accept, we set $\theta_{t+1} = \theta'$, and otherwise we retain the previous value, $\theta_{t+1} = \theta_t$.

While these procedures are reasonably simple to describe, performing inference using ABC methods can be quite tricky. The choice of prior can substantially impact performance, as an overly sparse prior can result in over-exploration of the parameter space, and an overly narrow one can result in missing optima in other regions of probability space. The choice and parameterization of the summary function, if one is used, can be equally important, as can the distance measure and its translation to probability. ABC methods offer no clear suggestion for how to optimize such hyperparameters. This paper reports a proof of concept using Bayesian optimization methods to select hyperparameters, exploring these ideas using a toy problem previously designed to investigate the use of variational autoencoders (VAEs) as summary functions for ABC methods. This work will proceed by describing Gaussian process-based Bayesian optimization methods, followed by additional details over the aforementioned toy problem. It will then discuss the choices required to employ this method to optimize different parameters in ABC and ABC-MCMC settings, as well as a few pitfalls encountered while performing these experiments.

2 Gaussian Processes and Bayesian Optimization

Bayesian optimization treats the target function to be optimized as a black box, modeling it using some form that is easier to reason about, and updating the model using observations generated by running the function with a particular set of inputs. To perform Bayesian optimization, we require two functions: a model function representing our belief over the target function, and an acquisition function, proposing the next point to evaluate with the target function. Within some execution budget, we use the acquisition function to suggest a new point to evaluate, evaluate this point using the target function, and update our model using this new point and its corresponding output. The use of Bayesian optimization to optimize machine learning hyperparameters was suggested and popularized by Snoek et al. (2012; for a thorough review, see Shahriari et al., 2016). The authors propose using a Gaussian process as the prior, or model for the target function and offer several potential acquisition functions. Our discussion in class around GPs centered on variational methods for fitting Gaussian processes to data, an approach most useful in big data regimes; given that operate from a small number of target function samples, this is not necessary to approach for this problem. While Gaussian processes are infinitely flexible function priors and contain parameters that drive their approximations (the observations from the target function), they are not, in a strict sense, variational approximations, as their form is not derived from minimizing the variational free energy (or a KL-divergence) between the target function and the GP approximation. However, as universal function approximators, they are well-suited as priors to perform inference over our target function.

Gaussian processes are defined by two function choices: the mean function (or prior), $m(x)$, and the kernel function $k(x, x')$, describing how points covary with one another. For the experiments used below, we employ the standard zero mean prior and the Matrén 5/2 kernel suggested by Snoek et al. (2013). We use the upper confidence bound (UCB) acquisition function, adapted to a lower confidence bound since this is a minimization problem, and using the value $\kappa = 2$ for its hyperparameter. To generate samples from the acquisition function, we follow the procedure suggested in a tutorial[1] (among other places), of sampling a few random points (from a provided distribution), and optimizing (using an auxiliary optimizer, in this case LBFGS, Limited-memory Broyden–Fletcher–Goldfarb–Shanno) the acquisition function around each one, to find local minima. The best candidate of those is then evaluated using the target function, which then allows us to update the GP, and progress to the next iteration. To seed this process, the GP is initialized using several randomly selected points and their target function values, providing some information about the target before beginning to choose points using the acquisition function.

Note that for the choice of UCB hyperparameter, as well as other design decisions, such as the choice of acquisition function and optimizer, the sample proposal distribution, and the number of samples to evaluate at each iteration, there is no trivial way to intelligently select these parameters either. Just like in Christopher Nolan's *Inception*, we don't know how much further up there is to optimize. While this paper will not consider this query beyond a philosophical one, it could perhaps make sense to consider a hierarchical setting for Bayesian optimizers, akin to what Andrychowicz et al. (2016) propose in their excellently titled *Learning to learn by gradient descent by gradient descent.*

[1]http://pyro.ai/examples/bo.html

3 Experiment Setup

To perform a few proofs of concept, we return to the toy problem previously investigated in Davidson's (unpublished) previous assignment for this course. In this toy problem, we generate data by selecting mixture weights for digits (in all examples below, 0.3 to zero and 0.7 to one), and generate data by sampling random MNIST (LeCunn & Cortes, 2010) digits and taking their weight sum according to the mixture probabilities. ***Figure 1*** provides an example of such digits. While assigning a likelihood to a set of parameters, in this case, is quite tricky, repeating the generative process with a separate set of weights is not. The ABC problem, then, is to infer the weights for each digit from a list (0/1 in the simplistic experiments, 0/1/2/3 in more complicated ones).

The previous assignment investigated using variational autoencoders (VAEs; Kingma & Welling, 2013) as the summary function for ABC and ABC-MCMC on this problem and found some promising initial results. VAEs create an autoencoder model, where the encoder transforms the input into some latent space, and the decoder attempts to recover the original input, starting from a random sample around the transformed value in the latent space. The loss function for such a model is precisely the two components of the ELBO: a reconstruction loss, the binary cross-entropy between the input and the reconstructed output, and a regularizing term, the KL-divergence between the estimated latent parameters and the latent prior (often chosen to be a zero-mean, unit-variance Gaussian). Variational autoencoders have been shown to be quite effective at reproducing inputs, given sufficiently expressive encoder and decoder modules, which means they learn an effective latent representation to encode essential information about the input and allow the decoder to reconstruct it. This observation motivated the previous work, which employed the encoder half of a trained VAE as a summary function over images in this problem.

While the previous work suggests that using a convolutional VAE produces better results than a VAE using a standard neural network, the following experiments were performed with a very basic multilayer perceptron (MLP) as the encoder and decoder. The encoder receives the input (28x28 pixels flattened to a 784-dimensional vector) and passes it through two layers, the first with 400 units, and the second split into two 20-unit layers, one representing the mean and the other the log-variance of the latent variables. The decoder receives a 20-dimensional latent representation and passes it through a hidden layer with 400 units, and then through a second layer to return to the 784-dimensional output space, at which point it can be restored to a 28x28 image. ***Figure 2*** provides an example of such reconstructions made by this VAE model.

We use two metrics to evaluate the quality of particular parameter proposals. The first is the distance between the latent representations of the observed training data and those of the simulated data, taking the mean pairwise distance between all images in each set. From running previous experiments, the best models tend to reach a mean distance of no less than 2.5 over the one hundred training images and 40-dimensional latent representations (20 mean variables, 20 log-variance variables). To evaluate the proposals in parameter space, we examine the absolute deviation between the proposed parameters $\hat{\theta}$ and the true parameters θ, summed over all digits. For instance, proposing weights of 0.25 to zero and 0.75 to one would result in an absolute deviation of 0.1. The former metric is the metric used by the ABC approaches to evaluate models; the latter mostly by the author to evaluate results.

4 Proof of Concept Experiment Results

We will now report the results of the four proof of concept experiments performed to preliminarily evaluate GP-based Bayesian optimization for hyperparameter selection. For each experiment, we will describe the parameters we are attempting to optimize, nuances and modifications in the optimization procedure, and a summary of results and plots observed.

4.1 ABC: Prior parameters

Given that the model attempts to learn a discrete probability weight assignment, the natural prior is a Dirichlet distribution of the appropriate dimension - two dimensions in the 0/1 case, and four in the 0/1/2/3 one. The parameters of a Dirichlet distribution are constrained to be positive, and in this experiment, we further constrain to the real line between $[1, 20]$. The GP was initialized with five random points sampled uniformly from this range, before providing some additional forty iterations.

One additional meaningful consideration became the choice of the target function. Clearly, it derives from the result of an ABC execution with the proposed prior parameters, but how should it be evaluated? We ended up settling on taking the mean of some top k results, trying to conform to a Goldilocks principle, taking neither too few points, which would allow for flukes, or too many, which would dilute the signal, since we do eventually care about the best results. In the two-digit example, each ABC execution ran for twenty iterations, and we evaluated all twenty values it returned, with both numbers increased to one hundred for the four-digit case, but future experiments tended to use more iterations per target evaluation and a smaller proportion of them.

Figures 3-5 demonstrate example results over the two-digit problem and ***Figures 6-9*** over the four-digit version of the task. We fairly consistently see that the ABC with a learned prior from the GP produces much better overall results, while also competitive when we take a smaller subset. ***Figures 10-11*** show density plots of the top 1000 results, where ***Figure 10*** takes all one thousand iterations of both the regular ABC and GP-prior ABC, while ***Figure 11*** allows the basic ABC 10,000 iterations, taking the top 1000 of those (note that we do not provide density plots of the two-digit problem, as its two digits must add to one, and this constraint to the line $x + y = 1$ would result in a fairly boring density). Their approximate equivalence demonstrates that the GP learns appropriate prior hyperparameters, allowing to produce equivalent results in an order of magnitude fewer samples (ignoring, of course, the myriad of samples taken to learn the GP).

4.2 ABC: Automatic relevance determination (ARD) distance weights

For the second experiment, we tweak the distance function used to evaluate generated parameters. Rather than using a standard Euclidean distance, we learn a scaling factor for each coordinate in the latent representation, which comes from the assumption that not all latent variables are equally representative of the outcomes we are concerned with. This became a much harder problem computationally, as the GP must now be 40-dimensional, rather than existing in two or four dimensions. The learned weights were constrained to the range $[0.01, 100]$, and to make this problem simpler, and only evaluate the choice of weights, the internal ABC model was provided with an appropriate prior, Dirichlet([3, 7]). This experiment used forty initial values for the GP, followed by forty fitting iterations, each iteration drawing 100 ABC samples and evaluating based on the top ten ones. On the four-dimensional problem, we run with one hundred initial values and one hundred Bayesian optimization iterations, with a $Dirichlet([3, 7, 1, 1])$ prior.

Note that evaluating the ARD weights is more complicated. Since they directly scale the distance measure, if we were to optimize based on it, it would encourage the weights to explore toward the constraint, rather than find an appropriate solution. Instead, we evaluate solutions based on the second measure described earlier, the absolute deviation of the parameters. This requires knowing the ground truth answer, which is only tractable in toy examples or devising an alternative evaluation measure independent of the distance function used. To test the weights proposed by the Bayesian optimization procedure, we initialize an ABC sampler with a flat prior and the ARD weights and compare it as in the previous experiment.

Figure 12 and ***Figure 13*** compare the results of 10,000 iteration ABC executions without and with the ARD weights. These results appear successful as well: the GP-ARD version learns slightly better top-100 results, and substantially better top-1000 results. The GP seemed to have a much harder time optimizing the four-digit case, as it didn't often appear to improve often, effectively devolving into a quasi-random search guided by the acquisition function. ***Figure 14*** and ***Figure 15*** demonstrate its relatively pedestrian performance, even if it managed to produce a better top overall set of parameters (as well as better top ten).

4.3 ABC-MCMC: Prior parameters

We move onto ABC-MCMC, returning to initially investigate the choice of prior parameters here. Given that the problem formulation, aside from being ABC-MCMC rather than ABC, is quite similar to the first experiment, I will detour to instead describe two issues I ran into, and how I resolved them.

First, a bug with the sampler. We use the SALT sampler (Director et al., 2017) discussed in the last assignment, which purports to generate proposals on the simplex. While their manuscript discusses using one of the probabilities in each proposal step and setting it to one minus the sum of the others, guaranteeing a proper simplex, their code[2] does not do that (to the best of my understanding). Particularly in the two-dimensional case, I encountered fairly substantial deviations fro the simplex if this step is omitted[3].

Second, a bug in my implementation. Our original implementation accidentally omitted one of the terms in the acceptance probability, the ratio of prior probabilities of each set of parameters. Added and implemented it.

Third, a problem with my distance to probability measure function. After discovering the Bayesian optimization routine fails to reach any meaningful results, we decided to plot one of my MCMC chains and see if we can discern why that might be. ***Figure 16*** visualizes the chain, where the x-axis represents each sample, the y-axis the probability assigned to the digit one (which should ideally converge to around 0.7), the color blue accepted samples, the color red rejected proposals, and the size inversely proportional to the score (lower, better scores plotted larger). Far too many poor proposals appeared to be accepted. We realized the culprit is probably my distance to probability function, for which we used $P(d) = \exp(-d)$, which does not normalize but does project to the unit interval. We realized that just as softmax functions sometimes employ an inverse temperature parameter to control the sensitivity to changes in value, so must this implementation, resulting in the function $P(d) = \exp(-\beta d)$. ***Figure 17*** plots the effect on the acceptance ratio (y-axis) of moving an increment (x-axis) in the wrong direction under a particular value of β, the inverse temperature parameter. Note that for $\beta = 1$, the default setting, increasing the score by one (a substantially negative change) would

[2]https://rdrr.io/cran/SALTSampler/

[3]I imagine they removed this step since they want to be working only in logits, rather than in actual probabilities, and verifying normalization requires returning to probability space. I could consider emailing them to inquire.

still result in acceptance probability a hair under 40% (in fact, of precisely $\frac{1}{e}$), whereas higher values of β drop the acceptance probability much faster. ***Figure 18*** demonstrates a sample chain with $\beta = 0.5$. As would be expected, the rejection rate is substantially higher, but the samples appear far more constrained to the plausible region of probability space. This change provides another parameter to optimize, leading us to...

4.4 ABC-MCMC: Prior and temperature parameters

In this last experiment, we attempt to jointly optimize over the ABC-MCMC prior distribution and temperature parameter. One problem repeatedly encountered with this setup was failures in the optimization of the acquisition function. Borrowing from the tutorial, I implemented the GP in a constrained space, only unconstraining to optimize. For some reason, in this setting, it led the model to get stuck in the 'corners' of the constrained parameter space, repeatedly examining permutations of parameters around the lower and upper bounds. As before, parameters were constrained to the interval $[1, 20]$, and this behavior continued even after we adapted the sampling to nudge away, sampling new proposals (before optimizing them) uniformly from the interval $[5, 15]$. The quickest workaround was to stop optimizing proposals on the acquisition function, again devolving into a quasi-random search.

In later experiments, we attempted to optimize a fourth parameter as well, a length-scale variable used by the SALT sampler in their sampling routine. This was guided by a hope it might lead to more effective sampling; if it did, it was not noticeable. The limited results achieved by the ABC-MCMC sampling experiments did not seem promising enough to merit reporting.

One additional consideration, again, comes to the choice of target functions. In preliminary evaluations of results from both the standard ABC-MCMC and the one using GP-proposed parameters, both chains appear to converge well ($\hat{R} < 1.1$ for all parameters, using PyMC3's (Salvatier et al., 2016) implementation), but unsurprisingly, the result from the GP, which had a narrower prior and higher inverse temperature, explored less of the posterior space, resulting in a lower effective sample size. It would perhaps be interesting to use these metrics in the Bayesian optimization target function, attempting to reward parameter settings that not only produce lower distances in the latent space but also healthier chains.

5 Conclusion

This work laid the ground for Bayesian optimization using Gaussian processes as a hyperparameter optimization method for ABC algorithms. While these approaches require substantially more computation to optimize the results, they appear to discover better parameter settings than naive ones when used with ABC methods and show some plausibility with ABC-MCMC solutions as well. It would be interesting to continue investigating these approaches on additional problems, and more rigorously evaluating the results, performing an ablation study over randomly generated tasks, rather than only evaluating on the 0-1 mixture toy problem. There are several different improvements that would be meaningful to implement and evaluate whether or not they improve performance. The one we would be most optimistic about is transforming the GPs to live in unconstrained space, and only transforming to constrained parameter space for the target function evaluations, as it might allow the GP to avoid optimizing the acquisition function into constraint boundaries. It would also be trivial to investigate an annealing schedule for the ABC-MCMC distance to probability function, perhaps allowing more exploration early in the execution of the chain while nudging toward exploitation later in the chain.

6 Code

The main Colaboratory notebook with my work is available here: https://colab.research.google.com/drive/1NOSeNym_tIbDa6EM1PdLoS2-5Qq0S7wO.

7 References

Andrychowicz, M., Denil, M., Colmenarejo, S. G., Hoffman, M. W., Pfau, D., Schaul, T., ... De Freitas, N. (2016). *Learning to learn by gradient descent by gradient descent.* arXiv: 1606.04474v2. Retrieved from https://arxiv.org/pdf/1606.04474.pdf

Director, H. M., Gattiker, J., Lawrence, E., & Vander Wiel, S. (2017). Efficient sampling on the simplex with a self-adjusting logit transform proposal. *Journal of Statistical Computation and Simulation*, *87*(18), 3521–3536. doi:10.1080/00949655.2017.1376063

Hartig, F., Calabrese, J. M., rn Reineking, B., Wiegand, T., & Huth, A. (n.d.). Statistical inference for stochastic simulation models-theory and application. doi:10.1111/j.1461-0248.2011.01640.x

Kingma, D. P. & Welling, M. (2013). *Auto-Encoding Variational Bayes.* arXiv: 1312.6114v10. Retrieved from https://arxiv.org/pdf/1312.6114.pdf

Lintusaari, J., Gutmann, M. U., Dutta, R., Kaski, S., & Corander, J. (2016). Fundamentals and Recent Developments in Approximate Bayesian Computation. *Systematic Biology*, *66*(1), syw077. doi:10.1093/sysbio/syw077

Marin, J.-M., Ceremade, C. P. R., Dauphine, P., Robin, P., & Ceremade, J. R. (2011). *Approximate Bayesian Computational methods *.* arXiv: 1101.0955v2. Retrieved from https://arxiv.org/pdf/1101.0955.pdf

Marjoram, P., Molitor, J., Plagnol, V., & Tavaré, S. (2003). Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, *100*(26), 15324–15328. doi:10.1073/PNAS.0306899100

Salvatier, J., Wiecki, T. V., & Fonnesbeck, C. (2016). Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, *2*, e55. doi:10.7717/peerj-cs.55

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & De Freitas, N. (n.d.). *Taking the Human Out of the Loop: A Review of Bayesian Optimization.* Retrieved from http://www.ibm.com/software/commerce/optimization/cplex-optimizer/

Snoek, J., Larochelle, H., & Adams, R. P. (2012). *Practical Bayesian Optimization of Machine Learning Algorithms.* Retrieved from https://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf

Tavaré, S., Balding, D. J., Griffiths, R. C., & Donnelly, P. (1997). Inferring coalescence times from DNA sequence data. *Genetics*, *145*(2), 505–18. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/9071603%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC1207814

8 Figures

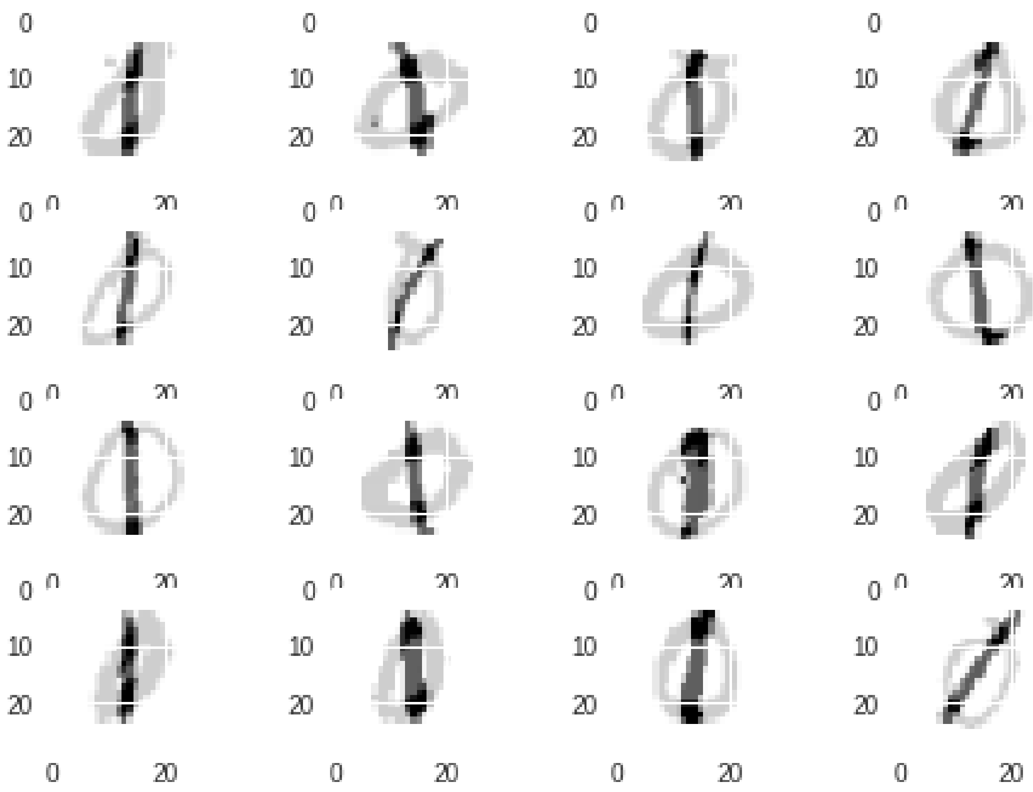


Figure 1:

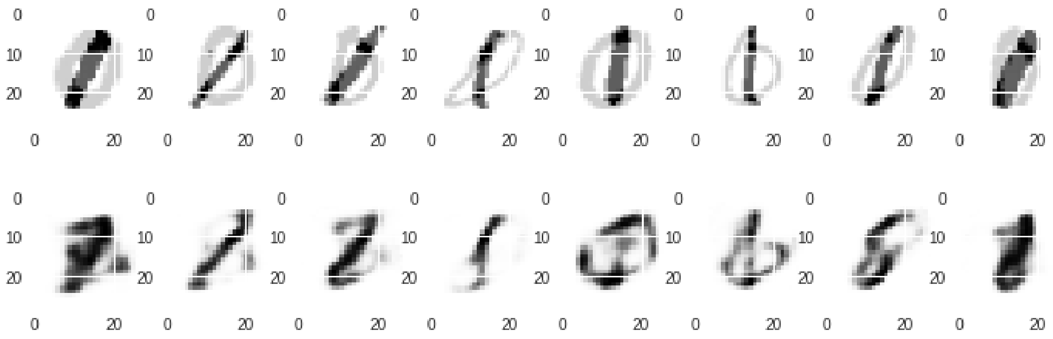


Figure 2:

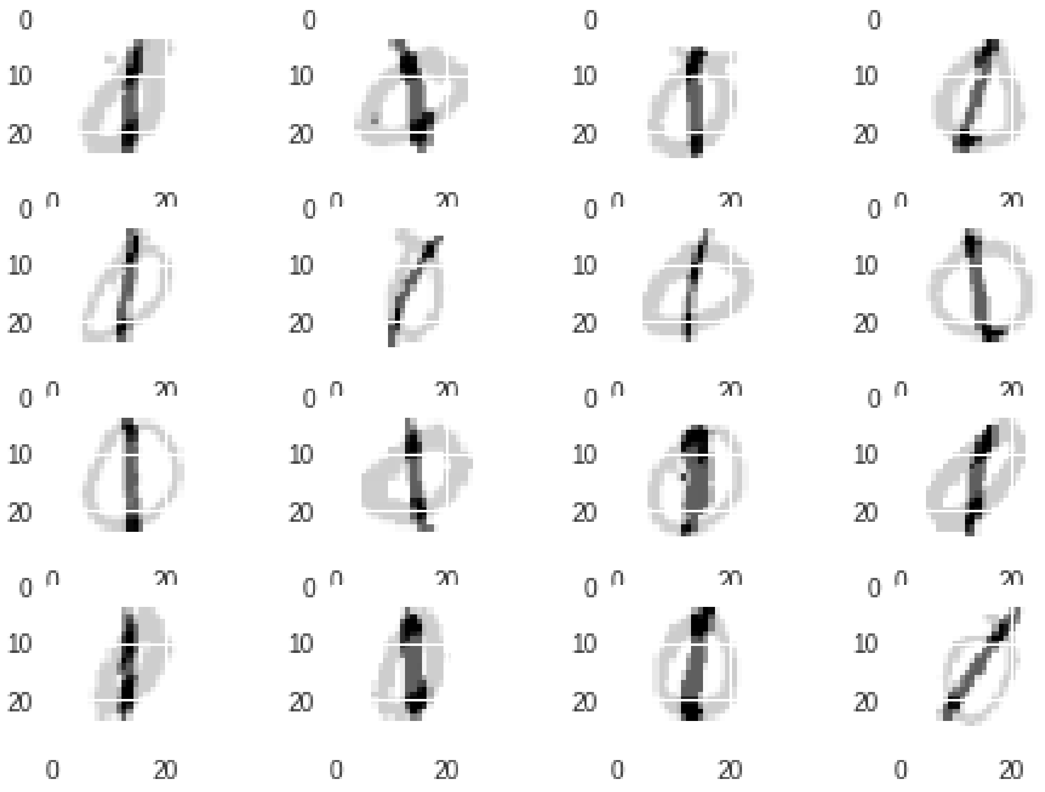


Figure 3:

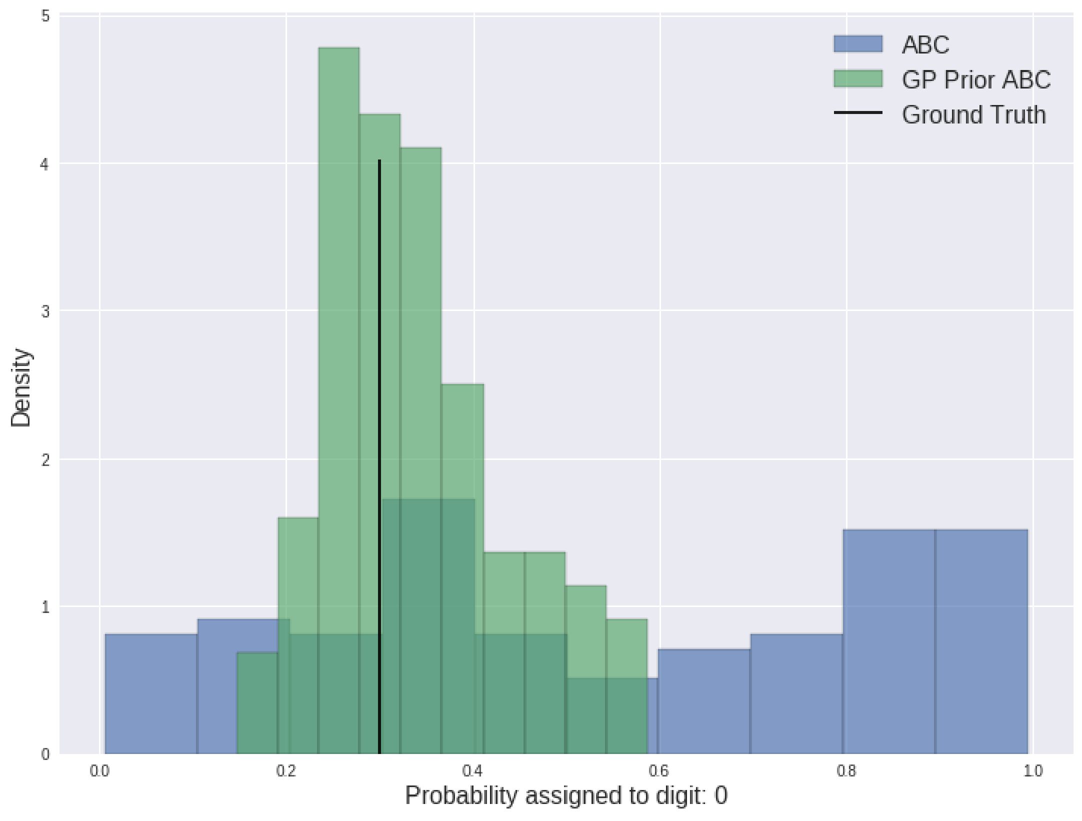
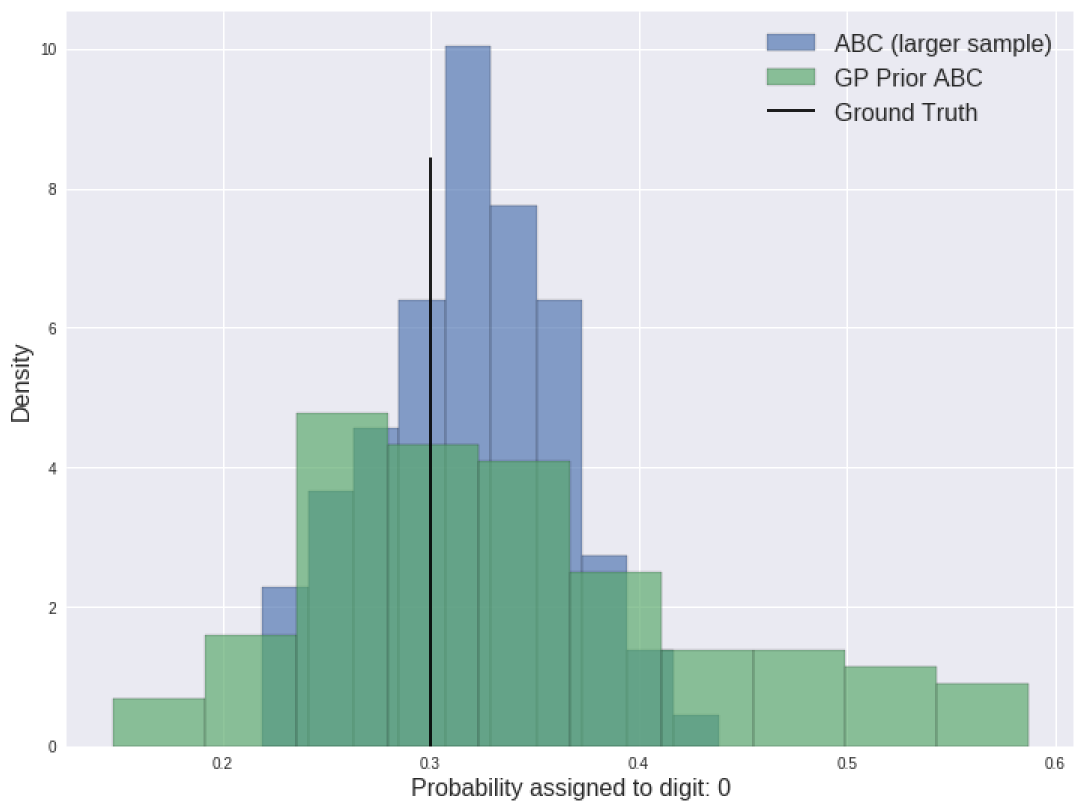


Figure 4:



ABC (larger sample)
GP Prior ABC
Ground Truth
Density
0
2
4
6
8
10
0.2
0.3
0.4
0.5
0.6
Probability assigned to digit: 0

Figure 5:

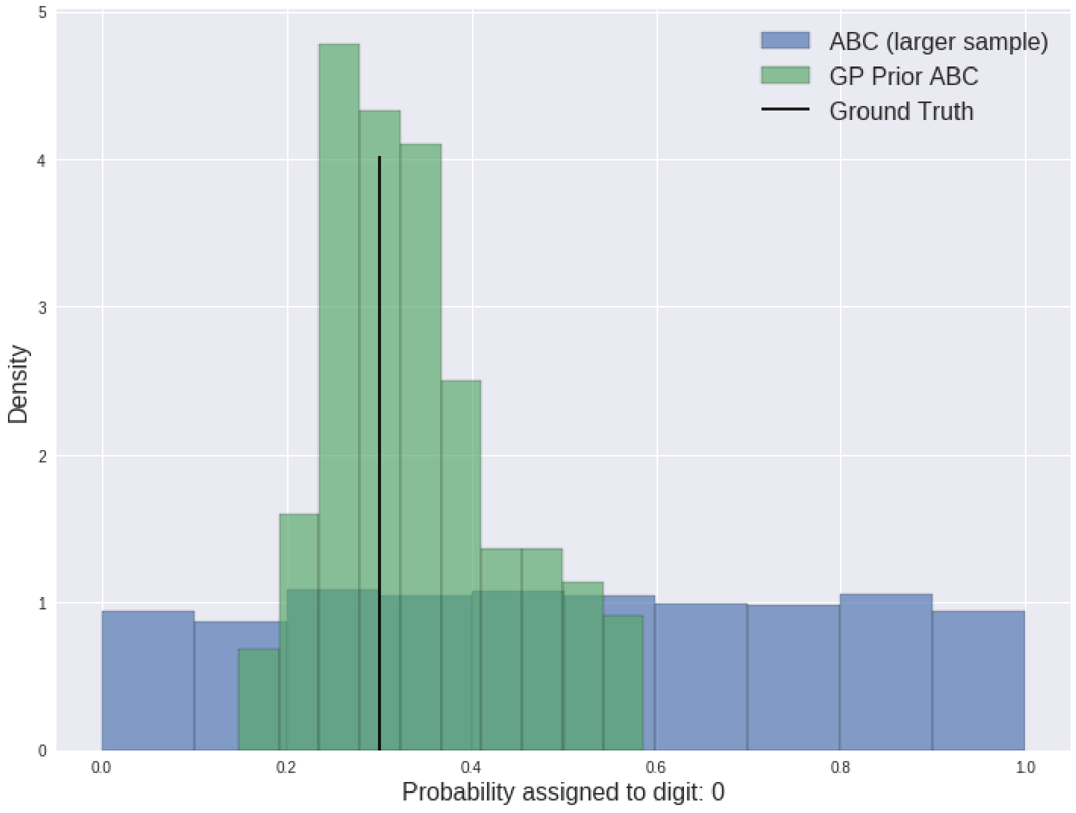


Figure 6:

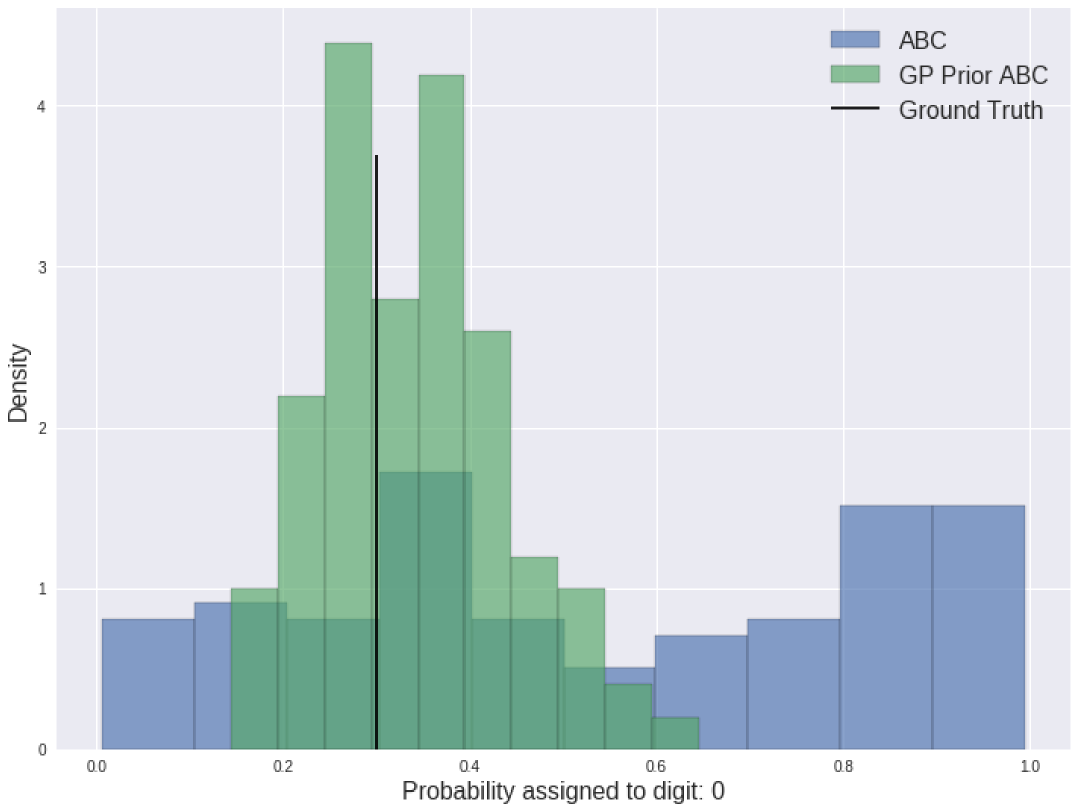


Figure 7:

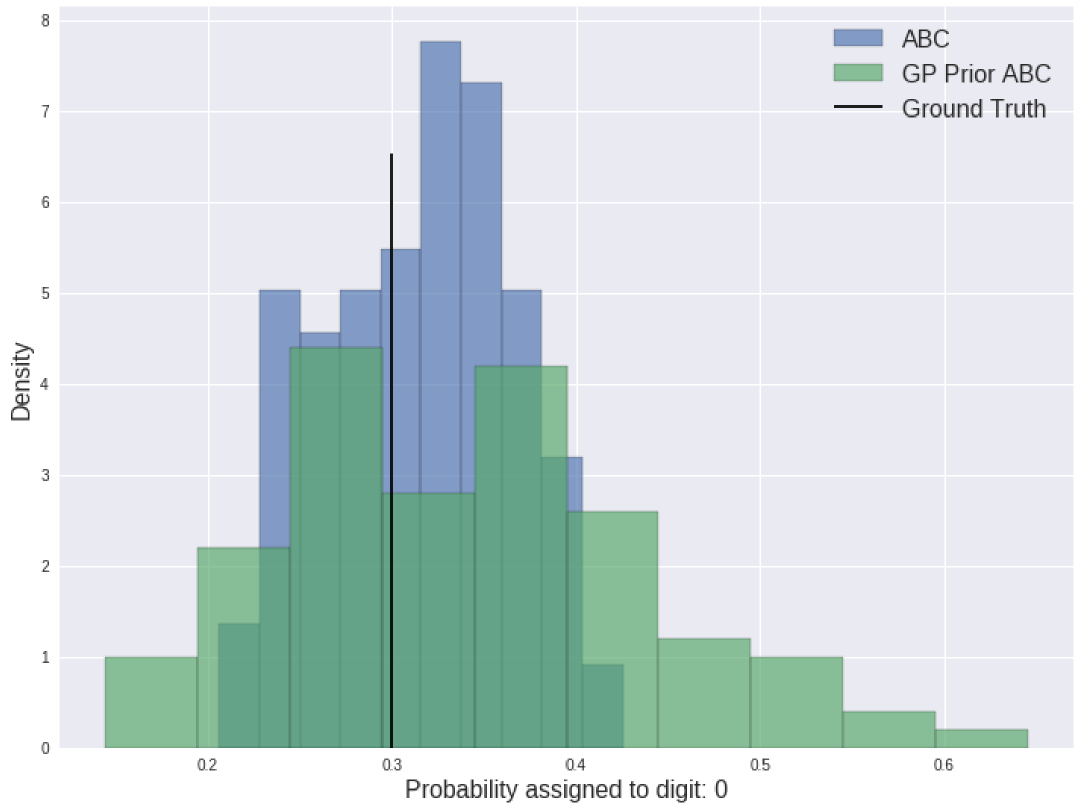

ABC
GP Prior ABC
Ground Truth
Density
Probability assigned to digit: 0
0
1
2
3
4
5
6
7
8
0.2
0.3
0.4
0.5
0.6

Figure 8:

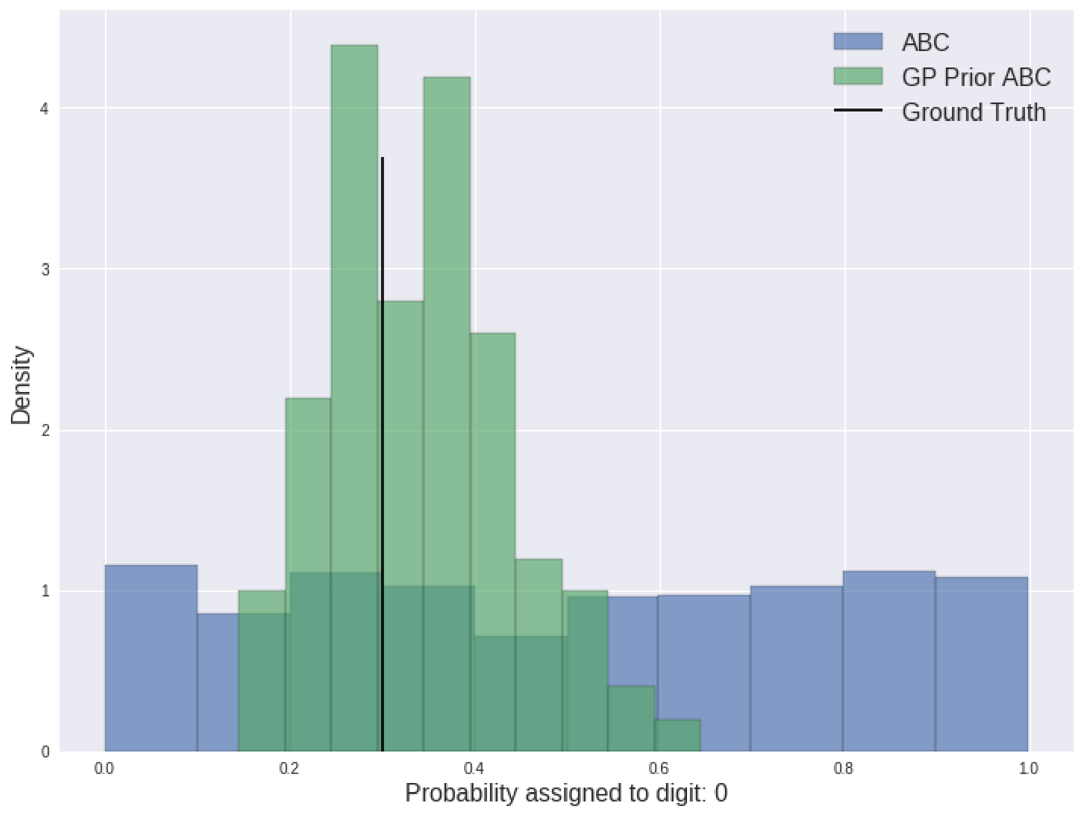


Figure 9:

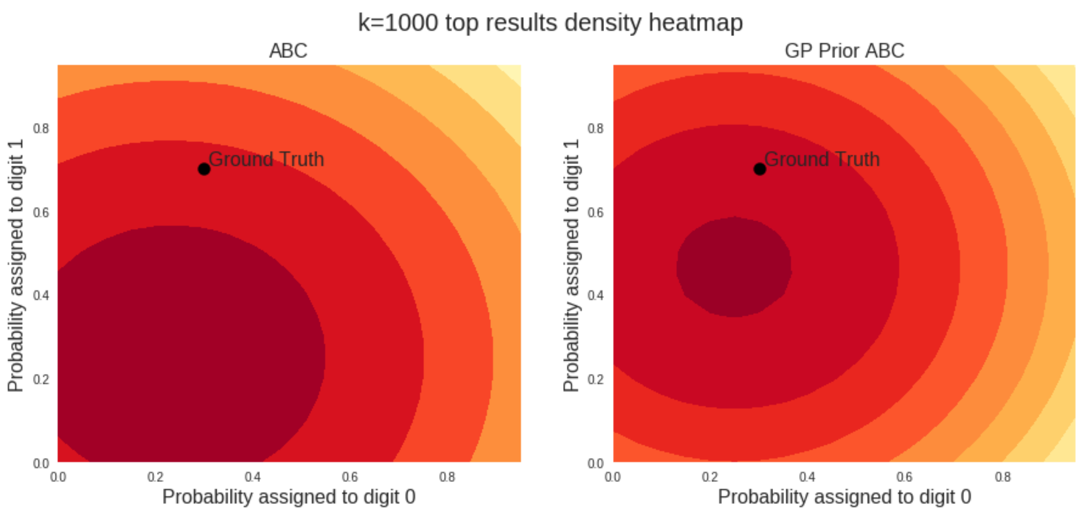


Figure 10:

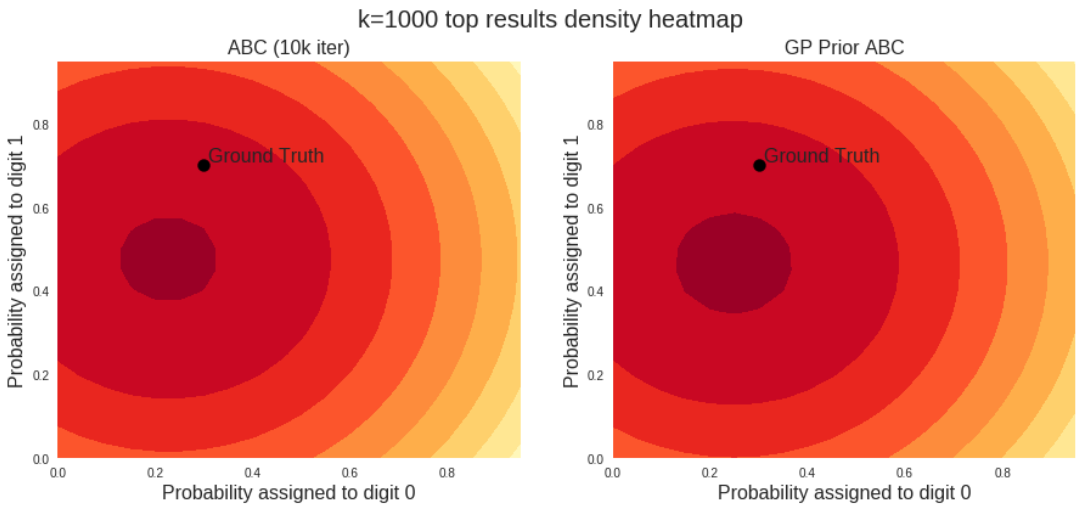


Figure 11:

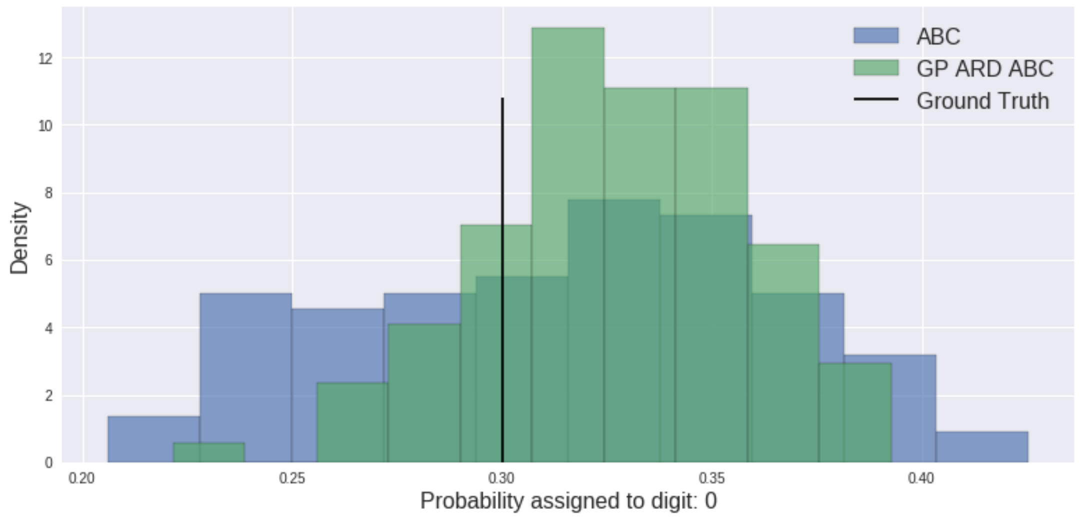


Figure 12:

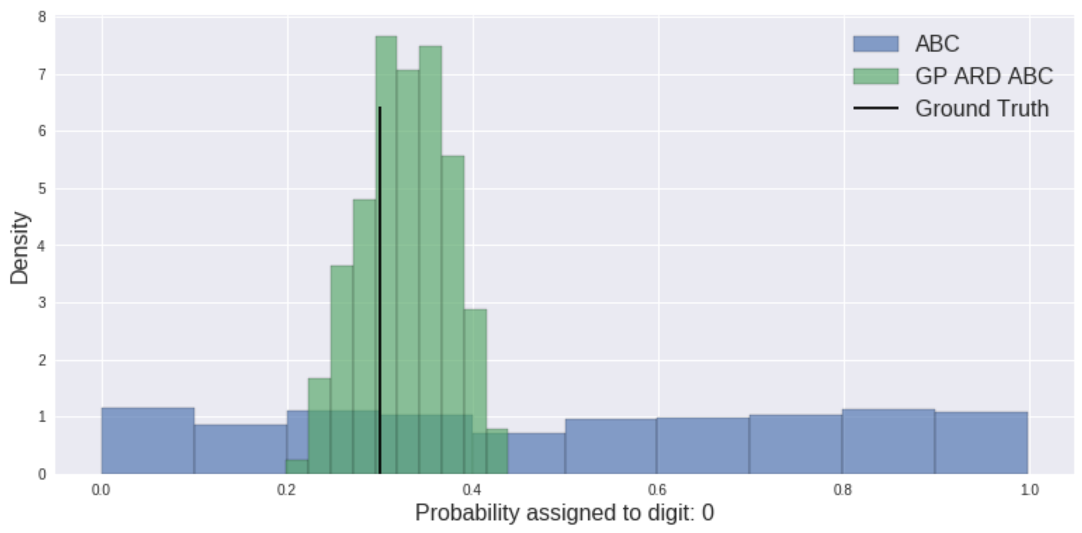


Figure 13:

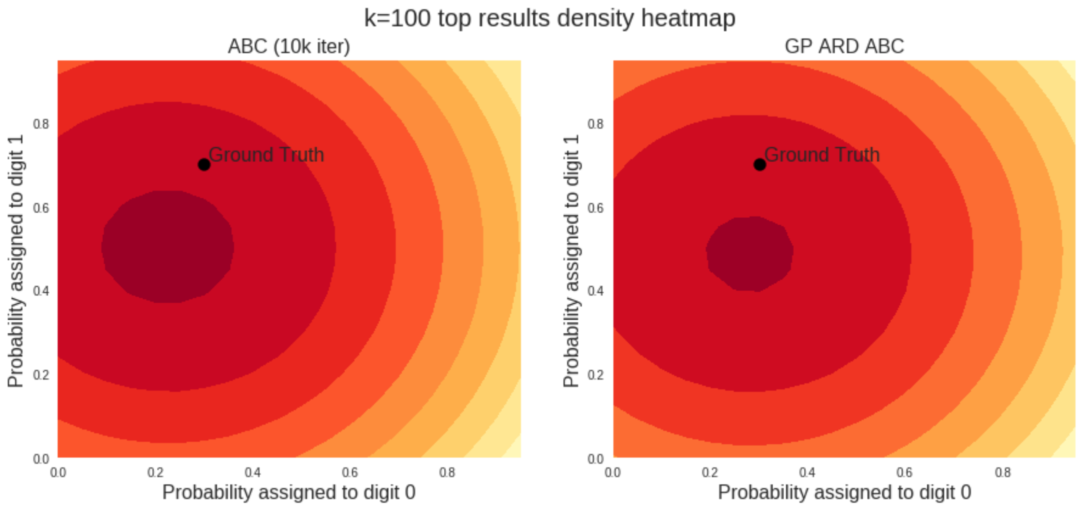


Figure 14:

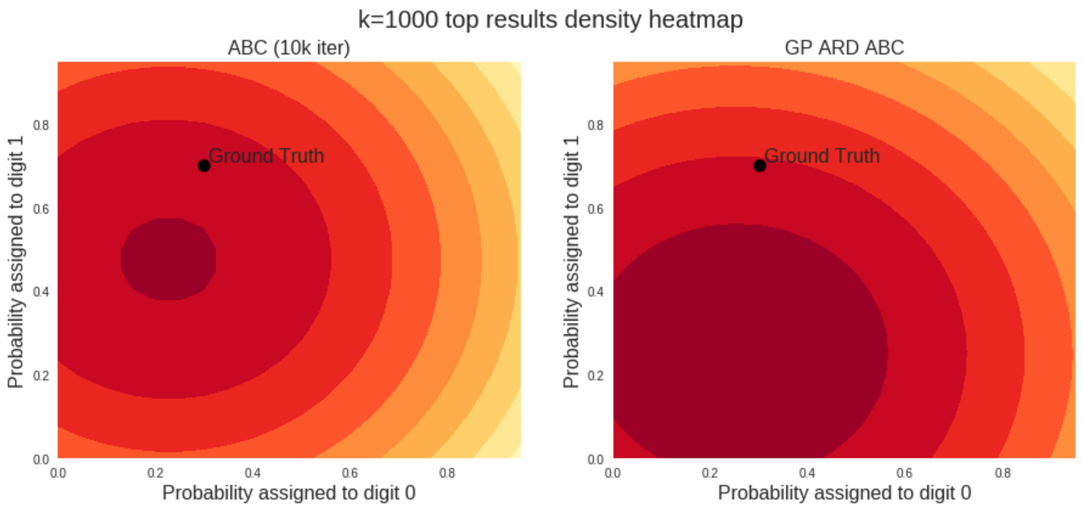


Figure 15:

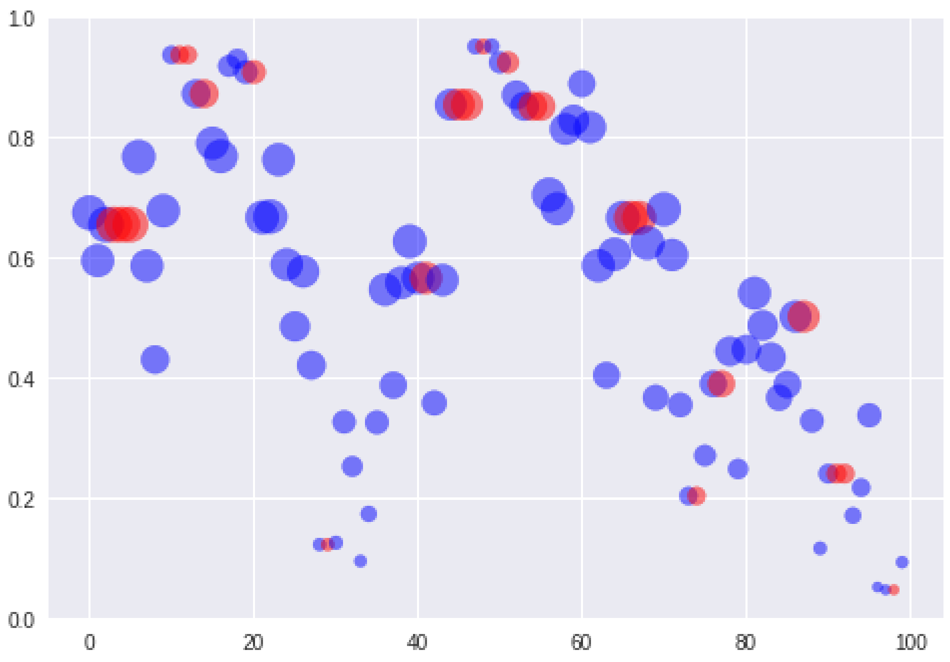


Figure 16:

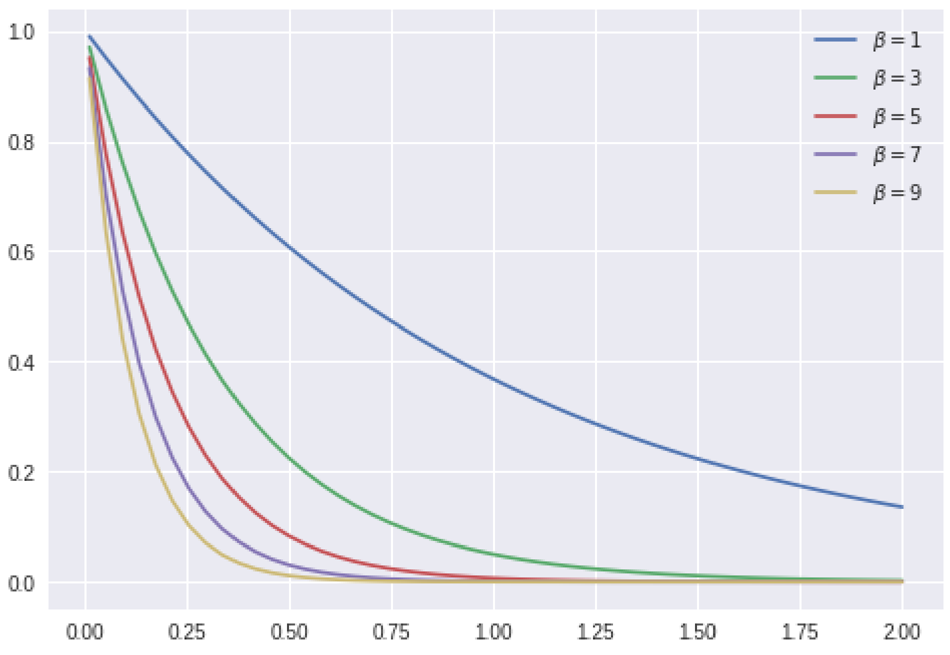


Figure 17:

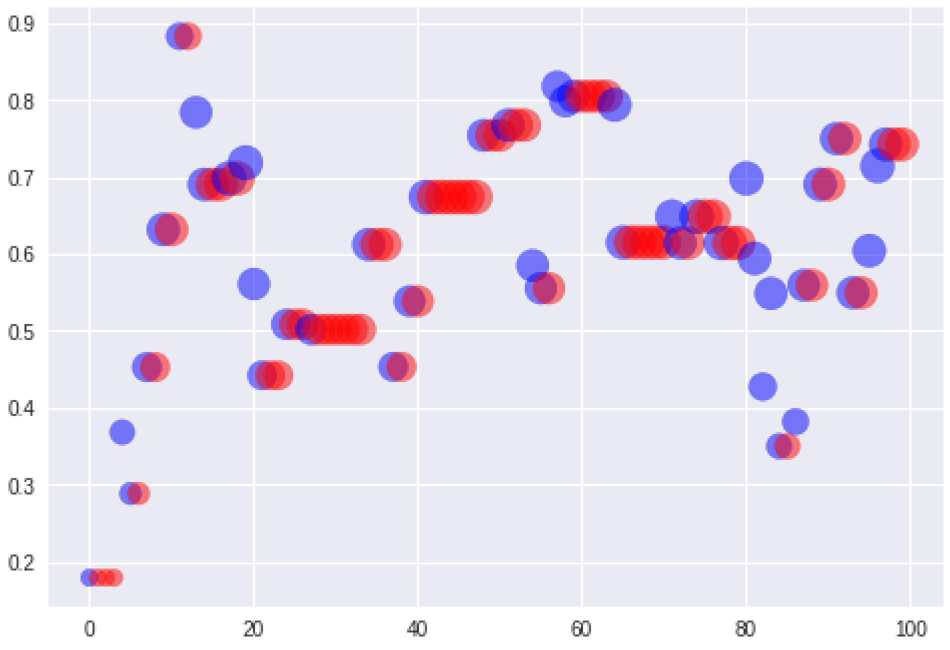


Figure 18: