Some Notes for the BIRS Workshop on Statistical Inference for High Energy Physics

Radford M. Neal
Department of Statistics and Department of Computer Science
University of Toronto, Toronto, Ontario, Canada
http://www.cs.utoronto.ca/∼radford/
radford@stat.utoronto.ca

25 June 2006

I've given some thought to the questions being addressed at the workshop, which overlap all the topic areas. I question whether the current way the problems are described is actually the most appropriate. I also report on some experiments I've done using Byron Roe's data.

The problem

First, let me summarize my understanding of the problem, and the approaches taken so far. Physicists wish to determine whether some hypothesized particle actually exists, based on descriptions (with perhaps dozens or hundreds of variables) of many recorded events (perhaps tens of thousands to millions, after some preliminary filtering?). From theory, they are able to write programs for randomly generating artificial events, both for the "background" in which the hypothesized particle does not appear, and for the "signal" in which the particle does appear. These programs are not perfect, however, and hence may generate slightly the wrong distribution of events. There may also be parameters (inputs) of these programs that are not completely known, whose values also affect these distributions.

The current approach is to use these programs to generate a fairly large number (tens of thousands or more) of background and signal events, and to then train a classifier to distinguish signal and background. This classifier is used to make hard decisions, by thresholding its output at some selected point. Using additional events produced by the background generation program, the fraction of background events that the classifier mistakenly identifies as signal events can be estimated. (This is apparently referred to as the "background efficiency", though "background error rate" would seem to me to be a better term.) Once real events are available, they are run though the classifier, with the result that some number, n, are identified as signal events. Since true signal events are presumably rare, we can compute the expected number of background events mistakenly classified as signal by just multiplying the total number of events by the background error rate to get an expected number of misclassified background events, b. The observed number, n, of events classified as signal is modeled as Poisson distributed with mean $b + s\epsilon$, where s is the expected number of true signal events and ϵ is the accuracy ("efficiency") of the classifier in identifying signal events (ie, one minus its error rate on signal events), which is estimated from events generated by the signal generation program.

The aim is to draw inferences about s. It is hoped that these inference will be "robust" to any flaws in the programs for generating background and signal events, including any small misspecification of the inputs for these programs.

Confidence intervals and the likelihood principle

Much effort has been put into devising methods for constructing confidence intervals for s. In my opinion, this is misguided. Confidence intervals are crude devices that are useful in practice only because the likelihood function is often approximately Gaussian in shape, and hence can be completely specified by two numbers, which could be the upper and lower limits of a 95% confidence interval. With experience, you can learn to interpret such things. If the likelihood function is not approximately Gaussian, it is unclear how a confidence interval should be interpreted. That a confidence interval has correct coverage properties is certainly not sufficient reason to believe it is useful, as is demonstrated by various well-known examples in the statistical literature.

Moreover, many methods for constructing confidence intervals violate the "likelihood principle" — that experimental data should affect your conclusions only via the likelihood function, not through other aspects of the experiment (such as the sampling properties of various statistics). For the Poisson$(b+s)$ model of n, for example, the likelihood function for s when $n=0$ is the same (apart from an irrelevant constant factor) for any value of b (which I take here to be known), so the same conclusion should be drawn when $n=0$ is observed regardless of what the value of b is. While there has been much debate about the likelihood principle and other foundational issues in statistics, in my opinion, there are persuasive argument in favour of the likelihood principle, and the only arguments against it that have any force are of a practical nature — using methods that obey the likelihood principle may sometimes not be worth the effort. For important scientific questions in which the model used is not especially complex, such practical issues do not seem to be a compelling consideration.

For the simple Poison$(b+s)$ example, with b known, the likelihood is a function only of s, and hence can easily be plotted. This plot of the likelihood is a full description of what was learned from the experiment. Moreover, any less detailed presentation is *not* a full description of what was learned. One-dimensional problems such as this are easy. It is when there are many nuisance parameters that practical problems with visualizing the likelihood function can become severe. Fortunately, it appears that for this problem the nuisance parameters (eg, b and ϵ when they are not known exactly) have widely-agreed-upon priors (based primarily on earlier experiments or simulations), so one can simply find the marginal likelihood for s integrating away the nuisance parameters. One can then again plot the (marginal) likelihood as a function of s, which presents the results of the experiment without the influence of any possibly controversial prior on s.

Similarly, I would not recommend use of p-values for this problem, as they also often violate the likelihood principle. There is a use for p-values in checking whether the model being used is valid — for this purpose, likelihood cannot be used, since it is defined only once one has chosen a model. This type of model checking doesn't seem to be an issue for the problems covered by this workshop, however.

Is this a statistical classification problem?

The scientific question of interest is what fraction of real events are true "signal" events, in which the new particle is present. If this fraction is f, the observed data distribution will be a mixture of a fraction f of the signal distribution and a fraction $(1-f)$ of the background distribution. Inference for the fraction f (or equivalently, for the expected number $s=Nf$ of signal events in N observed events) does not necessarily require that a classification model be trained. Learning a classifier is simply one way to reduce the dimensionality of the problem (in which an event is described by many variables) without losing information. Information *is* lost, however, if the output of the classifier is converted from a real value (eg, the probability that an event is a signal event) to a binary yes/no value, by thresholding the real-valued classifier output. As I demonstrate below, retaining more of the information in the real-valued classifier output allows for better conclusions about the scientific questions of interest (eg, whether $s = 0$). With more information retained, we can't use the simple Poisson($b + s\epsilon$) model that arises with hard thresholding. But as long as s is the only model parameter for which disagreement may exist regarding a suitable prior, we can still integrate over the nuisance parameters, and then present the results of the experiment in the form of a plot of the marginal likelihood as a function of s.

When we learn a classifier for this problem, we are in a sense not really doing statistics. The programs for generating background and signal events can generate arbitrarily many such events. There is no real statistical problem of inference from limited data. Instead, we have a computational problem of finding a classifier that is close to optimal given the signal and background distributions (which are in a sense known), without taking an excessive amount of computing time. Note that computation time is needed both to generate background and signal events and to create a classifier based on them. (It would help if we knew how much computation time is required to generate one background or signal event, so that an appropriate tradeoff could be found.) We also want the final classifier to be fairly fast at classifying real events, though my impression is that computational efficiency at this point isn't a very big issue.

In the experiments I report below, I used a Bayesian neural network classifier, which is designed for use with limited amounts of data. The training is rather slow when using the 60000 training events that I used. I think that the speed could be improved by focusing most of the computational effort on a minority of hard-to-classify events. This is reminiscent of "boosting", which other people have found to work well on this problem. I look at this matter a bit differently than is typical in the boosting literature, however. I think there is no statistical advantage to boosting; in this context, it is just a way of saving computation time.

Robustness

It appears that the programs for generating background and signal events may not be perfect. Perhaps they contain flaws of an unknown nature. These will naturally be hard to allow for, but the possibility of such flaws is perhaps a reason to look only at fairly coarse discretizations of the real-valued classifier output, whose distributions (for background and signal) may be less influenced by such flaws than the detailed distributions for the original real values.

It seems that these programs also have some input parameters that affect the distribution in a known way. The impression I have is that these inputs are currently fixed at the "best guess" for their correct values, but that people then explore whether the final classifier works well for data generated with somewhat different values for these inputs. This seems a bit unfair — you're asking that the classifier be robust to certain changes, but you don't tell it anything about what these changes might be. It would seem better to train the classifier on data generated using a variety of input parameter values. One could furthermore provide the classifier with the input parameter values used to generate each training event, so that it will learn to classify events with any specified setting of these parameters. When analysing real event data, one could then apply the classifier with various settings of these input parameters, and thereby discover how sensitive the results are to what parameter settings are assumed. One could also at this point determine which of these input parameter settings best fits the real data.

Experiments with Byron Roe's data

I've tried out some of these ideas using the data that Byron Roe supplied. This data consists of 130064 simulated events (36499 signal events, 93565 background events), each of which is described by 50 variables. I was unable to find any existing results using exactly this data set in the referenced papers, so I did my own division into training and test sets for the experiments I did. I randomly selected 60000 events for the training, with the remaining 70064 events used for testing. This division of the data is available via `http://www.cs.utoronto.ca/~radford/BIRS/index.html`. I'd be interested to know how well other classification methods do on this data set.

I used a multilayer perceptron neural network classifier with two hidden layers, with 25 units in the first hidden layer, and 10 units in the second hidden layer. There were connections from the inputs to the first hidden layer, from the first hidden layer to the second hidden layer, and from the second hidden layer to the single output unit, but no other "short cut" connections. The hidden units used tanh as the activation function. The value of the output unit was put through the inverse logit function ($1/(1+e^{-x})$) to produce a value between 0 and 1, which was interpreted as the probability that the event was a signal event.

I used Bayesian training, as described in my book, *Bayesian Learning for Neural Networks*, and implemented in my software package for Flexible Bayesian Modeling (available from my web page). I used an Automatic Relevance Determination prior for the weights on input-to-hidden connections, allowing the model to determine which of the 50 inputs have the most relevance. Training is done by Markov chain Monte Carlo simulation of the posterior distribution. I did three independent simulations to make sure that the results were reasonably correct. Sets of network parameters from the latter iterations of these simulations were used to make predictions for test cases (averaging the probabilities produced by these iterations). Training to a semblance of convergence took quite a long time with 60000 training events — about a week for each simulation run (I ran the three simulations I did in parallel).

If predictions for signal versus background on test cases are made by thresholding the final probabilities at 1/2, the overall error rate is 5.18%, with an error rate of 3.93% on background events and 8.39% on signal events. Figure 1 shows histograms for the logit of the predictive probability that an event is from the signal distribution, for test events from the signal and background distributions.

If these predictive probabilities are thresholded, to produce counts used in a Poisson model of the sort discussed above, it seems that the the optimal choice of threshold should be (at least approximately) the value that maximizes the ratio of the expected number of signal events that are correctly classified to the standard deviation of the number of background events that are incorrectly classified as signal events. The upper plot in Figure 2 shows this optimal threshold, which is approximately 0.98, as found using the test events. The lower plot shows the ROC curve found by varying the threshold and computing the corresponding error rates on test events from the signal and background distributions.

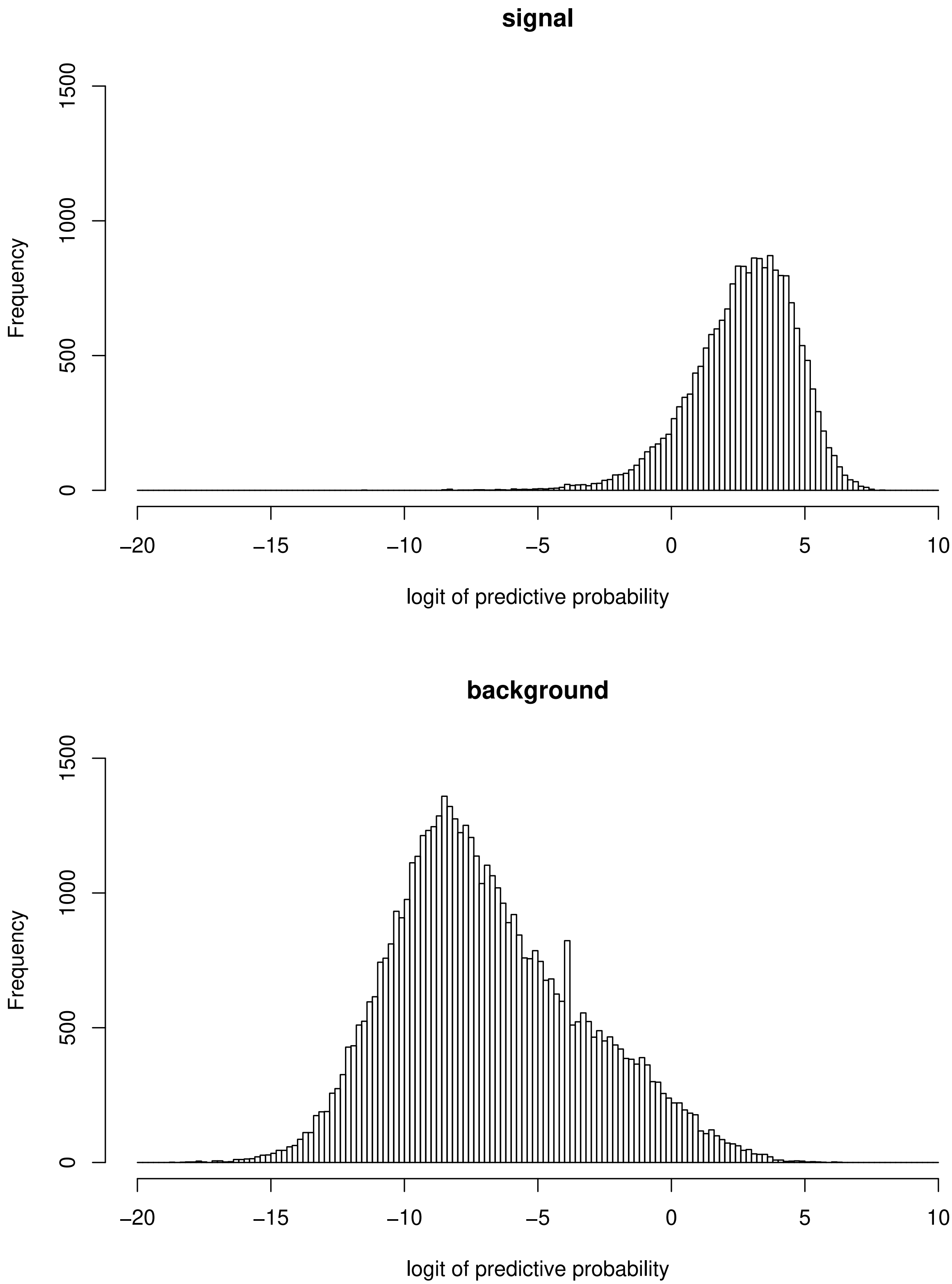


Figure 1: Histograms of the logit of the predictive probability produced by the Bayesian neural network classifier, for signal and background test events.

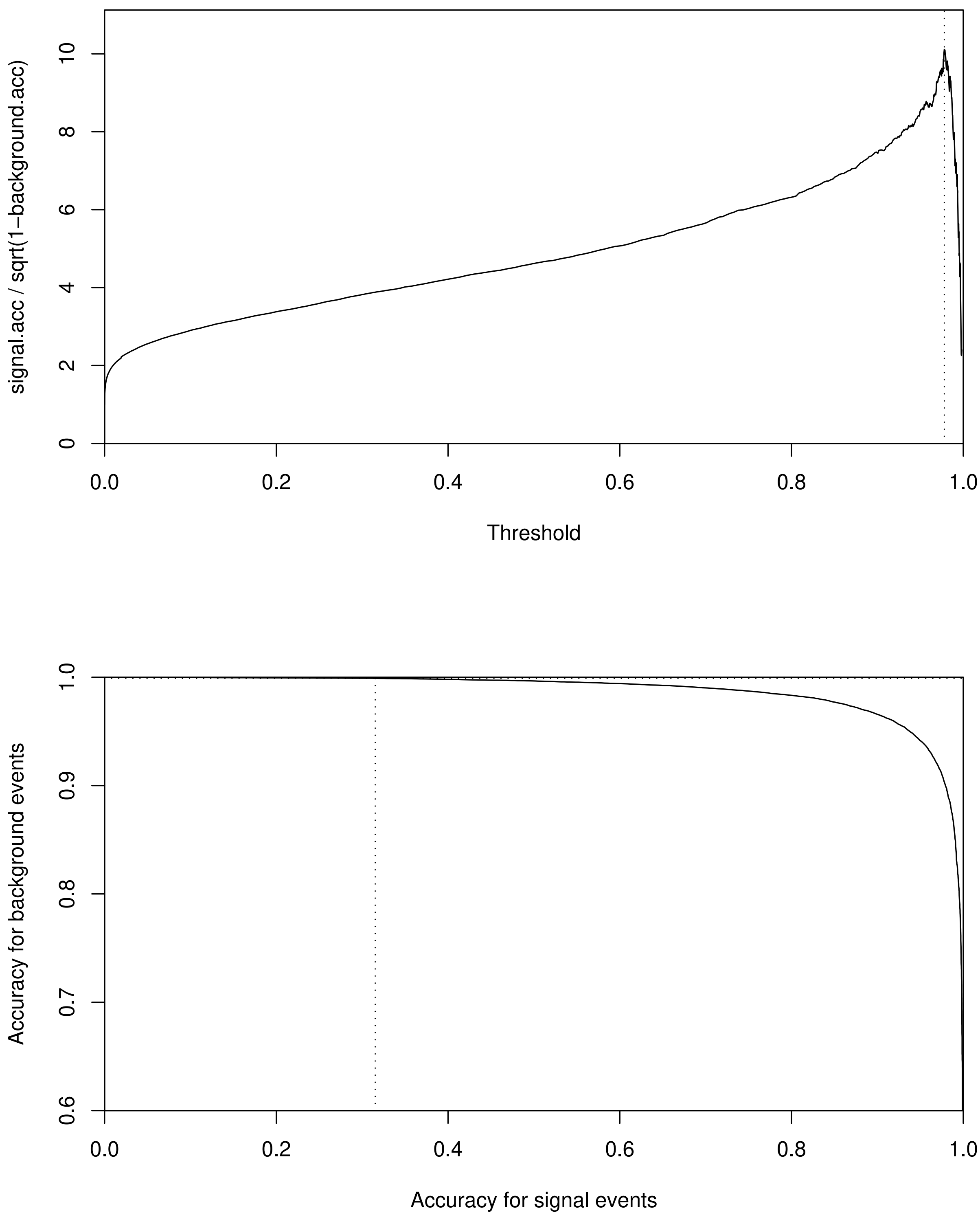


Figure 2: Plot identifying the optimal single threshold (top), and the ROC curve, with this optimal threshold marked (bottom).

I compared the results using this optimal single threshold with the results when the predictive probabilities from the Bayesian neural network are grouped into bins, using several thresholds. When using a single threshold, the likelihood function is of the Poisson form discussed above, based on the count of events exceeding the threshold. With several thresholds, the likelihood function is defined based on the count of how many events fall into each bin.

For this test, I used 10 bins, defined by the following thresholds:

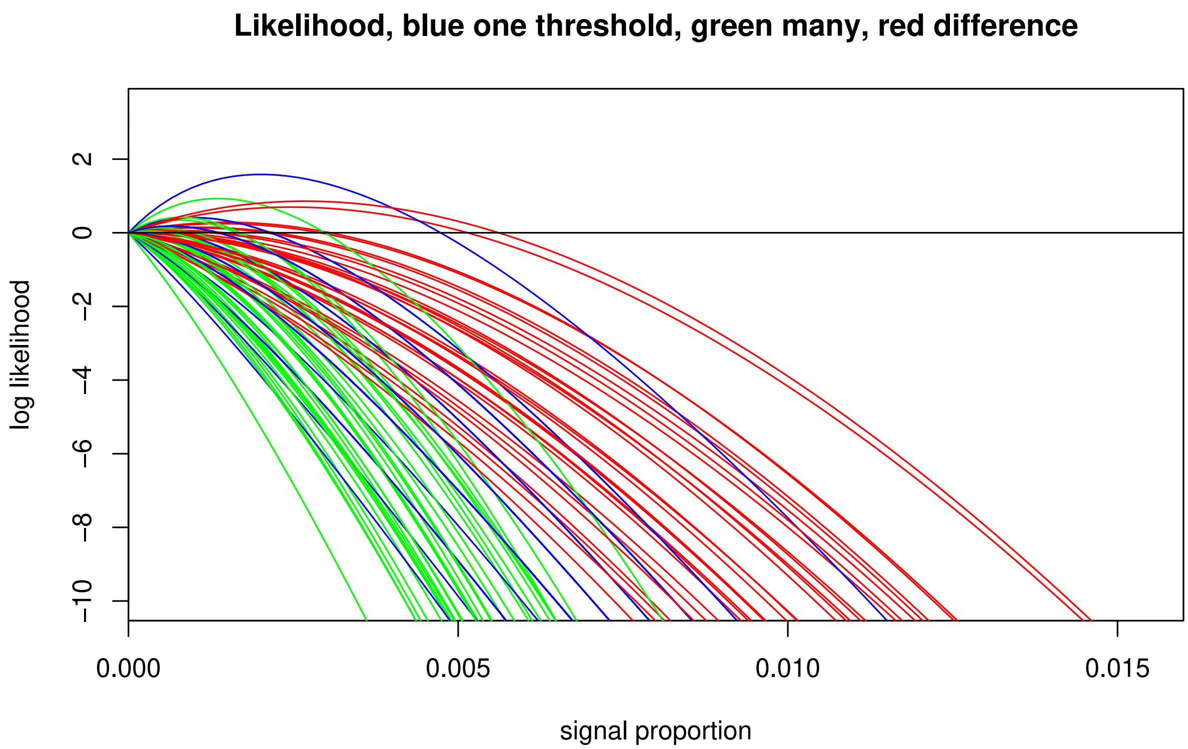
$$0.3, \quad 0.5, \quad 0.6, \quad 0.7, \quad 0.8, \quad 0.86, \quad 0.92, \quad 0.96, \quad 0.98$$

The number of bins and threshold values were chosen manually, not by any formal procedure. In theory, the more bins the better. In practice, using too many bins might lead to non-robustness (as discussed above), and might make estimating the probabilities for each bin more difficult. In this test, I estimated the probabilities for each bin under the signal and background distributions from the corresponding proportion of values in test events of each sort. Similarly, for the Poisson model, the expected number of events whose predictive probability exceeds the optimal single threshold under the signal and background distributions was estimated from these same test events.

To see how much better it is to use 10 bins rather than a single threshold (ie, two bins), I simulated 400 data sets from the event distribution when the true fraction of signal events is zero, and 400 from the event distribution when the true signal fraction is 0.004. Each simulated data set consisted of 10000 events, randomly picked (without replacement, within each simulation) from the sets of background and signal test events. For the simulations in which the signal fraction was 0.004, the number of signal events was randomly drawn from the binomial(10000,0.004) distribution. Note that the events in these simulated data sets were drawn from the same set of test cases as were used to estimate the probabilities of the bins, and to estimate the Poisson model parameters b and ϵ for the single threshold method. This mimics a situation in which these nuisance parameters are known exactly.

The results of these simulations are shown in Figure 3, where the true signal fraction is zero, and in Figure 4, where the true signal fraction is 0.004. In both figures, the top plot shows the log likelihood functions (shifted to be zero for signal fraction zero) that are obtained using both methods, for the first 30 simulated data sets. The single threshold method is shown in blue, and the method with 10 bins in green. The difference in log likelihoods for the two methods is shown in red. The bottom plots show the corresponding score functions (derivatives of the log likelihood functions). The score function plots are useful in verifying correctness, since the expected value of the score function at the true signal fraction should be zero. This seems so visually in these plots, and I confirmed this using the results for all 400 data sets.

The plots also provide visual evidence that the model with 10 bins produces better results than the single threshold model. When the true signal fraction is zero, the likelihood using the 10-bin model usually drops more sharply as the value of the signal fraction increases above zero. When the true signal fraction is 0.004, the likelihood at the true value of 0.004 when using the 10-bin model is usually higher than when using the single-threshold model. Quantitatively, the 10-bin model produces on average a log likelihood ratio for signal fraction of 0.004 versus signal fraction of zero that is greater by 1.70 (with 95% confidence interval $(1.52, 1.88)$) when the true signal fraction is 0.004. When the true signal fraction is zero, the difference is on average -1.79 (with 95% confidence interval $(-1.95, -1.64)$). Put another way, the odds in favour of the correct value (assumed to be



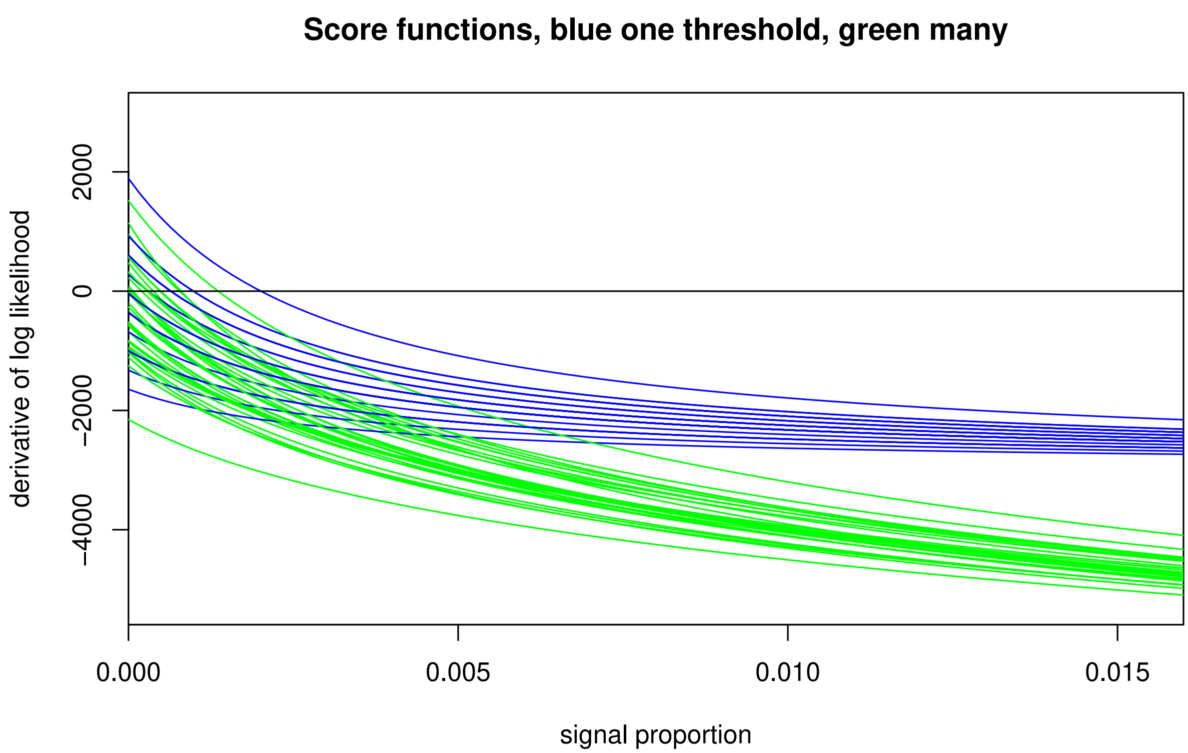


Figure 3: Results of the first 30 simulation runs when the true signal fraction is zero.

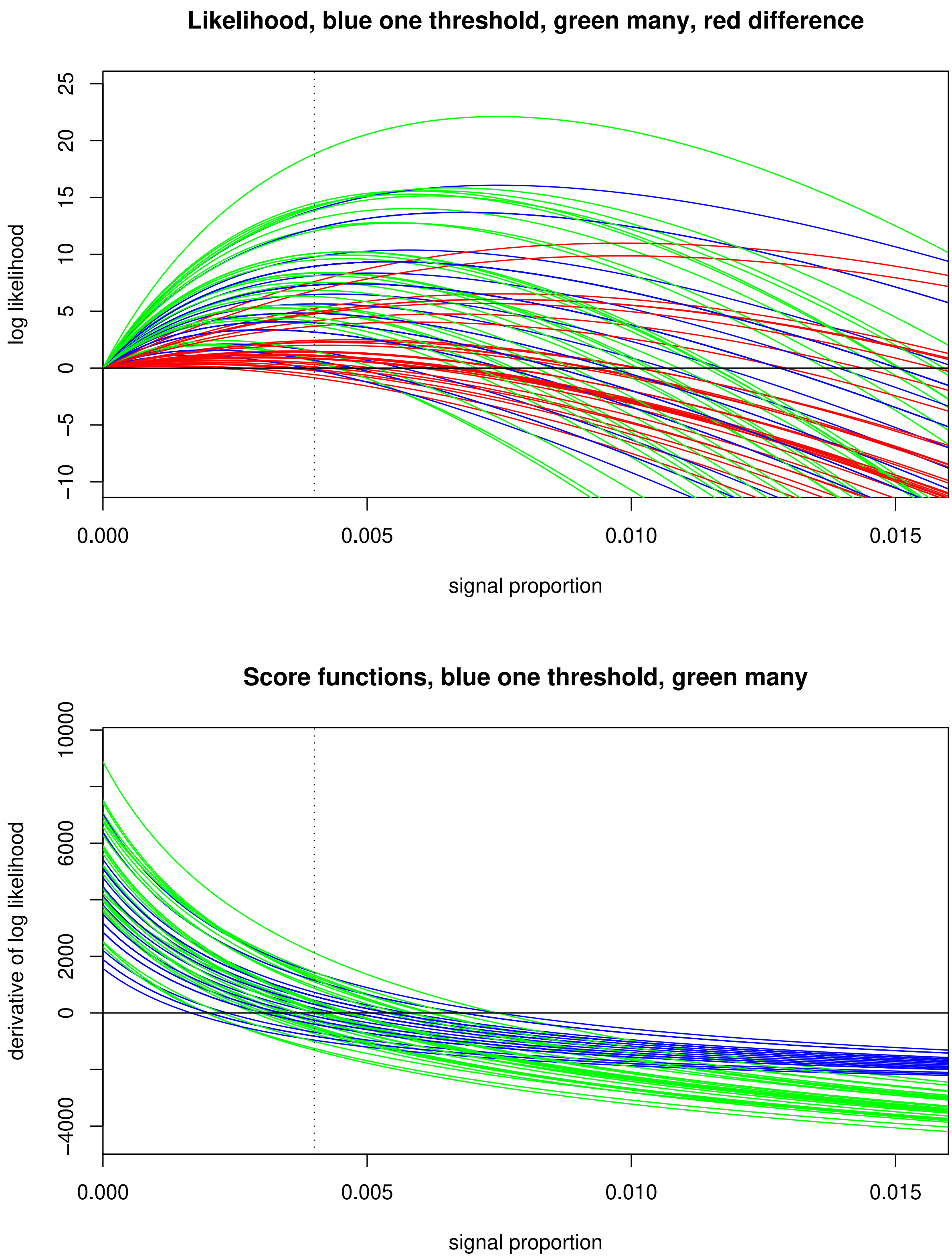


Figure 4: Results of the first 30 simulation runs when the true signal fraction is 0.004.

either 0 or 0.004) are a factor of about $\exp(1.7) \approx 5.5$ greater when using the 10-bin model than when using a Poisson model with counts obtained with a single threshold.

Figure 5 provides a more frequentist look at performance. Each point in this plot shows the likelihood ratio for signal fraction 0.004 versus signal fraction zero using the single threshold model (horizontal axis) and using the 10-bin model (vertical axis). The black points are for the 400 simulated data sets in which the true signal fraction was zero; the red points are for the 400 data sets in which the true signal fraction was 0.004.

The vertical line at -0.56 separates from the others the 23 largest likelihood ratios obtained using the single-threshold model applied to data sets where the true signal fraction is zero. (This point was chosen to avoid ties in the likelihood ratio.) A hypothesis test that rejects if the likelihood ratio is greater than this would have a Type I error rate of $23/400 = 0.0575$, and a Type II error rate if the true signal fraction is 0.004 of 0.0325 (since 13 of the 400 runs with true signal fraction of 0.004 produce likelihood ratios smaller than -0.56). The horizontal line marks the likelihood ratio obtained from the 10-bin method that when used to determine rejection in a hypothesis test would produce the same Type I error rate of 0.0575. The Type II error rate of this test would be 0.0125 (5 out of the 400 runs), which is a factor of 2.6 less than when using the single-threshold model.

Conclusions

These preliminary experiments show that preserving more information from classifier outputs by using many bins produces more informative results than using a single threshold, and thereby discarding the detailed information on how confident the classifier is its prediction. Several issues could be investigated further in this regard, including

1) Under what conditions can one formally show that the full classifier output preserves all relevant information?
2) Can one quantify how the amount of information discarded decreases as the number of bins increases?
3) Would some more sophisticated density modeling method be preferable to simply binning the predictive probabilities from the classifier?
4) How does the need for robustness and/or the limited number of simulated events available for setting nuisance parameters affect the best choice of the number of bins (or the parameters of some other density modeling method)?
5) Is the advantage of using several bins rather than a single threshold as great when the nuisance parameters are not completely known as was seen in the above experiments, where they are assumed to be known exactly?

I used a Bayesian neural network classifier for these experiments. This method was originally designed for use when training data is limited, but is here being used in a context where, apart from computational constraints, the amount of data available is unlimited. The computations for the Bayesian neural network model are dominated by simulations of Hamiltonian dynamics, used as a means of sampling from the posterior distribution of network parameters. These dynamic simulations currently take advantage of redundancy in the training set by performing integration

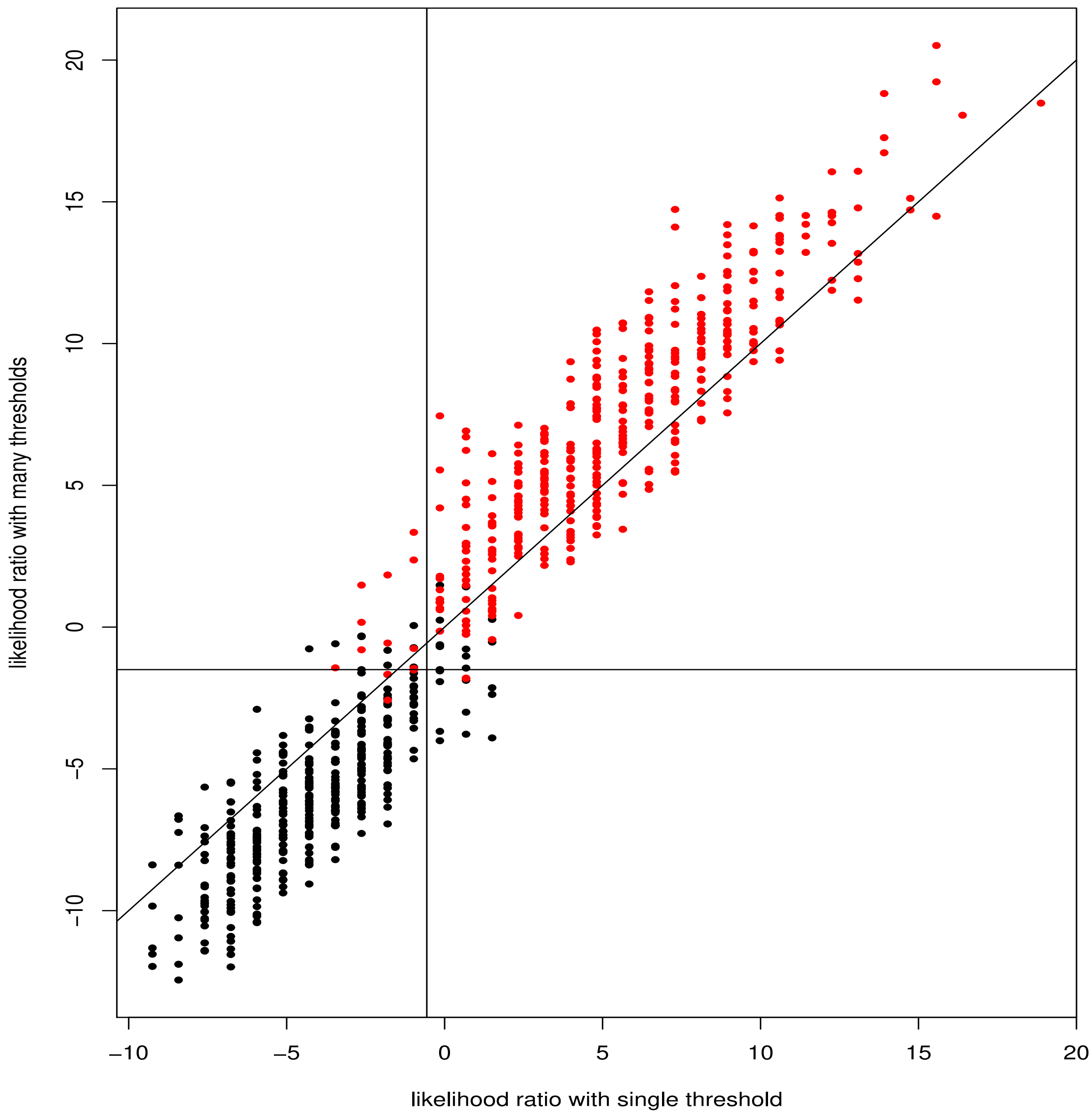


Figure 5: Comparison of methods on 400 simulation runs where the true signal fraction is zero (black) and 400 simulation runs where the true signal fraction of 0.004 (red).

steps on the basis of successive subsets of training cases. (Currently, I partition the data into 100 such subsets.) The computation time might be further reduced by using training cases (ie, simulated events) that are hard to classify more often than those that are easier to classify. To ensure that the correct answer is obtained in the end, the effect of each training case would be adjusted by the inverse of how often it is used. This focus on hard-to-classify cases seems somewhat similar to how boosting operates, but there are substantial differences, most fundamentally that I view this technique as a way of improving computational efficiency, not as a way of improving the statistical properties of the model.