

CS281B Spring 2004 – Project Report

Bagged decision trees vs. Bayesian neural network for sentence boundary detection

David Gelbart
gelbart@icsi.berkeley.edu

Task: Sentence boundary detection

The task is to detect sentence boundaries in conversational speech. This is done at the word level, using mainly prosodic features (based on patterns of pitch, duration, and stress). There are a few non-prosodic features such as an estimate of the speaker gender and whether there is a speaker change at that word. (Speaker change does not always signal a sentence boundary, since a second speaker may use backchannel utterances like "uh huh" while the main speaker continues a sentence.) The features and labels for this task were supplied to me by Yang Liu of the International Computer Science Institute.

Word identity and the identities of nearby words are relevant to the sentence boundary detection problem, but are not considered here. In fact, this task is a simplified version of another task, in which the word-level features are used to estimate a probability of boundary at each word, and these word-level probabilities are then combined with information about word identities using an HMM or other methods before final decisions about sentence boundaries are made.

There are 101 features. Two of them are categorical features taking on 8 possible values. When using the Bayesian neural net these were encoded as 8 binary features each (this encoding was suggested by Radford Neal as an appropriate way to encode unordered categorical variables for use with a Bayesian neural net), for a total of 115 features, and mean and variance normalization based on training data statistics was applied to all the features. I did not ask Neal to explain this suggestion, but it makes sense from the perspective that hidden units in a multi-layer perceptron act as "soft thresholds": by encoding each case separately we can have a separate threshold for each case. (Interestingly, even though there is redundancy in this encoding—7 binary features determine the value of the 8th—Neal commented by email that using all 8 is preferable as, "this treats them all symmetrically. Encoding them with n-1 inputs, with either one 1 or zero 1s, produces a prior on the effect of this variable that arbitrarily treats one of the values differently..."). The decision trees used the original 101 features, with mean and variance normalization based on training data statistics applied to the duration-, pitch-, and energy-based features.

The training set contains 480,560 points (each corresponding to one word). The test set contains 71,648 points. The features were automatically calculated, using word timing

information from forced alignment of word-level human transcriptions. (In a real application, this timing information would probably come from automatic speech recognition.) Binary labels (sentence boundary or not a sentence boundary) are available for all points. According to the labels, 84% of the test points are not at a sentence boundary.

Roughly 30% of the points in the training and test sets have one or more unknown features. In the training set, 111 out of 115 features are unknown for at least one point; in the test set it is 107 out of 115. Yang Liu suggested that there may be two types of unknown features, those that are "missing" (a meaningful value for the feature is possible, but was not provided) and those that are "undefined" (no meaningful value is possible). A pitch-related feature not provided due to failure of the pitch tracker is an example of the former. A feature related to the duration of the pause between a word and the previous word, not provided at the first word because there is no previous word, is an example of the latter.

Bagged decision trees

Yang Liu is currently using bagged decision trees for this task. The trees are CART trees which are implemented using NASA's IND toolkit. The training set is a reduced version of the full training set, with examples eliminated so that there are an equal numbers of examples for each class. (I did not do any such sub-sampling of training data to get class balance when training the BNN.) An ensemble of fifty trees is trained, with diversity created through bagging. Since the test set is not balanced in this way, the class probabilities estimated on test points by the ensemble are adjusted using the class priors.

The toolkit has several options for dealing with unknown feature values. Liu uses the default option which (quoting from the IND documentation) is to send the point "down each branch with the proportion found in the training set at that node. In effect, IND splits the example into fractional examples, with the larger piece going down the branch most of the data follows."

Bayesian neural network (BNN)

The BNN is interesting to me for two reasons. First, Radford Neal and Jianguo Zhang won the "feature selection challenge" pattern classification competition held at NIPS 2003 using BNN-based techniques, and they won by a surprisingly large margin. (However, the NIPS 2003 classification tasks involved much lower amounts of training data, so the NIP 2003 results were not strong evidence that the BNN would excel on the task I consider in this report.) Second, this approach allows estimation of the relative importance of different features to the classification decisions, using automatic relevance determination (ARD). The feature relevance information provided by ARD could be useful feedback for feature design, and also can be used for feature selection in order to pick a good reduced-size subset of the features. The design of the ARD is also intellectually appealing, as discussed in the next section.

Automatic relevance determination (ARD)

ARD for the BNN is based on adding an additional hyperparameter for each input unit, controlling the standard deviations of the weights on connections out of that input unit. Here is the explanation of the automatic relevance determination (ARD) from Neal's book:

... each input variable has associated with it a hyperparameter that controls the magnitudes of the weights on connections out of that input unit. These hyperparameters are given some prior distribution, and conditional on the values of these hyperparameters, the weights out of each input have independent Gaussian prior distributions with standard deviation given by the corresponding hyperparameter. If the hyperparameter associated with an input specifies a small standard deviation for weights out of that input, these weights will likely be all small, and the input will have little effect on the output; if the hyperparameter specifies a large standard deviation, the effect of the input will likely be significant. The posterior distributions of these hyperparameters will reflect which of these situations is more probable, in light of the training data.

Thus the state of these hyperparameters after training can be used to judge the relevance of features.

With ARD for the BNN, feature selection is an organic part of the classifier. This appeals to me more than filter methods for feature selection, since the optimal feature subset for a particular task can depend on the classifier. Also, unlike many wrapper approaches for feature selection, ARD does not have computational costs exceeding normal classifier training times; even if there is no interest in reducing the size of the feature set, ARD is a useful technique which can improve classifier performance by helping the net ignore counter-productive features.

Handling unknown features with the BNN

To use the BNN I needed a way to deal with the unknown features. I used three methods.

Global means

I first tried setting an unknown feature to the global mean of that feature, calculated over the points in the training data for which that feature had that value.

Flag features

For each feature which could become unknown, I added an associated binary feature which indicated whether or not that it was unknown for the current data point. The unknown feature values were filled in with the global feature means as before.

Conditional means

I also tried using the conditional mean of the unknown features given the values of the known features, instead of using the global means. For this purpose, I estimated the mean vector and covariance matrix of the 115 features over the 70% of training points for which all features are defined. Then, by modeling the joint distribution of the 115 features as a multivariate Gaussian, I could estimate the value of unknown features for a data point given the values of the known features using the following fact which I have cut-and-pasted right out of some class notes by Marc Serre of UNC that I found with Google:

Let \mathbf{x} be partitioned so that $\mathbf{x} = \begin{bmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{bmatrix}$. If \mathbf{x} is multivariate Gaussian with mean $\mathbf{m} = \begin{bmatrix} \mathbf{m}_a \\ \mathbf{m}_b \end{bmatrix}$ and covariance matrix $\mathbf{C} = \begin{bmatrix} \mathbf{C}_{aa} & \mathbf{C}_{ab} \\ \mathbf{C}_{ba} & \mathbf{C}_{bb} \end{bmatrix}$, then the conditional distribution of \mathbf{x}_a given that $\mathbf{x}_b = \boldsymbol{\chi}_b$, is also multivariate Gaussian, with mean $\mathbf{m}_{a|b} = \mathbf{m}_a + \mathbf{C}_{ab} \mathbf{C}_{bb}^{-1} (\boldsymbol{\chi}_b - \mathbf{m}_b)$.

(I was puzzled at first about how I could use this formula if the known and unknown features did not occur in two contiguous blocks; then Kofi Boakye pointed it out to me that I can simply re-arrange the order of the features as needed.)

This requires matrix inversion and matrix multiplication at each training or test point containing one or more unknown values. My current implementation uses linear algebra routines implemented in an interpreted scripting language (Perl) and handles less than a point per second! Caching $\mathbf{C}_{ab} \mathbf{C}_{bb}^{-1}$, which I have not tried, might greatly improve performance. For this caching to be practical, I think there needs to be structure to the patterns of what features are known and unknown, since if these patterns are unconstrained then there is a combinatorial explosion in the number of values of $\mathbf{C}_{ab} \mathbf{C}_{bb}^{-1}$ to be cached. I think there is indeed a lot of structure, for example, pitch-related features might become unknown as a group if pitch detection fails.

Speed

According to Yang Liu, using 8 or so CPUs in parallel, 50 bagged decision trees can be trained using the sub-sampled version of the full training set within “a day” (I don’t know whether this means a workday or a 24-hour period). The test points can be classified in about 15 minutes using one CPU.

The BNN training was not parallelized. BNN training times were large enough that I never used the entire training set to train the BNN. When I have recorded it, I give BNN training times (on 2.8 GHz Pentium 4 CPUs) next to test accuracy results. There is measurement noise in the training times due to the lack of control of other jobs that may have been using a computer’s other CPU, using the other “virtual CPU” provided by the

same CPU via hyper-threading, or using the same file server. The training times that I give are actually total times to run the experiment, but classifying the test set took only minutes (I timed it at about 5 minutes once, but there is probably some variation according to the net topology).

There is a “dynamic” mode of BNN training which is faster than the “hybrid” mode that I used, but I did not experiment with this.

Results: Decision Trees

A single decision tree: about 90.6% test set accuracy.

50 decision trees (bagging): 91.2% test set accuracy.

Results: BNN

I present the BNN results in several following sections, divided up according to the way unknown feature values were treated (global means, flag features, conditional means) and whether ARD was used.

When I was not using ARD, my training configuration was nearly identical to the example of a binary-classifying net included in the BNN toolkit (“Ex-netgp-b”). My only changes were to specify new train/test sets, change the number of input units and, sometimes, change the number of hidden units. When I was using ARD, I changed the configuration to add the extra hyperparameters (as in the toolkit’s “Ex-netgp-c” example). (At the same time, I made a minor change to some values for priors, but this did not seem to have much effect on accuracy.)

For net training, Markov chain Monte Carlo (MCMC) was run for 201 iterations, with test set classification performed based on the last 150 iterations.

An example of the format I present BNN results in is “16x15x1; train first 90,000; 34 hours: 91.98%”. 16x15x1 is the net topology, given as (number of input units)x(number of hidden units)x(number of output units). The number of training points used is given next; “train first 30,000” means that the first 30,000 points in the training set were used. If I have recorded the training time, it is given next (34 hours). Finally, the test set accuracy (91.98%).

Results: BNN; global means

15 hidden units

115x15x1; train first 9,000: 90.82%

30 hidden units

115x30x1; train first 9,000: 90.69%

115x30x1; train first 30,000: 91.49%

45 hidden units

115x45x1; train first 30,000: 91.46%

Discussion

There is a clear win from increasing the amount of training points. Increasing the number of hidden units results in a slight drop in accuracy in two cases (slight enough that I am not at all confident that the drop would generalize to other test sets). Regarding the number of hidden units for a Bayesian neural net, Radford Neal wrote, by email: “With Bayesian methods, the normal approach would be to just use as many hidden units as you can tolerate from a computational standpoint. With more hidden units, each iteration takes longer, of course, plus it may take more iterations for it to converge. Assuming you run things for as long as is needed for convergence, there is NOT any reason to expect that using a small number of hidden units would systematically result in better performance than a large number of hidden units (though that's possible for some particular problems and training set sizes).” I tried using 200 extra iterations for the “115x30x1; train first 9,000” case, but accuracy only changed by 0.01%.

Results: BNN; ARD; global means

15 hidden units

115x15x1; train first 9,000: 91.24%

30 hidden units

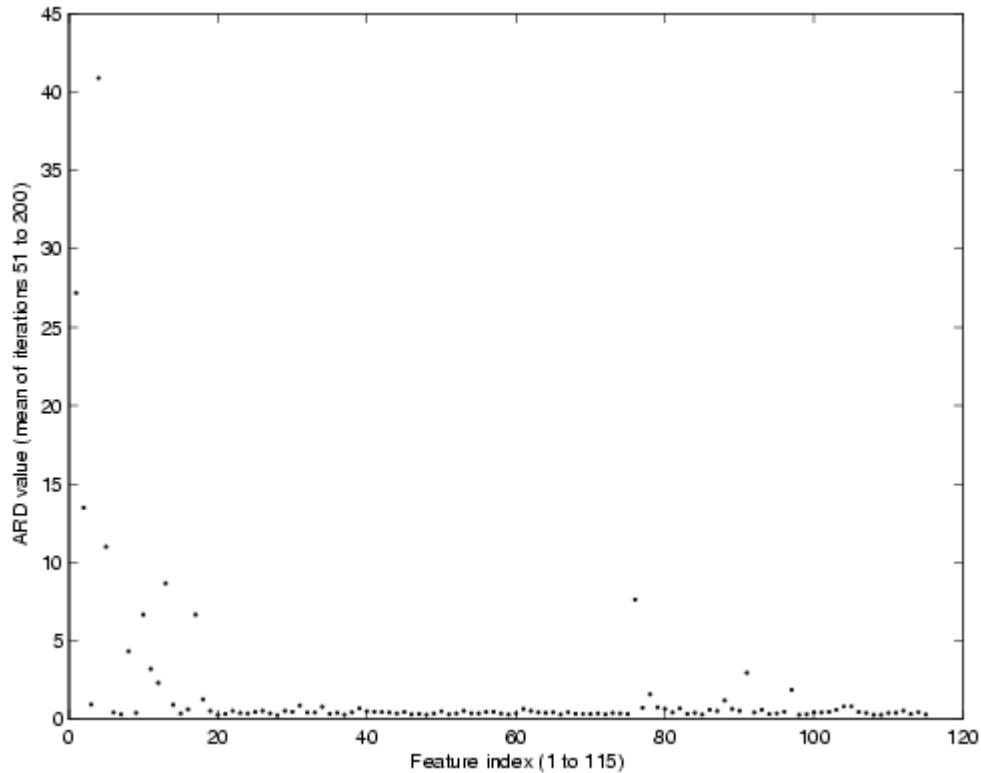
115x30x1; train first 9,000: 91.27%

Discussion

The use of ARD increased the performance of the 115x15x1 net from 90.82% to 91.24%, and the 115x30x1 net from 90.69% to 91.27%.

Feature selection

The plot below shows the value of the ARD hyperparameters (averaged over the last 150 training iterations) for each input unit of the 115x15x1 net.



The 16 features with ARD hyperparameter mean values > 1 were used in some further experiments. The lower number of input features sped up training, making it easier to use larger training sets. ARD hyperparameters were part of the net configuration in these further experiments as well.

15 hidden units

16x15x1; train first 9,000: 91.36%

16x15x1; train first 30,000: 91.72%

16x15x1; train first 90,000; 33.7 hours: 91.98%

16x15x1; train first 200,000; 75.6 hours: 91.88%

Earlier I had asked Neal by email, “If I increase the number of training examples are there particular parameters that I should increase or decrease?” and he had replied, “You may need to reduce the stepsize in the "hybrid" operation in order to keep the rejection rate low.” So I tried reducing the stepsize-adjust parameter from 0.3 to 0.1 or 0.05 and re-running the 200,000 point training:

16x15x1; train first 200,000; stepsize-adjust=0.1; 67.3 hours: 91.96%

16x15x1; train first 200,000; stepsize-adjust=0.05; 70.3 hours: 92.03%

The effect of stepsize seems interesting and deserving of further investigation. I did not use these alternate stepsizes in any other experiments, unless explicitly noted.

30 hidden units

With 30 hidden units I still used the 16 features selected from the training of the 115x16x1 net on 9,000 points.

16x30x1; train first 30,000; 20.8 hours: 91.81%

16x30x1; train first 90,000; 66.2 hours: 91.97%

Discussion

Delightfully, the 16 feature performance matches (actually slightly improves) the 115 feature performance for the 115x15x1 net trained on the first 9,000 training points. Adding more training data up to 90,000 points improves performance further. Increasing either the amount of training data or the number of hidden units does not improve on the best result (91.98% for 115x15x1 net trained on 90,000 points).

Results: BNN; ARD; global means; flag features

15 hidden units

226x15x1; train first 9,000; 11.8 hours: 91.10%

226x15x1; train first 30,000; 49.5 hours: 91.62%

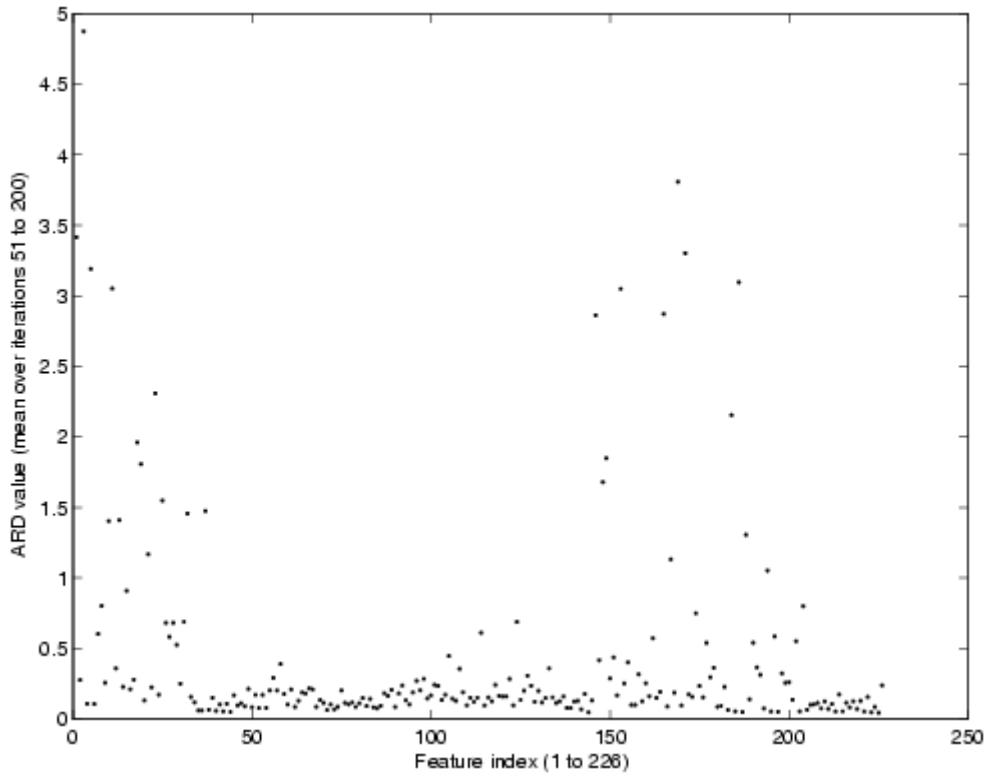
226x15x1; train first 90,000: 92.14%

Discussion

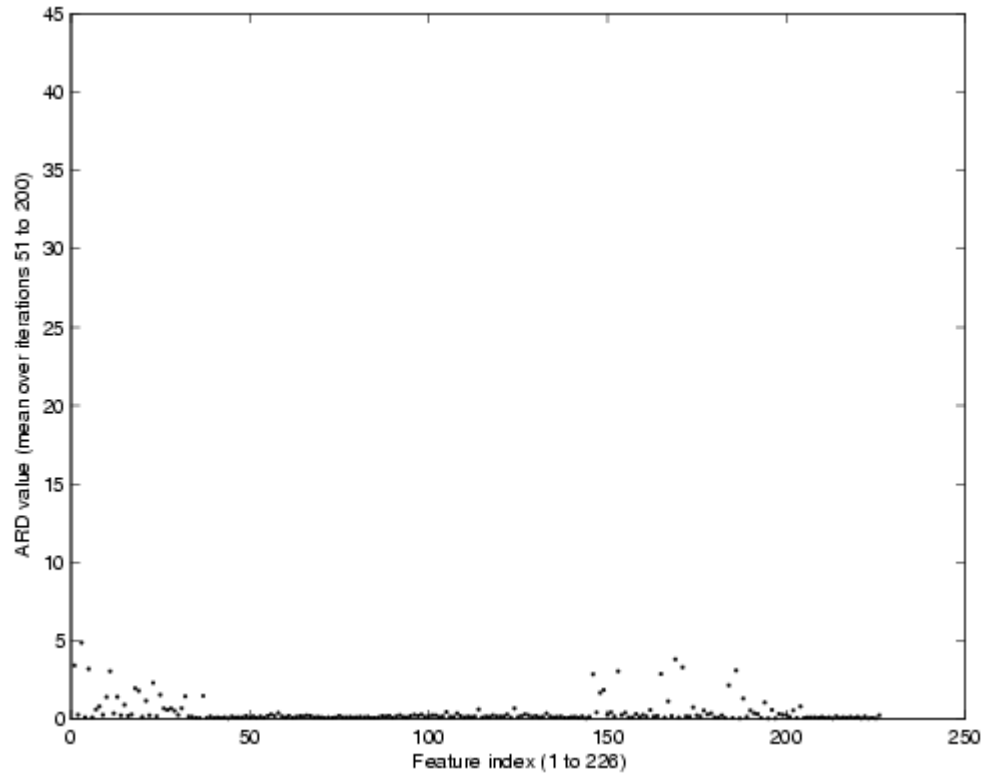
Compared to the 16x15x1 nets, the 226x15x1 nets with flag features perform 0.26% (absolute) worse for 9,000 training points, 0.10% worse for 30,000 training points, and 0.16% better for 90,000 training points. Perhaps the extra feature vector length and the resulting increase in the number of parameters makes training more difficult. This possibility, and a desire to speed up training to make it easier to use more training points, led me to try feature selection with ARD.

Feature selection

The plot below shows the value of the ARD hyperparameters (averaged over the last 150 training iterations) for each input unit of the 226x15x1 net trained on 90,000 training points.



The plot below is the same as the one above, except that the y axis has been rescaled for easier comparison with the ARD hyperparameter plot for a 115x15x1 net that was presented earlier in this report.



32 features (all features with hyperparameter mean values > 1)

32x15x1; train first 90,000; 39.6 hours: 91.70%

32x15x1; train first 200,000; 77.0 hours: 91.83%

45 features (all features with hyperparameter mean values > 0.5)

45x15x1; train first 90,000; 50.8 hours; 91.77%

92 features (all features with hyperparameter mean value > 0.25)

92x15x1; train first 90,000; 75.9 hours; 91.80%

Discussion

In these results, feature selection always seems to cause a loss in accuracy. Perhaps there is some number of features greater than 92 but less than 226 that would allow eliminating some features (thus speeding up training time) without losing accuracy.

Results: BNN; ARD; conditional means

15 hidden units

115x15x1; train first 9,000; 8.6 hours: 91.26%

Discussion

This is nearly identical to the result using global means (91.24%). This surprises me, since I expected a clear improvement. Perhaps there was a bug in my implementation of the conditional mean calculation, which was not completely tested.

Conclusions

As a comparison of the effectiveness of decision trees and Bayesian neural nets for this task, this report is weakened by the lack of consistency between the two classifiers regarding the training points used, the encoding of features (101 vs. 115), and the handling of unknown feature values. The real mentality behind this report was an engineering-minded attempt to increase the test set accuracy as much as possible. As such, it has been somewhat successful, since 92.1% accuracy was achieved with the highest-performing BNN configuration, improving over the bagged decision tree accuracy of 91.2%. However, this task is only a simplified version of the true task of interest, and it remains to be seen whether this improvement carries over to the true task.

There was some experimentation with how unknown features are handled, but further exploration may be justified considering the large number of unknowns in the data. With the Bayesian neural net, the conditional mean method provided no improvement over using the global mean, although the presence of a bug in the conditional mean implementation has not been completely ruled out. The use of flag features to signal when other features has given the best result (by a slight margin) and it could be worthwhile to try that approach with an increased number of training points or with a tuned stepsize. The paper “Modeling NERFs for Speaker Recognition” by Kajarekar, Ferrer, Sönmez, Zheng, Shriberg, and Stolcke (unpublished as of this writing) presents some other approaches for the dealing with the issue of unknown prosodic features.

There are many options for the BNN architecture and BNN training, such as the training stepsize (which may need to be reduced if the number of training examples is increased), the number of MCMC iterations, the values for priors, whether there are direct input-output connections, and whether there is more than one hidden layer (and whether there are direct connections to the inputs/outputs for layers which are not the first/last). I have not performed a thorough investigation of the effect of BNN options on performance. I do not separate “development” and “evaluation” test data which makes my methodology somewhat flawed for investigating option tuning in any case.

References

Decision Trees

The decision tree experiments were carried out (by Yang Liu) using the IND decision tree software, which is available from <http://opensource.arc.nasa.gov>

BNN

The BNN experiments were carried out using Radford Neal's "Bayesian Modeling and Markov Chain Sampling" software (2003-06-29 release), which is available from <http://www.cs.toronto.edu/~radford>

Neal's book *Bayesian Learning for Neural Networks* (Lecture Notes in Statistics No. 118, Springer-Verlag New York, 1996) describes the BNN approach including ARD. There are also papers on this at David MacKay's "Bayesian methods for neural networks" Web page at <http://www.inference.phy.cam.ac.uk/mackay/BayesNets.html>. There is a chapter on BNN in Christopher Bishop's book *Neural Networks for Pattern Recognition* (Oxford University Press, 1996).

BNN at NIPS 2003 challenge

The NIPS 2003 feature selection challenge is described at <http://clopinet.com/isabelle/Projects/NIPS2003> and in the upcoming book *Feature extraction, foundations and applications*, editors Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti Zadeh (publication by Springer-Verlag expected during 2004). The winning Neal and Zhang entries, which were based on BNN, are described in <ftp://ftp.cs.utoronto.ca/pub/radford/feat-sel-slides.pdf> and in a chapter in the upcoming book.

Postscript – July 2004

Yang Liu ran tests for the full task: using decision trees or Bayesian neural nets to estimate frame-level probabilities which are then used by an HMM. For the Bayesian NN, I ran net-pred with the n option which I believe produces class probabilities $p(\text{class} | \text{features})$. Yang divided these probabilities by class priors so that she could use them with the HMM.

Three probability estimators were tried:

- 1) The Bayesian NN with 226 inputs (using flag features to label missing features), 15 hidden units, trained on 90,000 points, with 92.14% frame accuracy on the test set without the HMM.

2) The Bayesian NN with 16 inputs (using the 16 features with highest ARD relevancies), 15 hidden units, trained on 200,000 points, stepsize-adjust=0.05, with 92.03% frame accuracy on the test set without the HMM.

3) The decision tree system mentioned above which uses 50 decision trees by bagging, with 91.2% frame accuracy on the test set without the HMM.

4) Another decision tree system, which Yang calls 'ensemble bagging': "I split the majority samples in the training set into N sets, each one has roughly the same number of samples as the total minority samples and is combined with the minority samples to train decision trees (also bagging is applied to each subset). Final decisions are the interpolated results from all the trees." 91.47% frame accuracy on the test set without the HMM.

The results with the HMM :

1)

Overall accuracy = $68389/71648 = 95.45\%$

Average recall = average of $58647/60280$ and $9742/11368 = 91.49\%$

Confusion matrix:

	0	S	TOTAL	CORR
0	58647	1633	60280	58647
S	1626	9742	11368	9742

2)

Overall accuracy = $68355/71648 = 95.40\%$

Average recall = 91.14%

Confusion matrix:

	0	S	TOTAL	CORR
0	58705	1575	60280	58705
S	1718	9650	11368	9650

3)

Overall accuracy = $68362/71648 = 95.41\%$

Average recall = 90.96%

Confusion matrix:

	0	S	TOTAL	CORR
0	58763	1517	60280	58763
S	1769	9599	11368	9599

4)

Overall accuracy = $68389/71648 = 95.45\%$

Average recall = 91.17%

Confusion matrix:

	0	S	TOTAL	CORR
0	58738	1542	60280	58738
S	1717	9651	11368	9651