NIPS 2003 Feature Selection Workshop

Classification for High Dimensional Problems Using Bayesian Neural Networks and Dirichlet Diffusion Trees

Radford M. Neal and Jianguo Zhang
University of Toronto

Outline of talk

1. Why and how of feature selection.
2. Our general approach — use simple techniques to reduce the computational difficulty of the problem, then apply more sophisticated Bayesian methods.
3. The simple techniques: PCA and significance tests.
4. Bayesian neural networks.
5. The Bayesian neural network models used, with large and small numbers of features.
6. Dirichlet diffusion trees.
7. The Dirichlet diffusion tree models useds.

Acknowledgements

Thanks to Geoffrey Hinton for use of a computing cluster, funded by CFI. Support also came from NSERC. Radford Neal holds a Canada Research Chair in Statistics and Machine Learning.

Why select a subset of the observed features?

You think using only a subset of features will improve performance.

Too many features cause overfitting for maximum likelihood, but not for good Bayesian methods. Dimensionality can be reduced by other means, such as principle component analysis (PCA).

You need to save computation time.

This may be necessary, especially for Bayesian methods. But again, dimensionality can be reduced in other ways.

You don't want to think too hard about your prior.

The bad effects of bad priors may be greater in high dimensions.

You want to avoid measuring lots of features in future.

To make an optimal tradeoff, you need to know the costs of measurements and of prediction errors.

The correct Bayesian approach

(ignoring practical difficulties)

Fit the best Bayesian model you can, incorporating your prior beliefs.

The best model is probably complex, and probably uses all the features.

Make predictions using this model for cases you think are representative of future test cases.

Predictions are found by integrating over the parameter space of the model.

Find the best model using a subset of features, fitting to the predicted labels in the test cases, penalizing models by the cost of features used.

This is an optimization problem — *not* involving integration over parameters.

Note: Knowing costs for prediction errors and feature use are essential. Any method that doesn't look at costs *can't* be making the right trade-off!

How to avoid selecting features

Suppose the number of training cases, n, is much less than the number of features, p. A general recipe:

Use a learning method that is invariant to rotations in the input space, and which ignores inputs that are always zero.

Examples: Neural network learning by gradient descent with early stopping, or Bayesian neural networks with a symmetrical prior on input-hidden weights.

Rotate the training cases so that only n inputs are non-zero for the training cases, then drop all but one of the zero inputs.

For later exploration, PCA is a good choice.

Rotate test cases accordingly, setting one input to the distance from the space of training cases.

I made challenge submissions of this sort using Bayesian logistic regression, neural network, and Gaussian process models. Performance was not bad, sometimes quite good.

Our pragmatic approach

The number of features for the challenge data sets is too large to be handled computationally by complex Bayesian models (except for Madelon, barely).

Our strategy:

1. Reduce the number of features by either taking the first k principle components, or by selecting k of the original features using significance tests (or use a combination of these).

2. Fit hierarchical Bayesian models that can potentially discover that some of these k inputs are less relevant than others.

3. If a smaller number of features is desired, use the relevance hyperparameters from the Bayesian model to pick a smaller subset.

4. Use the validation set to choose among various models found this way; when validation labels are known, just retrain the previously chosen model(s).

Since the Bayesian models can learn to mostly ignore some inputs, we can make k fairly big (50-1000), so as to avoid leaving out relevant features.

Dimensionality reduction with PCA

There are probably better dimensionality reduction methods than PCA, but that's what we used. One reason is that it's feasible even when p is huge, provided n is not too large — time required is of order $\min(pn^2, np^2)$.

Some issues:

- Should features be transformed before PCA? Eg, take square roots.
- Should features be centred? Perhaps not if zero is special.
- Should features be scaled to have the same variance? Perhaps not, if the original scale carries information about relevance.
- Should principle components be standardized before use? Again, maybe not.

A plausible power transformation was chosen for each feature so as to maximize correlation with the class. Whether to use these transformations, and the other choices, were made based on intuition and validation set results.

PCA was done using all the data (training, validation, and test).

Feature selection using significance tests

An initial feature subset was found by simple univariate significance tests.
Assumption: Relevant variables will be at least somewhat relevant on their own.

Three significance tests were used, applied only to features that were non-zero in at least four training cases:

- Correlation of class with ranks of feature values — sensitive to any monotonic relationship.
- Correlation of class with binary form of feature (zero/non-zero) — merging non-zero values may detect relationships that rank correlation doesn't.
- A runs test on the class labels reordered by increasing feature value (counts how often adjacent class labels are the same) — can detect non-monotonic relationships, but less powerful than the other tests.

For all tests, a p-value was found by comparing to the distribution found when permuting the class labels. Separate p-value thresholds were picked by hand for each test, and the union of the feature sets found with each test was used.

Bayesian Neural Networks

I used multilayer perceptron networks, with two hidden layers with tanh activation function. I usually used 25 hidden units in the first layer, and 8 hidden units in the second layer.

Nets with two hidden layers can easily represent some functions that are harder to represent with only one hidden layer.

Bayesian learning integrates over the posterior distribution for the network parameters, rather than picking a single "optimal" set of parameters. This avoids overfitting.

A hierarchical prior was used, in which the priors for groups of weights were controlled by higher-level hyperparameters.

It's all implemented with MCMC, using runs of about a day for each model.

For details, see my book, *Bayesian Learning for Neural Networks*.

Automatic Relevance Determination (ARD)

Using a hierachical prior, we can automatically determine how relevant each input is to predicting the class.

We group together the weights on all the connections from a particular input to the hidden units. Conditional on a “relevance hyperparameter”, the weights in this group are independent, with mean zero and variance given by the relevance hyperparameter, which is itself given a higher-level prior.

If an input is irrelevant, its relevance hyperarameter will tend to be small, forcing the weights from that input to be near zero.

ARD was used adjust the relevance of both the original features, and of the principle components.

The Arcene data

10000 features, 100 training cases, 44% positive

BayesNN-small

Used 1070 features selected using significance tests.

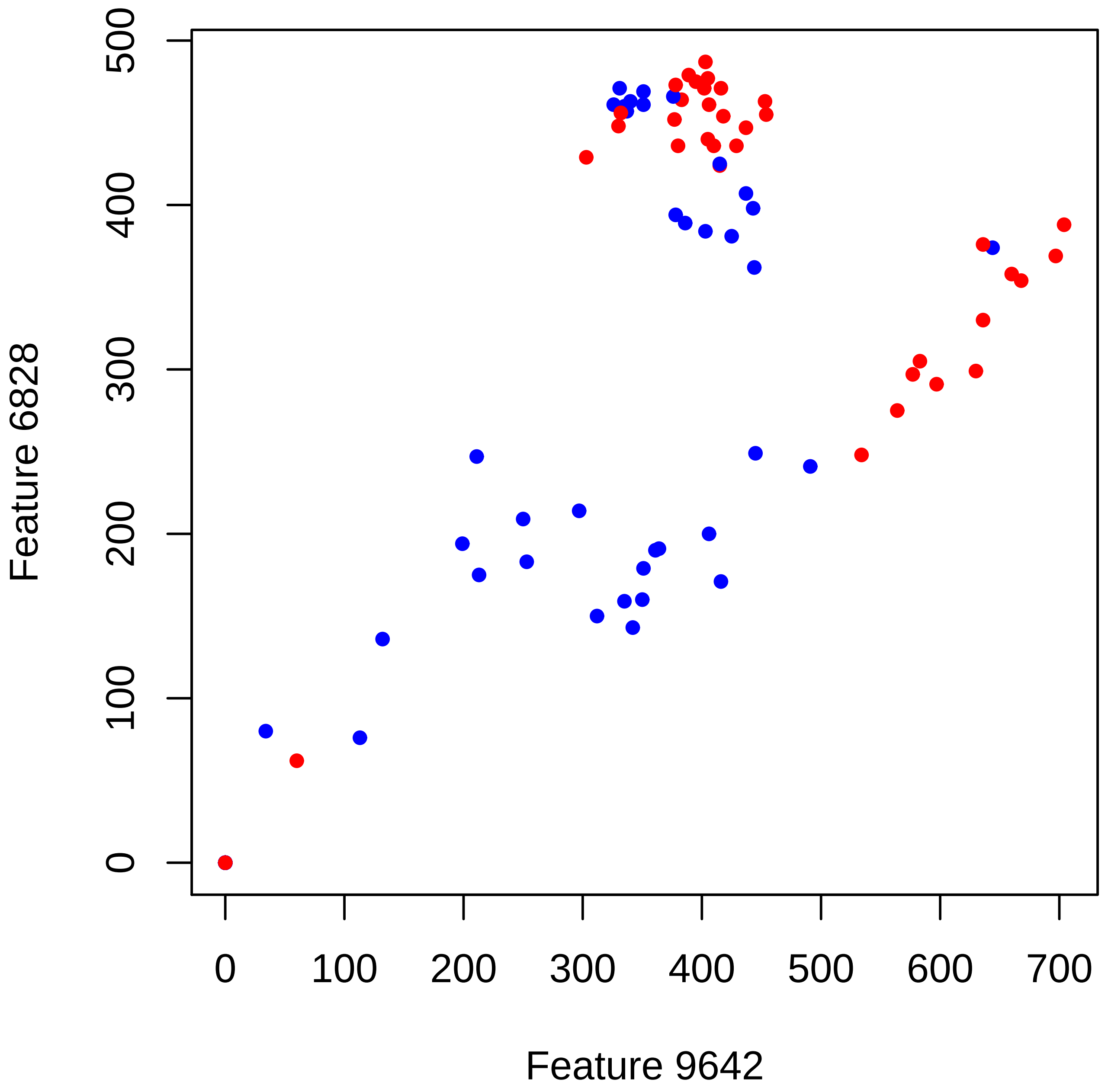
BayesNN-large

Used 50 principle components, applied to transformed features, with centering and scaling.

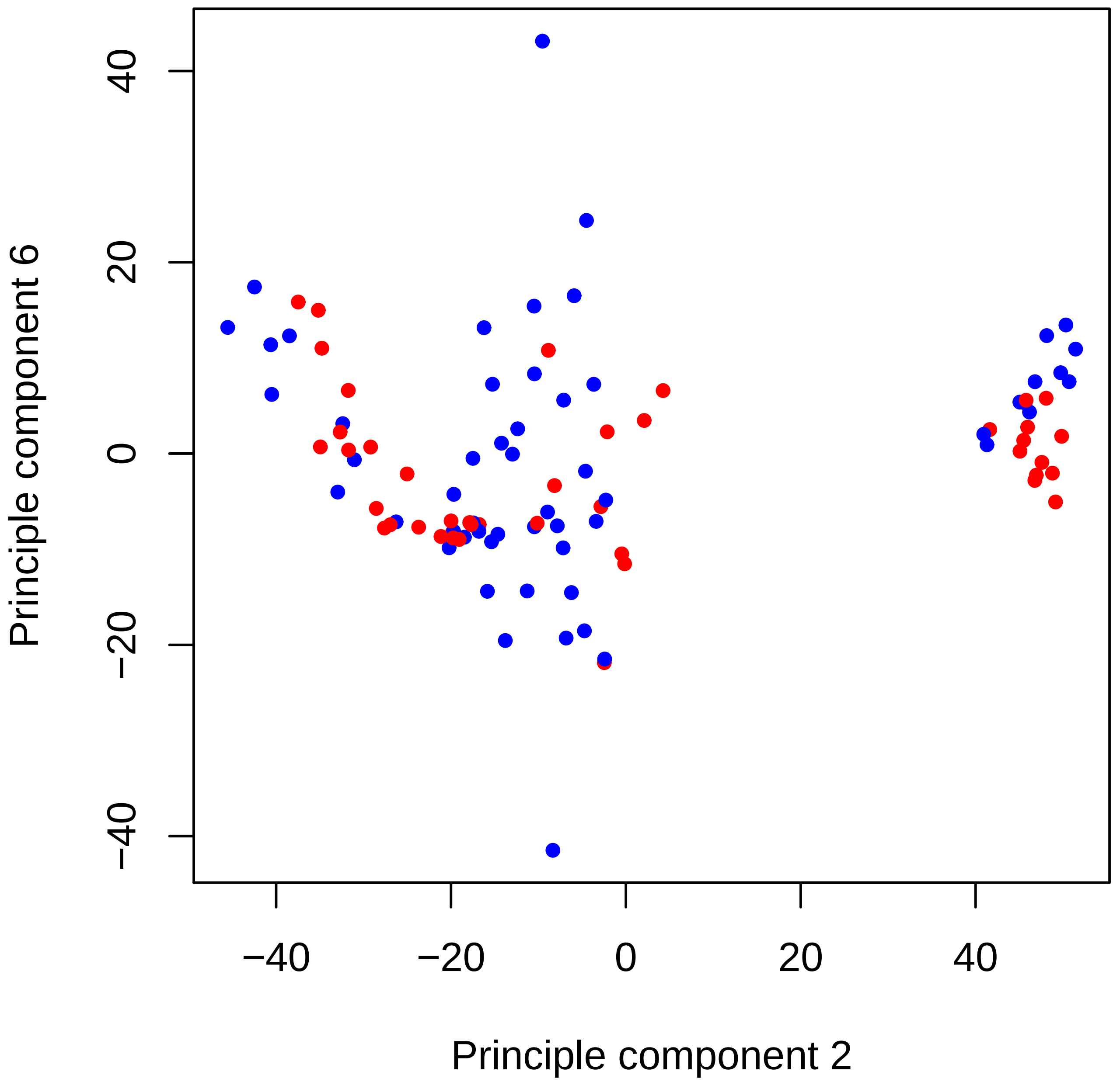
BayesNN-DFT-combo

Used a Dirichlet diffusion tree model, since it gave better results on the validation set than either of the neural network models.

Arcene: Two highly relevant features



Arcene: Two principle components



The Gisette data

5000 features, 6000 training cases, 50% positive

BayesNN-small

379 features were selected using significance tests.

BayesNN-large

50 principle components were used, found using centering and scaling.

BayesNN-DFT-combo

Same as BayesNN-large, since the large number of cases makes applying a Dirichlet diffusion tree model computationally difficult.

The Dexter data

20000 features, 300 training cases, 50% positive

BayesNN-small

Only 1458 features are non-zero in four or more training cases. 303 of these features were selected using significance tests. A few of these were picked only because of the runs test.

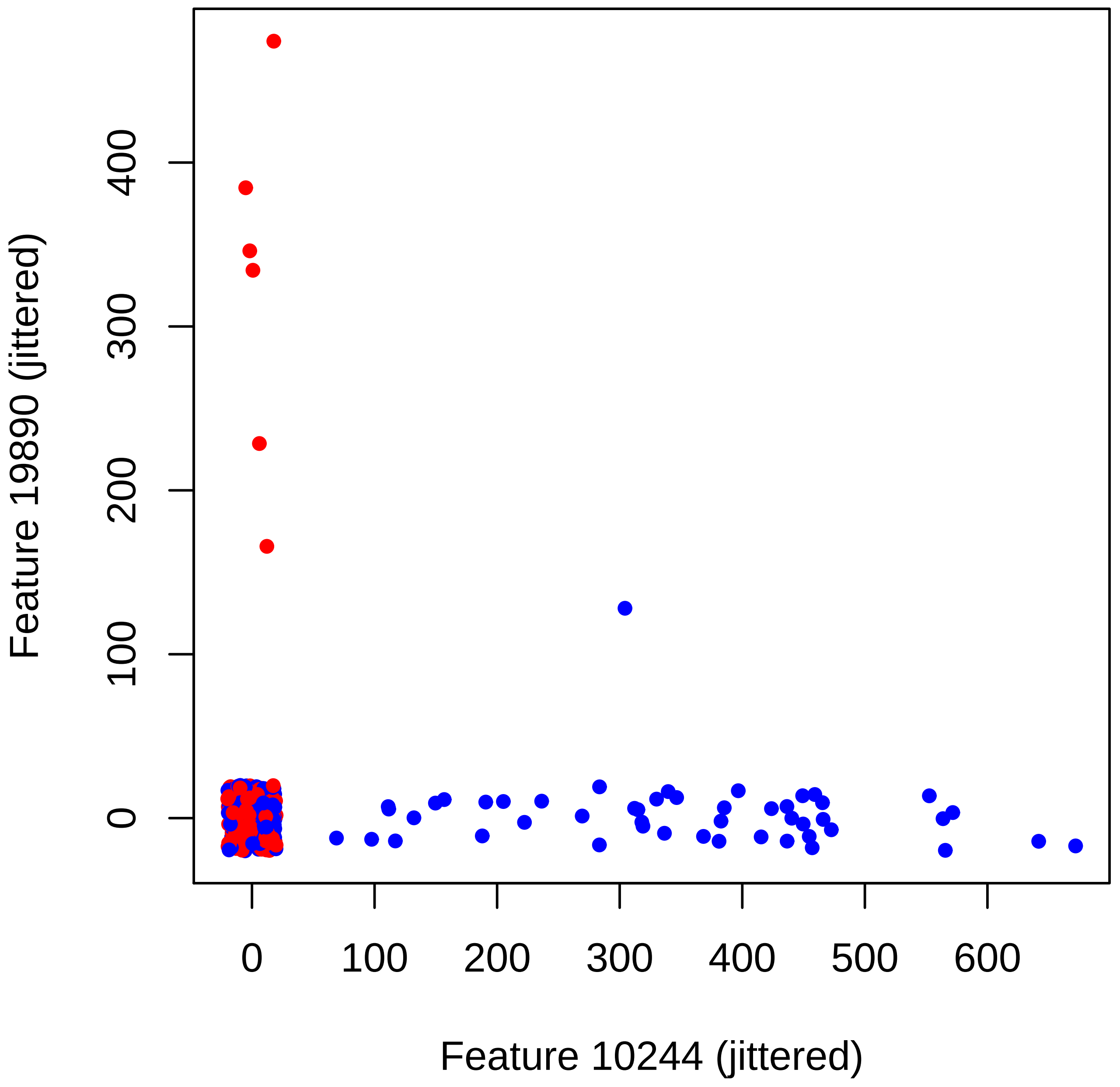
BayesNN-large

The same as BayesNN-small, since results using principle components weren't as good on the validation set.

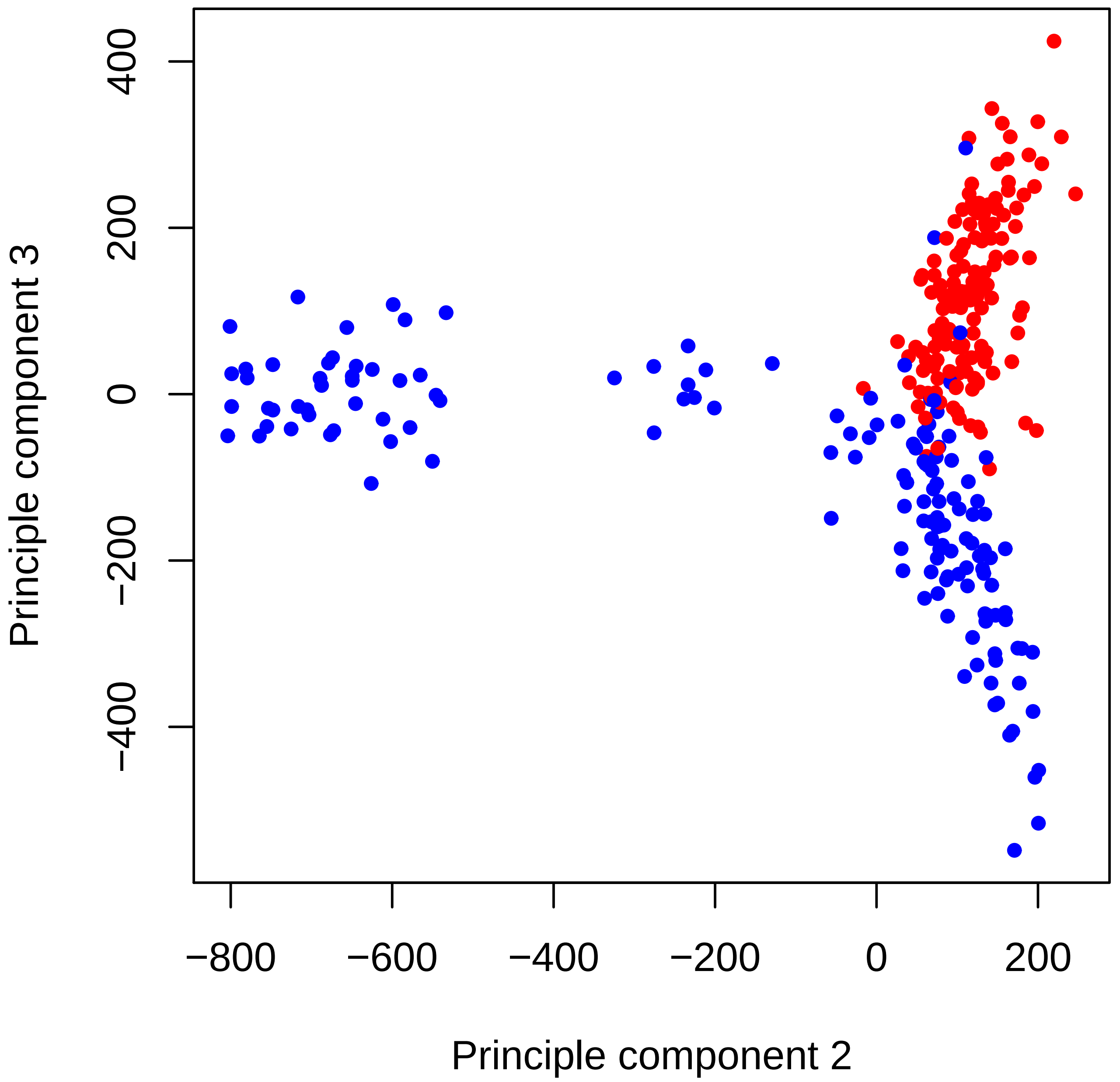
BayesNN-DFT-combo

Also the same as BayesNN-small, since results with a Dirichlet diffusion tree model weren't as good on the validation set.

Dexter: Two highly relevant features



Dexter: Two principle components



The Dorothea data

100000 features, 800 training cases, 10% positive

BayesNN-small

500 features were selected using significance tests.

BayesNN-large

50 principle components were found using features that are non-zero in at least two cases, without centering or scaling. 60 features selected using ARD hyperparameters were used as well, along with counts of the number of rare features present.

BayesNN-DFT-combo

Same as BayesNN-large, since results with a Dirichlet diffusion tree model weren't as good on the validation set.

The Madelon data

500 features, 2000 training cases, 50% positive

BayesNN-small

17 features found using the ARD hyperparameters from various models were used, including models using the entire set of features.

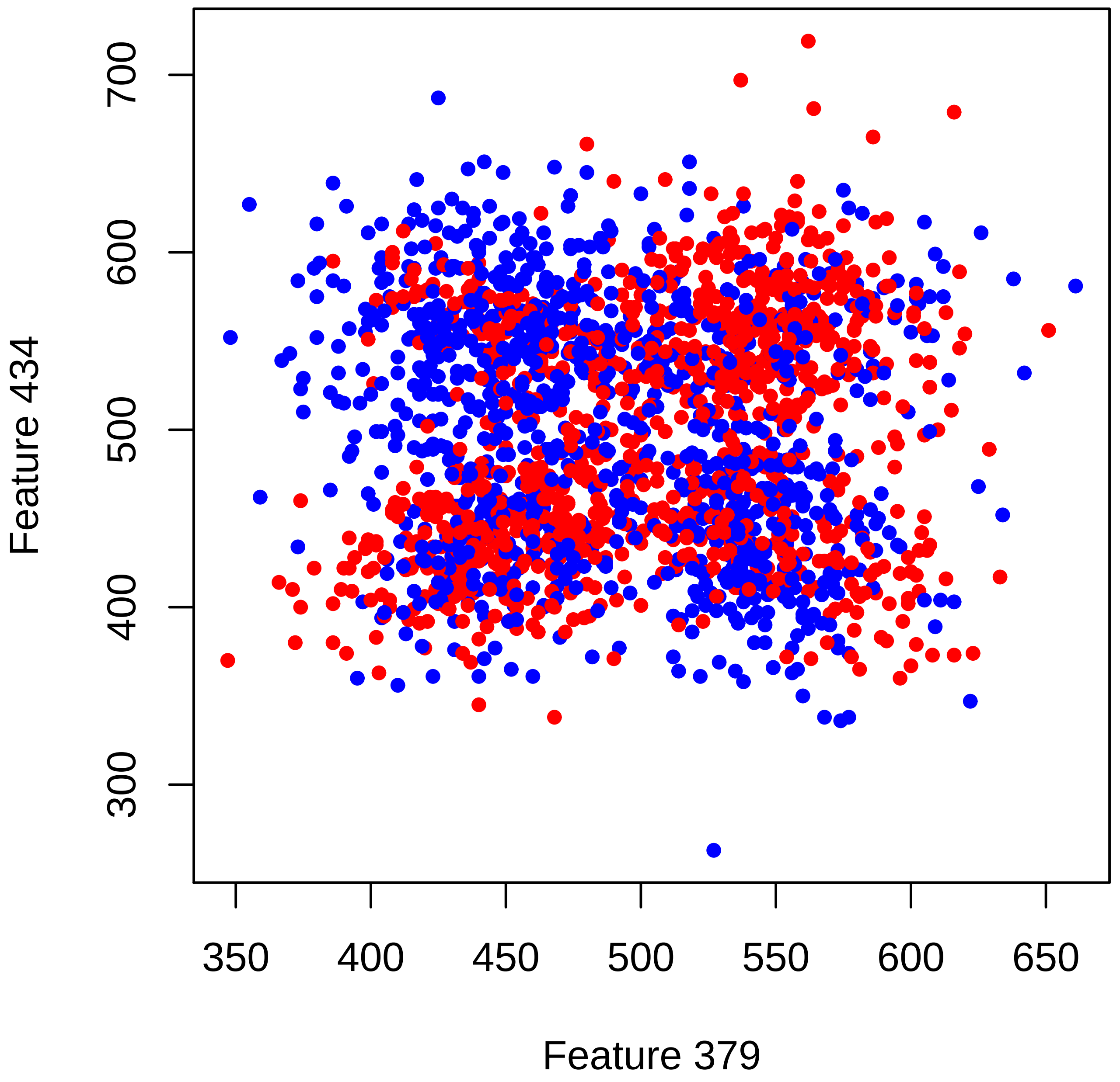
BayesNN-large

Same as BayesNN-small, since results using principle components weren't as good on the validation set.

BayesNN-DFT-combo

A Dirichlet diffusion tree model had validation error similar to that of BayesNN-small. Accordingly, we made predictions based on the average of the predictive probabilities produced by these two models.

Madelon: Two highly relevant features



Introduction to Dirichlet Diffusion Trees

- Dirichlet Diffusion Tree Prior
  - The first point is generated by a simple Gaussian diffusion process from time 0 to 1
  - The second point follows the path of the first one initially.
  - The second point diverges from the path at a random time *t*.
  - After the divergence, the second point follows a Gaussian diffusion process independent of the first one.

- The *n*-th point follows the path of those before it initially.
- The *n*-th point diverges at a random time *t*
- At a branch, the *n*-th point selects an old path with probability proportional to the numbers of points that went each way previously
- After the divergence, it follows a Gaussian diffusion process independently.
- The variances and noises of different features may be different

- Divergence Function
  - If a point following a path of *n* points did not diverge at time *t*, the probability of divergence during *dt* is *a*(*t*)*dt/n*.
  - With more points passing the way, the probability of divergence for a new point is smaller.

- Selection of Divergence function
  - Select *a*(*t*) such that

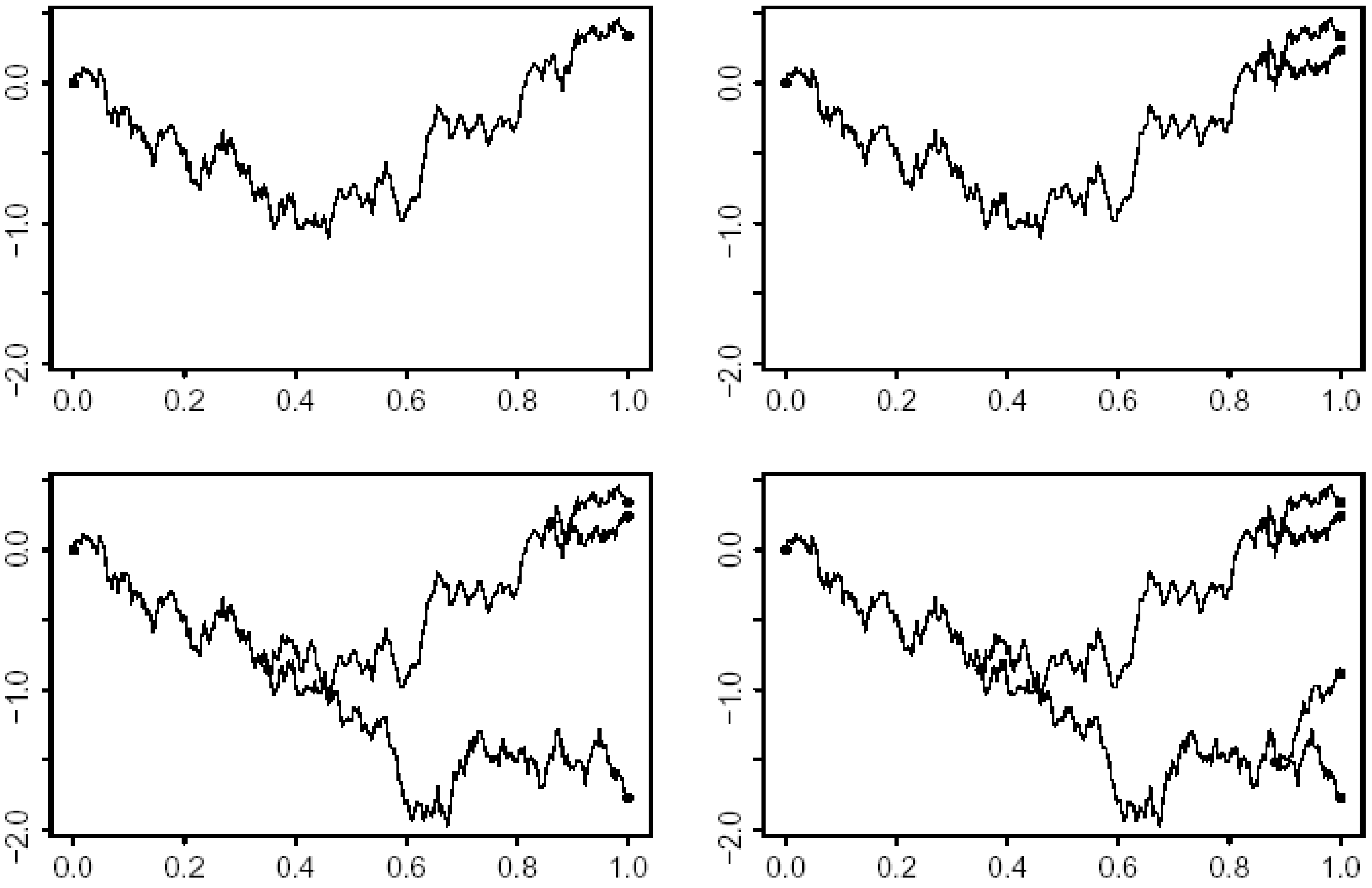
$$\int_0^1 a(t)\,dt = \infty$$

    to keep the distribution continuous.
  - Two possibilities

$$a(t) = \frac{c}{1-t}$$

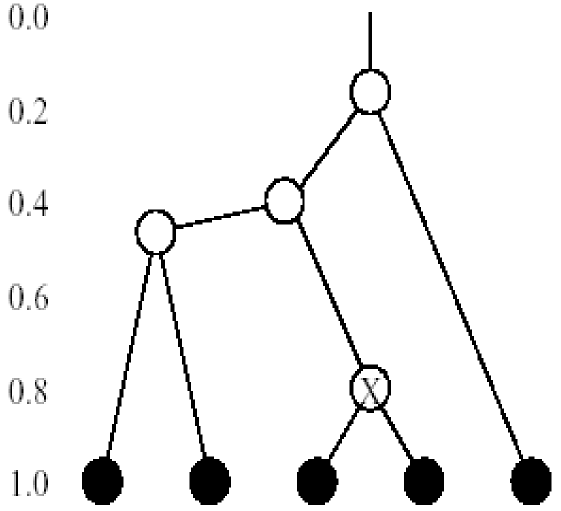
    or

$$a(t) = b + \frac{c}{(1-t)^2}$$

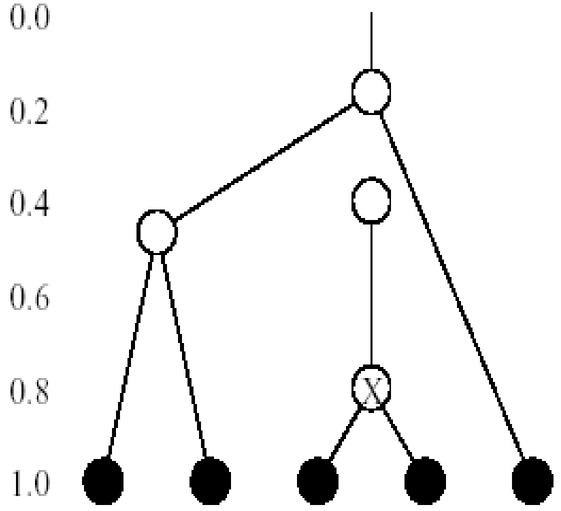

0.0
−1.0
−2.0
0.0
0.2
0.4
0.6
0.8
1.0

- Markov Chain Sampling for Dirichlet Diffusion Tree Models
  - The state of the Markov chain used to sample from the posterior distribution will consist of at least:
    - The structure of the tree
    - The divergence times for non-terminal nodes
  - Other parameters
    - Diffusion variance, noise
    - Parameters of the divergence function, etc.

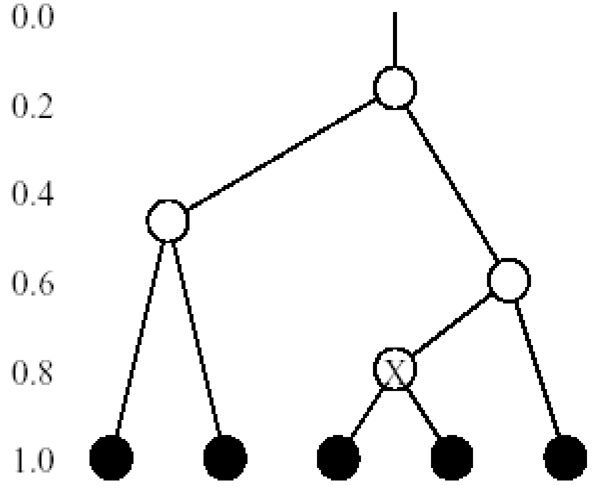
- Updating of the trees
  - The tree structure, divergence time, and node locations can all be updated by parent moves.



The original tree



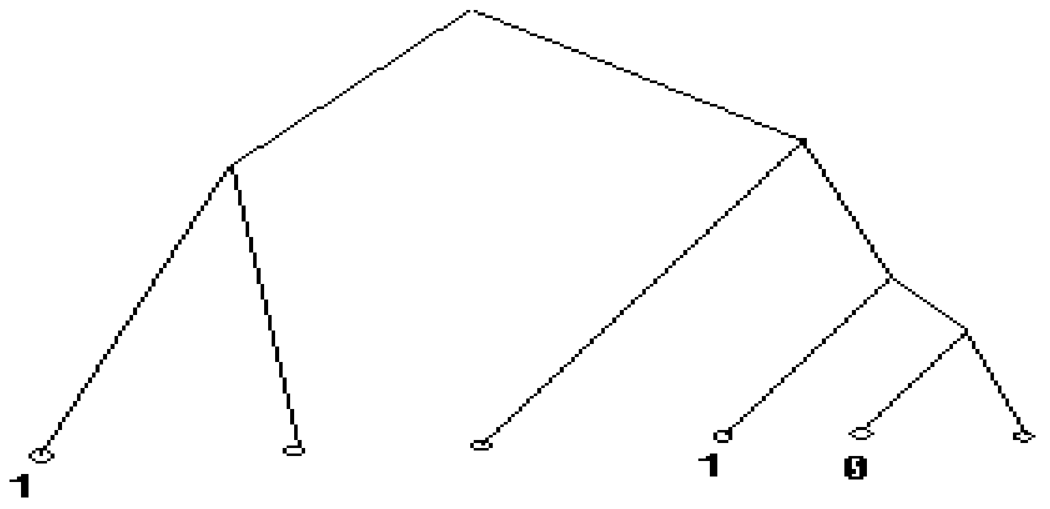
After removing X’s parent



A tree with X’s parent added back in

Methods of classification from trees

- Classify using the structure of the tree
  - Use the nearest neighbor
  - Use the nearest neighbor with branch weights



  - Take the average over trees from the posterior

- Use (*1 - divergence time*) as a measure of distance
  - Take the average of the divergence times
    - Use nearest neighbor method
    - Use (1- divergence time) as a measure of dissimilarity
      - Construct a further tree from this dissimilarity
    - Use a weighted average based on divergence times

$$P(y_i = 1) = \frac{\sum_{y_j = 1,\, j \neq i} (e^{rd_{ij}} - 1)}{\sum_{j \neq i} (e^{rd_{ij}} - 1)}$$

      - r is selected by cross validation

- Other tree-based classification methods
  - Gaussian processes
  - The method of Kemp, et al.

- Results for the Arcene dataset and Madelon Dataset (on validation sets)

	Arcene	Madelon
Number of principle components	100/900	6/500
Nearest neighbor with branch weight	9.99%	7.83%
Weighted average divergence times	9.25%	10.17%
Dissimilarity	9.01%	--

- Reference
  - Neal, R. M. (2003), Density Modeling and Clustering Using Dirichlet Diffusion Trees, *Bayesian Statistics 7, 619-629*
  - Neal, R. M. (2001), Defining priors for distributions using Dirichlet diffusion trees. *Tech. Rep. No. 0104*, Department of Statistics, University of Toronto, Canada

Conclusions

- Complex Bayesian models can be used for datasets with large numbers of features, after reducing dimensionality one way or another.
- The resulting performance is very good.
- Validation set results are a noisy guide to model selection.
- Selecting between Bayesian models is still a difficult problem — particulary when there are computational issues (did the MCMC really converge?).
- MCMC software used is available from www.cs.toronto.edu/∼radford.