Comparison of Adaptive Neural Classification Techniques

Malcolm Stagg
Electrical and Computer Engineering
Louisiana State University
Baton Rouge, LA 70803
mstagg3@lsu.edu

Suresh Rai
Electrical and Computer Engineering
Louisiana State University
Baton Rouge, LA 70803
srai@lsu.edu

Abstract

Online and offline classification learning algorithms of the Random Forest [1, 2] and AdaBoost [3] are implemented in Matlab and compared on data from the LIBSVM datasets [4] in terms of their accuracy and training time. Comparison also takes place to a standard multilayer perceptron trained with backpropagation. Random Forest with a large enough tree depth exhibited the overall best results; the online version is slower but eventually approaches the same error rate. AdaBoost and the multilayer perceptron classify certain datasets with very high accuracy, but have much lower accuracy on others.

1 Introduction

Classification is an important problem with many applications, ranging from security [5] to medical purposes and spam filtering [6]. It is desirable to design a robust supervised classifier for such a problem without the need for manually setting many parameters in order to obtain a high degree of accuracy, and high speed. Ideally such a classifier is also capable of online learning, such that as a distribution of data changes, it will be able to continue learning without retraining the entire classifier on a new dataset.

In this paper, a set of supervised classification algorithms, online and offline, are compared for accuracy and training time. To allow for fair comparison, each algorithm is implemented using similar techniques in Matlab, with minimal optimization. It should be noted that several of the algorithms used can be optimized to have a GPU implementation [7], so a long learning/evaluation time is not always a drawback in the results given in this paper.

Data is imported from the DNA, Letter, Mushrooms, SatImage, and USPS datasets from LIBSVM [4], and normalized such that each feature has a range between 0 and 1. This is for the ease of online learning algorithms, which cannot immediately determine the minimum and maximum values of features it will encounter.

The layout of this paper is as follows: Section 2 provides background information on four typical approaches that we have used for comparison. Section 3 considers a sample learning problem. We give results for some LIBSVM datasets [4] in Section 4. Finally, Section 5 concludes the paper. Furthermore, Appendix 1 gives pseudocode, while Appendix 2 illustrates typical parameters that we have used to implement four typical approaches discussed in Section 2.

2 Background

This section provides a brief description of the four approaches of the Offline Random Forest, Online Random Forest, AdaBoost, and Multilayer Perceptron, which are implemented in Matlab.

2.1 Offline Random Forest

The offline Random Forest, based on the original paper by Breiman [1] and with some implementation details from [8], is implemented with the parameters for number of trees (T), maximum layers (L), number of random tests per node (N_T), and number of features per test (N_F).

Bagging first takes place to give each tree a random subset, with repetitions, of the data. It is decided to make $M = N$, where M is the number of samples presented to each tree and N is the total number of input data. Due to the bagging, some data will be repeated multiple times, and some will not be presented at all to a given tree.

The algorithm works by starting out with a set of N_T random tests for a single root node, each based on the parameter N_F. A small number for N_F, such as 1 or 2, is found to work well. The test is evaluated such that a positive result will cause the data to propagate to the right child, and negative result to the left child. The test with the most positive information gain is kept, and the node is split. If the information gain is not positive, or the node is on layer L, splitting does not take place.

Parameters α and β are also implemented to match the online Random Forest below, but the best performance is found to occur when they are both set to zero.

2.2 Online Random Forest

The online Random Forest is based on [2], though some discrepancies are found from the parameters given by the authors for the test datasets [4]. Parameters are implemented for α: the minimum number of samples a node must have received for it to split, and β: the minimum information gain of the node. Parameters for number of trees (T) and maximum layers (L), number of random tests (N_T), and number of features per tests (N_F), are again implemented, as well as the number of epochs of input data, given as 10 in [2].

Bagging takes place using the Poisson distribution with parameter 1 as an estimate [9]. Each sample is presented zero or more times based on a random number from the Poisson distribution.

The mechanics of the tree are very similar to the offline version, though each node only receives a single data sample at a time. After reaching a minimum number α, if the best information gain is above β, and the node is not on layer L, a split takes place. When a node splits, the number of samples and sample distributions are propagated to child nodes, and each child node generates a new set of N_T random tests.

The two information gain measures of the Gini index and entropy are implemented, but there does not appear to be a significant difference in terms of overall error. Thus, for improved speed due to its lower computational complexity, the Gini index is used as an information measure (1), and for information gain (2).

$$info_j = \sum_{i=1}^{C} p_i^j (1 - p_i^j) \tag{1}$$

$$\Delta info_j = info_j - \frac{n_{l(j)}\, info_{l(j)} + n_{r(j)} info_{r(j)}}{n_j} \tag{2}$$

where p_i^j is probability of class i at node j, C is the number of classes, n_j is the number of samples propagated to node j, and $l(\cdot)$ and $r(\cdot)$ are indices of left and right split data, respectively.

Discrepancies are found between the parameter values reported in [2] and the actual values required to obtain similar results. In particular, $\beta \approx 0$ rather than 0.1 is needed, and $\alpha \approx 150$ rather than $0.1*N$, where N is the number of samples. The reason for this discrepancy is not known, but it is seen in the code given by [2] that $\beta = 0$ is actually used in their implementation.

2.3 Multi-class AdaBoost

AdaBoost is first implemented using the binary classification version, detailed in [3] with implementation details from [5, 10]. For use with the LIBSVM dataset, which contains several multi-class problems, an extension based on [11] is used.

Each weak learner is implemented as a single layer perceptron, with an output corresponding to each class. The maximum output from each weak learner is chosen as its classification result, which is weighted by the AdaBoost algorithm to determine the overall best prediction.

It is decided to implement data weights $\boldsymbol{D}$ (see Algorithm 3 in Appendix 1 for a description) as a probability distribution used to sample the data for each classifier, with replacement, rather than altering the learning algorithm to utilize weighted data [12]. This boosting distribution scheme can be seen to be in fact *identical* to bagging when the distribution is uniform. Thus, the same criteria $M = N$, where M is the number of samples presented to each tree and N is the total number of input data, is used for AdaBoost as is used with the Random Forest.

AdaBoost has an adjustable parameter T, for the number of classifiers used. It can be considered equivalent to T in the Random Forest algorithm for number of trees, as each tree is a weak classifier.

2.4 Multilayer Perceptron

The Matlab implementation of backpropagation is used, using the *trainscg* algorithm, due to intensive memory requirements of the default *trainlm* [13], which made it impractical to use on a large dataset. Initially 2 hidden layers, each with $N_h = F$ nodes, where F is the number of features in the input data, are used, and each layer using the *tansig* activation function.

After noticing poor performance with some datasets, the hidden layer sizes were increased to $N_h = 2F$.

Several other parameters are modified in an attempt to prevent the learning from stopping before adequate classification is achieved. In particular, a stop condition occurs when the error rate $\varepsilon < \varepsilon_{min}$, or validation checks fail v_{fail} times. It is decided to use 10% of the training data for such validation, to prevent overtraining.

One output neuron is used for each class, and the node with the maximum value is used as the classification result. The node corresponding to the target is trained to 1, while all other output nodes are trained to -1 for each given input.

3 Illustrative Example

As a sample problem, two sets of points are arranged in a pattern which is not linearly separable. This problem, shown in fig. 1 below, is originally given by Hagan [14].

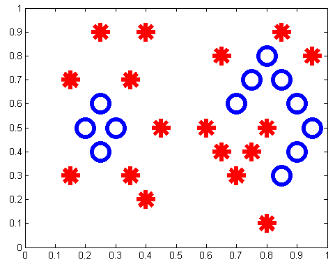

Figure 1: Test data from Hagan [14]

The decision boundaries of a single tree (weak learner) can be seen below with the offline Random Forest algorithms below, with 1 or 2 features per test (fig 2).

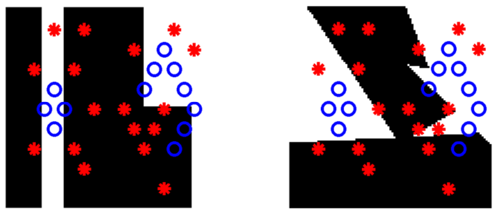

Figure 2: Single tree results with 1 (left) or 2 (right) features/test

With 100 trees voting on the class (strong learner), the decision boundaries change as follows (fig 3):

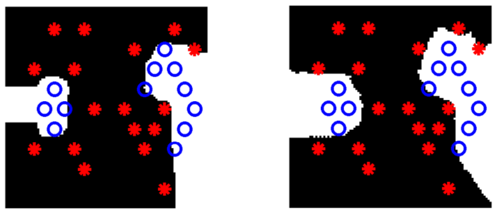

Figure 3: Forest results with 1 (left) or 2 (right) features/test

It can be seen that, while the single tree makes several classification errors, in this case the forest makes none.

Similar results are obtained for the online Random Forest algorithm after training for 10 epochs.

Results from a single AdaBoost weak classifier, and the entire strong learner are shown in fig. 4. While the decision lines are more jagged, there are again no errors.

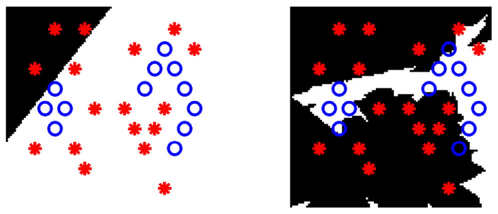

Figure 4: Decision boundaries of single weak learner (left) and strong learner result (right) for AdaBoost

As a comparison, the multilayer perceptron decision space, where 2 hidden layers of 3 neurons are trained over 39 epochs can be seen in fig 5.

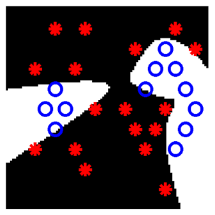

Figure 5: Decision boundaries of Multilayer Perceptron

Networks with hidden layers smaller than 3 nodes are unable to learn this problem accurately.

4 Results and Discussion

Table 1 shows the classification results for each algorithm, obtained over 3 trials, taken from the LIBSVM Datasets [4], and a synthetic dataset generated from 5 gaussians (fig. 6), shown in green (G), red (R), cyan (C), purple (P), and blue (B).

The training time, in seconds, of each algorithm is also measured, over all training samples for offline learners, and over 1 epoch for online learning (table 2).

Table 1: Classification Accuracy Results
(Percent Error ± standard deviation)

Dataset	RF	ORF	AB	MLP
DNA	9.9 ± 0.4	13.4±1.1	6.8 ± 0.2	7.8 ± 0.2
Letter	6.1 ± 0.2	7.0 ± 0.2	47.2±5.9	14.2±2.7
Mushrooms	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.1	0.0 ± 0.0
SatImage	12.1±0.1	12.9±0.1	26.4±4.7	10.7±0.3
USPS	8.9 ± 0.1	7.5 ± 0.2	9.2 ± 0.6	11.2±4.9
5 Gaussian	0.7 ± 0.0	0.3 ± 0.1	1.0 ± 0.0	0.8 ± 0.0

RF –Random Forest ORF – Online Random Forest
AB –AdaBoost MLP – Multilayer Perceptron

Table 2: Training Time Results (seconds)

Dataset	RF	ORF	AB	MLP
DNA	88	41	32	33
Letter	97	458	2549	128
Mushrooms	34	183	13	44
SatImage	48	93	23	22
USPS	201	242	563	389
5 Gaussian	4	14	14	3

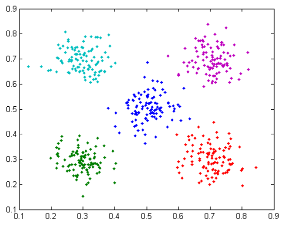

Figure 6: Synthetic 5-gaussian dataset
N(μ_x, μ_y, σ_x, σ_y) for G, R, C, P, B are as follows:
(0.3, 0.3, 0.1, 0.1), (0.7, 0.3, 0.1, 0.1), (0.3, 0.7, 0.1, 0.1), (0.7, 0.7, 0.1, 0.1), (0.5, 0.5, 0.1, 0.1)

Initially, the Random Forest with maximum depth of 10 layers exhibits poor classification abilities for the "Letter" dataset in particular, so the effect of changing the maximum depth is experimented on. The results for a tree depth of 10, 20, and 30 layers can be seen in table 3.

The Online Random Forest has a similar comparison, though generally with slightly higher error rates (for all except the "USPS" and 5-gaussian dataset).

Similarly, the effect of increasing the hidden layer size of the Multilayer Perceptron from F to $2F$, where F is the number of features, is tested (table 4).

Table 3: Comparison of Tree Depth
(Percent Error ± standard deviation)

Dataset	10 depth	20 depth	30 depth
DNA	8.6 ± 1.5	10.4 ± 0.6	9.9 ± 0.4
Letter	26.9±1.8	7.7 ± 0.4	6.1 ± 0.2
Mushrooms	0.7 ± 0.1	0.0 ± 0.0	0.0 ± 0.0
SatImage	14.8±0.1	12.1 ± 0.4	12.1 ± 0.1
USPS	13.2 ±0.3	9.3 ± 0.1	8.9 ± 0.1
5 Gaussian	0.8 ± 0.1	0.7 ± 0.1	0.7 ± 0.1

Table 4: Comparison of Hidden Layer Size
(Percent Error ± standard deviation)

Dataset	$N_h = F$	$N_h = 2F$
DNA	21.9±24	7.8 ± 0.2
Letter	47.1±24	14.2 ± 2.7
Mushrooms	0.0 ± 0.0	0.0 ± 0.0
SatImage	11.2±0.6	10.7 ± 0.3
USPS	5.3 ± 0.2	11.2 ± 4.9
5 Gaussian	10.1±0.2	0.8 ± 0.0

5 Conclusion

It is found that Offline Random Forest performed very well overall for all the datasets, given a large enough number of layers.

Online Random Forests are generally not as accurate, and require much longer training time than the offline counterpart to obtain similar results. As well, it is difficult to determine good criteria for selecting parameters such as the maximum depth, and minimum number of samples for a node to split. With the latter based on the total number of samples, not enough splits took place to accurately classify the data for the "Letter" dataset, so a constant value of 150 is selected instead.

This being said, it appears to be true that the performance of the Online Random Forest does approach (or even exceed, as is the case with the "USPS" dataset) the performance of offline, given enough training samples, as is claimed by [2].

Increasing the maximum depth of a tree, whether online or offline, is often able to increase the overall accuracy. However, it comes at a cost of increasing the learning and evaluation time, as well as the potential for overtraining.

AdaBoost has good results with some of the datasets, but very poor results for others, in particular "Letter" and "SatImage." Only a perceptron weak learner is selected, so it is possible the results could be improved if another algorithm such as decision stumps were used instead.

The Multilayer Perceptron has some parameters which are difficult to determine how to optimize for the test cases, such as number and size of layers, and learning parameters such as the validation size. It performs well on all datasets except "Letter", and has fast learning speed when small hidden layers are used. Learning speed decreases greatly with the number of neurons.

The theoretical speed advantages of an algorithm such as Random Forest are not very noticeable in the actual Matlab implementation. However, with optimizations and use of a lower level language, these speed advantages would likely be more apparent.

References

[1] L. Breiman, "Random Forests," Machine Learning, vol. 45, issue 1, pp. 123-140, 2001.

[2] A. Saffari et al, "On-line Random Forests," OLCV, 2009.

[3] Y. Freund and R. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," Journal of Computer and System Sciences, vol. 55, issue 1, pp. 119-139, 1997.

[4] C. Chang and C. Lin, "LIBSVM: a library for support vector machines," ACM Transactions on Intelligent Systems and Technology, vol 2, issue 3, pp. 27:1-27:27, 2011. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm

[5] J. Meynet, "Fast Face Detection Using AdaBoost," 2003.

[6] X. Carreras and L. Màrquez, "Boosting Trees for Anti-Spam Email Filtering," Proceedings of RANLP, 2001.

[7] D. Slat and M. Lapajne, "Random Forests for CUDA GPUs," Thesis, Blekinge Institute of Technology, 2010.

[8] J. Shotton. et al, "Boosting & Random Forests for Visual Recognition," ICCV tutorial, 2009. Available: http://mi.eng.cam.ac.uk/~tkk22/iccv09_tutorial

[9] N. Oza and S. Russell, "Online Bagging and Boosting," Proc. Artificial Intelligence and Statistics, pp. 105-112, 2001.

[10] C. Autermann, "Boosted Decision Trees," HEP Seminar, Hamburg, 2007. Available: http://www.desy.de/~auterman/

[11] J. Zhu et al, "Multi-class AdaBoost," Technical report, Stanford University. Available: http://www-stat.stanford.edu/ hastie/Papers/samme.pdf

[12] Y. Freund and R. Schapire, "Experiments with a New Boosting Algorithm", Proceedings of the Thirteenth International Conference on Machine Learning, 1996.

[13] M. Beale et al, "Neural Network Toolbox 7 - User's Guide," 2010. Available: http://www2.cs.siu.edu/~rahimi/cs437/slides/nnet.pdf

[14] G. Hagan et al, "Neural Network Design," Campus Publishing Service, Colorado University Bookstore, 1996.

Appendix 1: Learning Algorithms

Algorithm 1: Offline Random Forest

```
Given inputs $\boldsymbol{data}$ of size $N$:
1:  for $t \leftarrow 1$ to $T$
2:      $\boldsymbol{x}_{j0} \leftarrow$ Bagging($\boldsymbol{data}$, $M$)
3:      for each node $j$ in tree $t$
4:          $\boldsymbol{tests} \leftarrow$ RandomTests($N_T$, $N_F$)
5:          for s $\leftarrow$ 1 to $\mathrm{N_T}$
6:              $\boldsymbol{x}_{LS}(s)$, $\boldsymbol{x}_{RS}(s) \leftarrow$ EvalTest($\boldsymbol{tests}(s)$, $\boldsymbol{x}_j$)
7:              $\Delta info_j$(s) $\leftarrow$ InfoGain($\boldsymbol{x}_j$, $\boldsymbol{x}_{LS}(s)$, $\boldsymbol{x}_{RS}(s)$)
8:          end
9:          $besttest \leftarrow \mathrm{argmax}_s(\Delta info_j)$
10:         if $\Delta info_j(besttest) > 0 \wedge \mathrm{layer}(j) < L$
11:             SplitNode($j$, $\boldsymbol{x}_j$, $\boldsymbol{tests}(besttest)$)
12:             $\boldsymbol{x}_{l(j)} \leftarrow \boldsymbol{x}_{LS}(besttest)$
13:             $\boldsymbol{x}_{r(j)} \leftarrow \boldsymbol{x}_{RS}(besttest)$
14:         end
15:     end
16: end
```

Algorithm 2: Online Random Forest

```
Given each input $\boldsymbol{x}$ in $\boldsymbol{data}$, for each epoch 1 to $E$:
1:  for $t \leftarrow 1$ to $T$
2:      $k \leftarrow$ Poisson(1)
3:      repeat $k$ times
4:          $j \leftarrow$ LeafNode($t$, $\boldsymbol{x}$)
5:          $\boldsymbol{stat}_j \leftarrow$ UpdateStatistics($t$, $j$, $\boldsymbol{x}$)
6:          for s $\leftarrow$ 1 to $\mathrm{N_T}$
7:              $\boldsymbol{n}_L(s)$, $\boldsymbol{n}_R(s) \leftarrow$ EvalTest($\boldsymbol{tests}(t, j, s)$, $\boldsymbol{x}$)
8:              $\boldsymbol{ct}_{j,L}(s) \leftarrow \boldsymbol{ct}_{j,L}(s) + \boldsymbol{n}_L(s)$
9:              $\boldsymbol{ct}_{j,R}(s) \leftarrow \boldsymbol{ct}_{j,R}(s) + \boldsymbol{n}_R(s)$
10:             $\Delta info_j(s) \leftarrow$ InfoGain($\boldsymbol{stat}_j$, $\boldsymbol{ct}_{j,L}(s)$, $\boldsymbol{ct}_{j,R}(s)$)
11:         end
```

```
12:         $besttest \leftarrow \text{argmax}_s(\Delta info_j)$
13:         if $\Delta info_j(besttest) > \beta \wedge \|\boldsymbol{stats_j}\| > \alpha \wedge \text{layer}(j) < L$
14:             SplitNode($t$, $j$)
15:             $\boldsymbol{stat}_{l(j)} \leftarrow \boldsymbol{ct}_{j,L}(besttest)$
16:             $\boldsymbol{stat}_{r(j)} \leftarrow \boldsymbol{ct}_{j,R}(besttest)$
17:             $\boldsymbol{tests}(t, l(j)) \leftarrow$ RandomTests($N_T$, $N_F$)
18:             $\boldsymbol{tests}(t, r(j)) \leftarrow$ RandomTests($N_T$, $N_F$)
19:             $\boldsymbol{ct}_{l(j),L}, \boldsymbol{ct}_{l(j),R}, \boldsymbol{ct}_{r(j),L}, \boldsymbol{ct}_{r(j),R} \leftarrow \mathbf{0}$
20:         end
21:     end
22: end
```

Algorithm 3: Multi-Class AdaBoost

```
Given inputs $\boldsymbol{data}$ of size $N$:
1:  for $t \leftarrow 1$ to $T$
2:      $\boldsymbol{D} \leftarrow$ UniformDistribution($N$)
3:      $\boldsymbol{x} \leftarrow$ SampleDistribution($\boldsymbol{data}$, $\boldsymbol{D}$, $M$)
4:      TrainWeakLearner($t$, $\boldsymbol{x}$)
5:      $\boldsymbol{h}^t \leftarrow$ WeakLearner($\boldsymbol{x}$)
6:      $\varepsilon \leftarrow \sum_{i=1}^{M} \boldsymbol{D}_i I(target(\boldsymbol{x}_i) \neq \boldsymbol{h}_i^t)$
7:      $\alpha_t \leftarrow \log((1 - \varepsilon)/\varepsilon) + \log(C - 1)$
8:      $\boldsymbol{D}_i \leftarrow \boldsymbol{D}_i \exp(\alpha_t I(target(\boldsymbol{x}_i) \neq \boldsymbol{h}_i^t))$
9:      $\boldsymbol{D} \leftarrow$ Normalize($\boldsymbol{D}$)
10:     if $\varepsilon > \varepsilon_{max} \wedge t > T_{min}$: StopTraining
11: end
```

Algorithm 4: Multilayer Perceptron

```
Given inputs $\boldsymbol{data}$ of size $N$:
1:  $\boldsymbol{net} \leftarrow$ NeuralNet([$F$;$N_h$;$N_h$;$C$], [tansig;tansig;tansig])
2:  $\boldsymbol{data}_{train}, \boldsymbol{data}_{val} \leftarrow$ ValidationSet($\boldsymbol{data}$, $v_{size}$)
3:  $bestval \leftarrow \infty$, $valfails \leftarrow 0$
4:  for $epoch = 1$ to $E_{max}$
5:      $\boldsymbol{outputs}, \boldsymbol{errors} \leftarrow$ PresentData($\boldsymbol{net}$, $\boldsymbol{data}_{train}$)
6:      $\boldsymbol{net} \leftarrow$ PresentFeedback($\boldsymbol{net}$, $\boldsymbol{errors}$, trainscg)
7:      $\varepsilon \leftarrow$ MSE($\boldsymbol{errors}$)
8:      if $\varepsilon < \varepsilon_{min}$: StopTraining
9:      $\boldsymbol{outputs}_{val}, \boldsymbol{errors}_{val} \leftarrow$ PresentData($\boldsymbol{net}$, $\boldsymbol{data}_{val}$)
10:     $\varepsilon_{val} \leftarrow$ MSE($\boldsymbol{errors}_{val}$)
11:     if $\varepsilon_{val} < bestval$
12:         $bestval \leftarrow \varepsilon_{val}$
13:         $valfails \leftarrow 0$
14:     else
15:         $valfails \leftarrow valfails + 1$
16:         if $valfails > v_{max}$: StopTraining
17:     end
18: end
```

Appendix 2: Selection of Final Parameters

After some experimentation, the following values were selected for use in this paper:

Table 5: Offline Random Forest Parameters

Number of Trees	T	100
Max Depth (or layers)	L	30
Bagging Size	M	N (number samples)
Random Tests / Node	N_T	10
Random Features / Test	N_F	2
Min Samples for Split	α	0
Min Info Gain for Split	β	0

Table 6: Online Random Forest Parameters

Number of Trees	T	100
Max Depth (or layers)	L	30
Epochs	E	10
Random Tests / Node	N_T	10
Random Features / Test	N_F	2
Min Samples for Split	α	150
Min Info Gain for Split	β	0

Table 7: Multi-Class AdaBoost Parameters

Number of Classifiers	T	100
Sample Size	M	N (number samples)
Type of Classifier	Simple Perceptron	
Stopping Error (max)	ε_{max}	$1 - 1/C$ (C classes)
Minimum Classifiers	T_{min}	$T/10$

Table 8: Multilayer Perceptron

Hidden Layers	H	2
Size of Hidden Layers	N_h	$2F$, (F features)
Learning Rule	*Trainscg*	
Max Epochs	E_{max}	1000
Stopping Error (min)	ε_{min}	1E-10
Validation Size	v_{size}	$N/10$ (N samples)
Max Validation Fails	v_{max}	50