C2FS: An Algorithm for Feature Selection in Cascade Neural Networks

Lars Backstrom
Computer Science
Cornell University
lb87@cornell.edu

Rich Caruana
Computer Science
Cornell University
caruana@cs.cornell.edu

Abstract

Wrapper-based feature selection is attractive because wrapper methods are able to optimize the features they select to the specific learning algorithm. Unfortunately, wrapper methods are prohibitively expensive to use with neural nets. We present an *internal wrapper* feature selection method for Cascade Correlation (C2) nets called C2FS that is 2-3 orders of magnitude faster than external wrapper feature selection. This new internal wrapper feature selection method selects features at the same time hidden units are being added to the growing C2 net architecture. Experiments with five test problems show that C2FS feature selection usually improves accuracy and squared error while dramatically reducing the number of features needed for good performance. Comparison to feature selection via an information theoretic ordering on features (gain ratio) shows that C2FS usually yields better performance and always uses substantially fewer features.

1. Background

Many learning problems have extraneous features that are not helpful to the learned classifier. When there are many extraneous features, learning generally becomes harder, as the model is likely to find spurious relationships between extraneous features and the training outputs that do not generalize well. Feature selection is one way to avoid this. A common approach to feature selection is to use a wrapper method [Kittler, 1978][Kohavi and John, 1997] to find a subset of features that yields good performance with the specific learning method. Feature selection wrapper methods usually hold aside some training cases to use as a feature selection set, and try to find a subset of features for which the learning method performs best on that held-out feature selection set.

One common wrapper method is greedy forward stepwise feature selection [Kittler, 1978] in which features are added to the feature set one at a time. To determine which feature to add at each step, a model is trained adding each unused feature one at a time to the feature set selected previously. The feature that provides the best performance on the held-out feature selection set is added to the set. The major drawback to this method is that it requires training many models. If there are M features, as many as $M(M-1)/2$ models must be trained. This can be prohibitively expensive with learning methods such as neural networks.

Training an artificial neural net with even one fully-connected hidden layer of modest size typically takes about 1-100 minutes on a modern workstation for problems with 10-1000 input dimensions and 1k-100k training cases. Assuming it takes 10 minutes on average to train a neural net on a problem with 100 features, forward stepwise selection will run for 34 days if most of the 100 features are needed, and will still run for 13 days if only 20 of the 100 features are needed. We recently attempted forward stepwise feature selection on a problem with 240 input features and were only able to select the first 30 features in 2 months of computation on a small cluster of computers working in parallel.

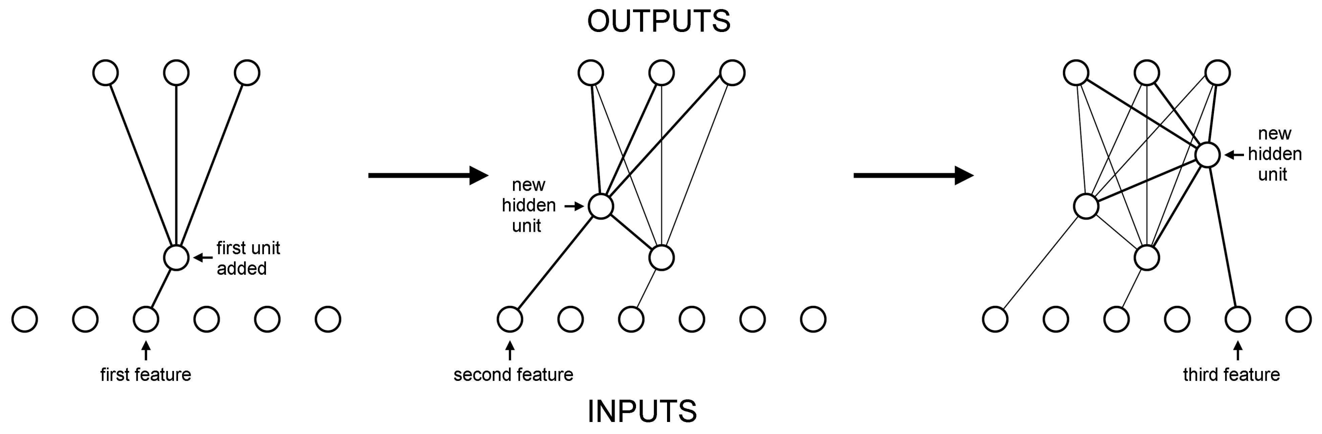
Heuristic methods other than stepwise selection have been used to try to determine the most relevant features for neural network learning [Leray and Gallinari, 1998]. Some of these methods attempt to determine which features to use before training the network [Setiono and Liu, 1996]. Other methods modify the neural network after it has been trained using all features [van de Laar *et al.*, 1999][LeCun *et al.*, 1990]. While these heuristic methods usually yield better models than those trained using all features, they often are not as effective as wrapper-based feature selection at finding small, near-optimal subsets of features.

In this paper we present C2FS, an *internal wrapper* feature selection method that combines forward stepwise selection with the Cascade Correlation (C2)[Fahlman and Lebiere, 1990] method for incrementally constructing networks. Nets trained with C2FS usually generalize better than C2 nets trained with all features, and train much faster than using a wrapper method with the C2 algorithm. On five test problems with 54-240 input features, C2FS needed less than 10% of the available features. Moreover, C2FS found significantly smaller feature sets than could be obtained with an information theoretic ordering of the attributes using gain ratio.

2. The C2FS Algorithm

2.1 The Cascade Correlation Algorithm

The Cascade Correlation (C2) algorithm [Fahlman and Lebiere, 1990] is an alternative to the more common fixed topography networks that are commonly used. In a C2 neu-



1: C2FS Network Topography. The first frame shows the state of the network after a single hidden unit has been added, with the inputs on the bottom, and the outputs on the top. The next two frames then show the network after the second and third hidden units have been added, respectively. Notice that each time a new hidden unit is added, a feature is added also.

ral network, there is a layer for the inputs, and a layer for the outputs. However, rather than a fixed number of hidden layers and hidden units, hidden units are added one at a time to the network, each hidden unit forming a new layer in the growing cascade architecture. Each hidden unit is given an incoming edge from each input node and each previous hidden unit, and an outgoing edge to each output node. After each hidden unit is added, a round of training occurs during which the edge weights are adjusted.

2.2 Combining C2 with FS

In previous work Schetinin [Schetinin, 2003] combined feature selection with C2 by ordering features in an initialization step, and then adding a single feature and a single hidden unit during each iteration of C2. In Schetinin's work, features are ordered by training univariate neural nets with single input features and then sorting by performance. During each iteration of the their algorithm, a single hidden unit and input feature is added, and a new round of training occurs. The new hidden unit is discarded if it does not improve performance.

C2FS is similar in that it adds a new hidden unit, and potentially uses a new feature, at each iteration. However, the forward stepwise selection C2FS uses to determine which features to add differs considerably from the heuristic used by Schetinin. It is based upon a combination of forward feature selection and the C2 network topography. Instead of using an external wrapper, however, the feature selection step is moved inside the training algorithm, creating an 'internal wrapper' method.

Pseudocode in Algorithm 1 presents a high-level version of the C2FS algorithm. It starts with a totally unconnected neural network. The network contains **INS** nodes for the inputs, and **OUTS** nodes for the outputs, but no hidden units, or connections between nodes. During the first iteration of C2FS, a single hidden unit is added, along with an edge from a single input node to that hidden unit and edges from the hidden unit to each of the output nodes. Next, the edge weights are trained using standard backpropagation of errors. Backpropagation is done on the training set, and the early stopping set is used to determine when to stop training the edge weights. The C2FS algorithm repeats this process for each of the **INS** inputs independently, training **INS** *separate* networks. From these networks, C2FS selects the one that gives the lowest RMSE on the feature selection set, and discards the rest.

In subsequent iterations, C2FS continues to add more hidden units, and potentially more input features, to the C2 network. It starts each iteration with the network selected from the previous iteration, preserving the edge weights trained during that iteration. C2FS then repeats the process of training **INS** networks, each of which has a single additional hidden unit with an incoming edge from one input node, incoming edges from all previous hidden nodes, and outgoing edges to all output nodes. As before, C2FS selects the network that performs best on the feature selection set after training, discards the inferior networks, and moves on to the next iteration. In each iteration, C2FS can add connections from a new input feature. However, C2FS may also add new hidden units without using new features – it is possible for new hidden units to connect only to previous hidden units and input features. Because of this, growing a large C2FS net does not require using many input features.

Like most neural net feature selection methods, C2FS needs data for training, regularization (early stopping), and feature selection. Because dividing the train data into three distinct sets would often make each subset quite small, feature selection may be done using the same data used for early stopping. In our experiments, we use the same data for early stopping and feature selection.

3. Data Sets

To evaluate the performance of C2FS, we selected 5 problems that each have a moderately large number of features. Two of the problems are from the UC Irvine machine learning repository [Blake and Merz, 1998], two were used in the 2004 KDD-CUP [Caruana and Joachims, 2004], and the fifth

```
initialize network prev
for k = 0 to MAX_HIDDEN
  network best = NULL
  foreach input i
    network new = prev with an additional hidden unit added
    add edge from node i to the new hidden unit
    add edges from each hidden unit in main to the new hidden unit
    add edges from the new hidden unit to each output node
    train(new)
    if(best == NULL or RMSE(new) < RMSE(best))
      best = new
    end
  end
  prev = best
end
```

Algorithm 1: The C2FS Algorithm Combines Feature Selection with the C2 Algorithm's Candidate Unit Installation Step.

is a medical risk prediction data set we are working with.

DNA Splice-Junction

This is a problem from the UCI repository. The data set contains 2000 cases, which we randomly split into training, early stopping, and test sets containing 667, 666 and 667 cases, respectively. We repeatedly sample the data set this way to generate 10 trials. The problem consists of 60 inputs, each representing a single nucleotide, and 3 output classes. We encode each of the 60 inputs, which have values of A, C, G and T, as 4 boolean inputs, exactly one of which is true. Similarly, we encode the 3 possible output classes (IE, EI and neither) as 3 boolean outputs. In this data set, 25% are IE, 25% are EI, and the remaining 50% are neither IE nor EI.

Forest Cover Type

This problem is also from the UCI repository, and contains a total of 581012 test cases. To be consistent with prior work [Caruana *et al.*, 2004], we use training sets containing only 4000 samples, and use only 1000 samples for the early stopping sets (which also is used for feature selection). Each case has 54 inputs and 7 boolean outputs, where each output represents a single forest cover type. Prior to running any experiments with this data set, we scale each input so that the largest input value is 1, and the smallest input value is 0.

Protein Homology

This problem, from the 2004 KDD-CUP, requires that a model predict whether or not two proteins are homologous based upon 74 comparisons between the two proteins. Each test case is formed by the comparison of two proteins and there is a single output which is 1 if the two proteins are homologous, and 0 otherwise. The data set has 285,409 test cases, only 2608 (0.91%) of which are homologous proteins. Again, we scale all of the inputs so they are between 0 and 1. We then randomly generate training, early stopping, and test sets with sizes 70,000, 70,000, and 145,409, respectively.

Quantum Physics

This is the second problem from the 2004 KDD-CUP, and contains 78 inputs, with a single boolean output. We use training, early stopping and test sets with 4000, 1000, and 5000 cases, respectively, which were randomly drawn from the full data set. This problem is well balanced, with 49% positive class, and 51% negative class.

Pneumonia

This problem requires models to predict the probability of a poor outcome (e.g., cardiac failure) in a pneumonia patient based on 192 different measures of patient health (blood pressure, temperature, etc.). There are a total of 2287 test cases, 261 (11.4%) of which have poor outcomes. Our training, early stopping and test sets contain 762, 762, and 763 cases respectively.

4. Experimental Results

We ran 10 trials using C2 and the new feature selection algorithm C2FS on each of the five problems. We compare the performance of C2 and C2FS by looking at the squared error and accuracy of the models on the final test sets. We also examine how many features are used by the feature selection models.

4.1 C2

For each problem we ran 10 trials with C2. After C2 inserts each new hidden unit, the network is trained until performance on the early stopping set stops improving, at which point a new hidden unit is added. With C2, each new hidden unit is connected to all input features.

4.2 C2FS

The C2FS algorithm was run using the same train, early stopping (also used for feature selection), and final test sets as C2. For each problem set, we used the same learning rates and other parameters for both C2 and C2FS.

4.3 Comparison of C2 and C2FS

As shown in Table 1 and Table 2, C2FS performs significantly better than C2 on 3 of the 5 problems (DNA Splice Junction, Forest Cover Type, and Quantum Physics) with $p < 0.01$. On the Pneumonia problem C2FS performs slightly better than C2, but not significantly ($p = 0.187$).

1: Comparison of C2 and C2FS using t-Test for 10 trials. Each value is the average over 10 independent trials. Bold entries indicate differences that are different at $p = 0.05$. p values are also shown.

Data Set	Root-Mean-Squared-Error			ACCURACY		
	C2	C2FS	p-value	C2	C2FS	p-value
DNA Splice-Junction	0.2120	**0.1667**	0.000	0.9187	**0.9525**	0.000
Forest Cover Type	0.2249	**0.2218**	0.004	0.7585	**0.7680**	0.012
Quantum Physics	0.4374	**0.4319**	0.001	0.7040	0.7058	0.500
Pneumonia	0.2987	0.2920	0.187	0.8898	0.8932	0.358
Protein Homology	0.0489	0.0499	0.132	0.9974	0.9972	0.094

2: Comparison of C2 and C2FS using Sign Test. Columns show how many of the 10 trials each model performed best on. Again bold entries show differences that are significant at $p = 0.05$.

Data Set	Root-Mean-Squared-Error			ACCURACY		
	C2	C2FS	p-value	C2	C2FS	p-value
DNA Splice-Junction	0	**10**	0.002	0	**10**	0.002
Forest Cover Type	1	**9**	0.021	1	**9**	0.021
Quantum Physics	0	**10**	0.002	4	6	0.754
Pneumonia	3	7	0.344	5	5	1.000
Protein Homology	8	2	0.109	8	2	0.109
Sum over all trials	12	38		18	32	

On the Protein Homology problem, C2FS performs slightly worse than C2, but again not significantly ($p = 0.132$).

The pneumonia problem presents an interesting challenge for C2FS. The data set is small (only 762 cases in the train and early stopping sets), there are many features (192), and the problem is imbalanced (only 11.4% positive class). Because of this it is relatively easy to overfit by picking features that improve performance on the training and feature selection sets, but which do not help on the independent test data.

Figure 2 shows the RMSE of the models on the final test sets as hidden units are added. Each plot shows the average across the 10 trials. Lower RMSE indicates better performance. Notice that C2 sometimes does better than C2FS when there are only a few hidden units because it is able to use all of the features from the beginning. Since C2FS can use no more features than it has hidden units, it must always add a few hidden units before it can do as well as C2.

On average, C2FS yields moderate performance gains when compared to C2. More impressively though, it does so using only 5–15 of the 54–240 features. The mean number of features used by the C2FS nets are summarized in Table 4. On average, C2FS uses less than 10% of the available features.

4.4 Comparing C2FS to Feature Selection Using Gain Ratio

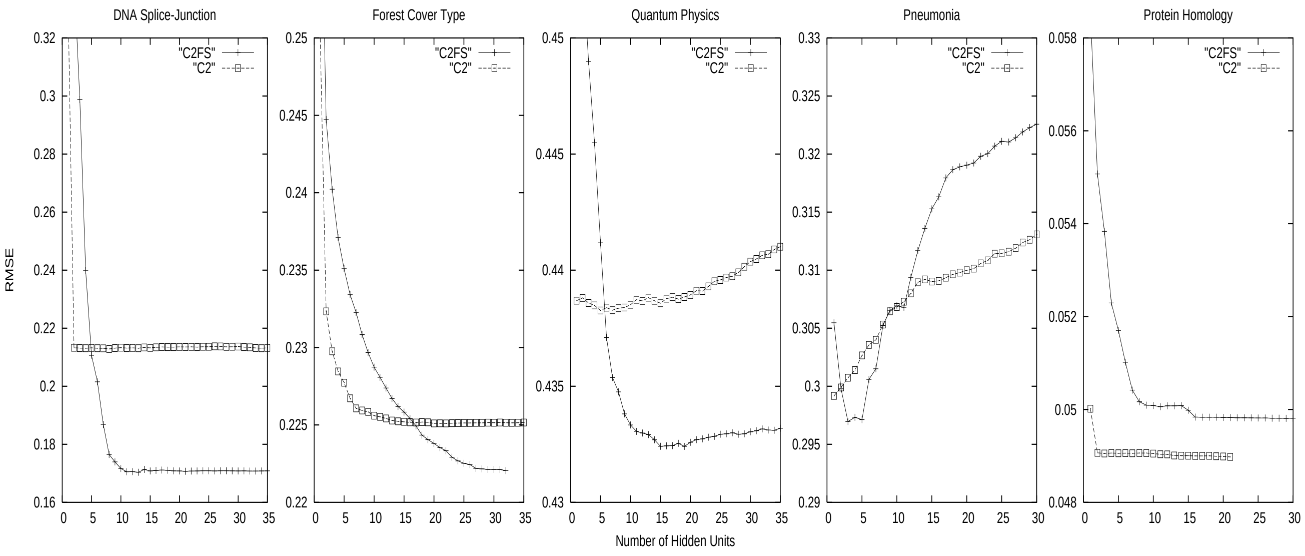
The results in the previous section indicate that C2FS does a good job at training C2 nets that need few features. On average, C2FS uses less than 10% of the available features, and yet outperforms C2 nets trained on all features on 4 (only 3 significantly) of the 5 test problems.

We also wanted to compare the features selected by C2FS with features selected using a simpler method: ordering features by gain ratio. We ran a second set of experiments using essentially the C2FS algorithm, but rather than picking the features with the *internal wrapper* method discussed previously, we used gain ratio to order the features. This results in an algorithm very similar to Schetinin's method [Schetinin, 2003] because ordering features by gain ratio yields a very similar ordering to Schetinin's method of sorting features by performance in univariate models. It differs from their algorithm in that we add features (and hidden units) according to the ordering, regardless of their impact on performance, while [Schetinin, 2003] only adds features that improve performance. In practice, however, this makes little difference – in our experiments adding a few new features to C2FS nets rarely causes performance to decrease.

Gain ratio is commonly used as a splitting criterion in decision tree induction. It is defined whenever a data set is split into two or more subsets. The more a split helps create subsets with homogeneous classes, the better the gain ratio. Gain

4: Mean number of features required by C2FS and C2FSGR. Each value is the average number of features required when the RMSE of the model stops improving. Models are evaluated separately in each trial, and then the mean is computed over the 10 trials.

Data Set	Tot # Feats	C2FS	C2FSGR	p-value
DNA	240	**14.1**	49.7	0.000
Forest	54	**21.8**	37.1	0.000
Quantum	78	**7.4**	52.6	0.000
Pneumonia	192	**2.9**	17.9	0.000
Protein	74	**6.9**	41.7	0.000
Mean	127.6	10.62	39.8	



2: Each graphs shows the comparison of the C2 and C2FS architectures on one data set. The graphs are all generated by taking the average RMSE over 10 independent trials.

3: Comparison of C2FS and C2FSGR using t-Test over 10 independent trials.

Data Set	Root-Mean-Squared-Error			ACCURACY		
	C2FSGR	C2FS	p-value	C2FSGR	C2FS	p-value
DNA Splice-Junction	0.1672	0.1667	0.956	0.9526	0.9525	0.806
Forest Cover Type	0.2254	**0.2218**	0.003	0.7564	**0.7680**	0.000
Quantum Physics	0.4322	0.4319	0.123	0.7048	0.7058	0.371
Pneumonia	0.2883	0.2920	0.447	0.8932	0.8978	0.215
Protein Homology	0.0506	0.0499	0.305	0.9972	0.9972	0.611

ratio is information gain divided by a correction term to compensate for split arity that eliminates the bias of information gain toward high-arity splits[Quinlan, 1986].

$$GainRatio(S,A) = \frac{Gain(S,A)}{\sum_{i\in A} -lg(\frac{p_i}{N})}$$

$$Gain(S,A) = Entropy(S) - \sum_{i\in A} p_i Entropy(S_i)$$

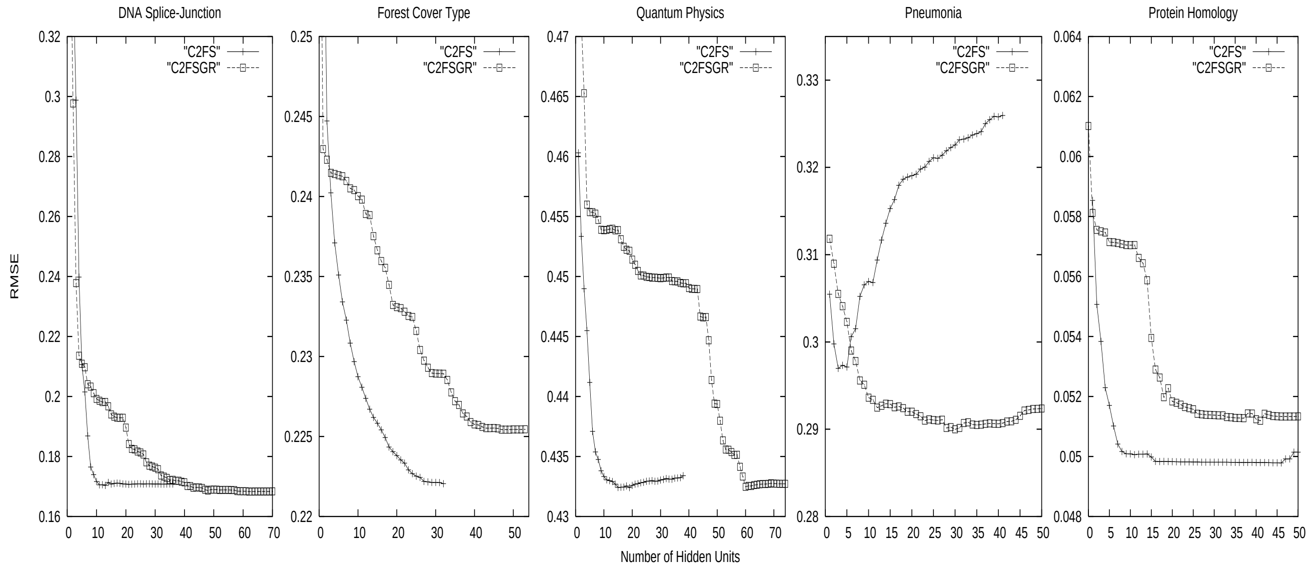
where S_i represents the the i^{th} subset of the split, p_i represents the fraction of the data in S that goes into S_i, and A is the set of values for the attribute. Most of our attributes are low arity, or continuous (on which we do binary splits), so the difference between gain ratio and information gain is not that significant for these data sets.

For each of the 10 trials, we calculated the gain ratio of each feature on the union of the training and early stopping sets. For each variable, we considered all possible split points, and chose the best one. We then ran the C2FS algorithm, but instead of trying each feature during each iteration and picking the best one, we simply installed the next feature as dictated by the gain ratio ordering.

As shown in Table 3, C2FS does significantly better than C2FSGR on one of the problems – Forest Cover Type. On the other four problems it is not significantly different from C2FSGR. Figure 3 shows the average RMSE performance over the ten trials for C2FS and C2FSGR on the five problems. In every graph, C2FS improves faster than C2FSGR when there are few features, and then levels off earlier. This indicates that, in each case, it is picking the most important features better than C2FSGR.

Table 4 shows the average number of features used by models trained with C2FS and C2FSGR. The graphs in Figure 3 plateau as the number of features increases. To calculate the statistics in Table 4, we look at each of the 10 trials independently, and find the smallest number of features that yields peak performance. As the table shows, C2FS reaches peak performance using four times fewer features on average than C2FSGR. From the results in Tables 3 and 4 and Figure 3 we conclude that on average C2FS yields generalization performance that is somewhat better than C2FSGR, and is able to yield this performance using one-fourth as many features.

On the protein homology problem (the only problem where C2FS is not better than C2), C2FSGR also performs worse than C2. Comparing C2FS to C2FSGR on the protein homology problem, we see that C2FS does better. This leads us to believe that C2FS is learning a good ordering on the features for this problem, but for some reason this ordering



3: Each graphs shows the comparison of the C2FSGR and C2FS methodologies for the various data sets. Each graph is generated by taking the average RMSE over 10 independent trials.

is not yielding improvements in performance over C2 models trained with all input features.

5. Conclusion

C2FS is an *internal wrapper* feature selection method that adds feature selection to the candidate (hidden) unit training procedure of the C2 algorithm. This internal wrapper approach is 2-3 orders of magnitude more efficient than external wrapper methods for feature selection. Without an approach like this, wrapper methods on neural nets are infeasible. On 3 of 5 test problems, C2FS clearly outperforms C2 nets trained with all available features, and yields performance comparable to C2 on the other 2 problems. On average, across the five test problems, nets trained with C2FS use less than 10% of the available features. When compared to an alternate approach for combining feature selection with C2 nets – ordering features by gain ratio – C2FS still yields somewhat better generalization performance, and is able to find nets that use one fourth as many features. On the one problem where C2FS does not perform better than C2 using all features (the protein homology problem), C2FS still outperforms C2FSGR.

6. REFERENCES

[Blake and Merz, 1998] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[Caruana and Joachims, 2004] R. Caruana and T. Joachims. Kdd cup 2004. urlhttp://kodiak.cs.cornell.edu/kddcup, 2004.

[Caruana *et al.*, 2004] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 18, New York, NY, USA, 2004.

[Fahlman and Lebiere, 1990] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, Denver 1989, 1990. Morgan Kaufmann, San Mateo.

[Kittler, 1978] J Kittler. Une generalisation de quelques algorithms sous-optimaux de recherche d'ensembles d'attributs. In *Proc. Congr'es Reconnaissance des Formes et Traitement des Images*, 1978.

[Kohavi and John, 1997] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[LeCun *et al.*, 1990] Y. LeCun, J. Denker, S. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems II*, San Mateo, CA, 1990. Morgan Kauffman.

[Leray and Gallinari, 1998] P. Leray and P. Gallinari. Feature selection with neural networks. *Behaviormetrika*, 26, January 1998.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Schetinin, 2003] V. Schetinin. A learning algorithm for evolving cascade neural networks. 17:21–31, 2003.

[Setiono and Liu, 1996] Rudy Setiono and Huan Liu. Improving backpropagation learning with feature selection. *Applied Intelligence*, 6(2):129–139, 1996.

[van de Laar *et al.*, 1999] P. van de Laar, T. Heskes, and S. Gielen. Partial retraining: A new approach to input relevance determination. *International Journal of Neural Systems*, 9:75–85, 1999.