Model Compression

Cristian Bucilă
Computer Science
Cornell University
cristi@cs.cornell.edu

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

Alexandru Niculescu-Mizil
Computer Science
Cornell University
alexn@cs.cornell.edu

ABSTRACT

Often the best performing supervised learning models are ensembles of hundreds or thousands of base-level classifiers. Unfortunately, the space required to store this many classifiers, and the time required to execute them at run-time, prohibits their use in applications where test sets are large (e.g. Google), where storage space is at a premium (e.g. PDAs), and where computational power is limited (e.g. hearing aids). We present a method for "compressing" large, complex ensembles into smaller, faster models, usually without significant loss in performance.

Categories and Subject Descriptors: I.5.1 [Pattern Recognition]: Models – Neural nets.

General Terms: Algorithms, Experimentation, Measurement, Performance, Reliability.

Keywords: Supervised Learning, Model Compression

1. INTRODUCTION

An ensemble is a collection of models whose predictions are combined by weighted averaging or voting. Ensemble methods have been the focus of significant research in the past decade, and a variety of ensemble methods have been introduced. Well known ensemble methods include bagging [2], boosting [14], random forests[3], Bayesian averaging [9] and stacking [17]. Much of the interest in ensemble methods has been fueled by their excellent empirical performance.

Ensembles, however, have one disadvantage that often is overlooked: many ensembles are large and slow. This makes ensemble methods unusable for applications with limited memory, storage space, or computational power such as portable devices or sensor networks, and for applications in which real-time predictions are needed. Consider, for example, boosted decision trees, bagged decision trees or random forests. These models often contain hundreds or thousands of decision trees, each of which must be stored, and executed at run-time to make predictions. Executing a single tree is fast, but executing a thousand trees is not.

KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.

In this paper we show how to compress the function that is learned by a complex model into a much smaller, faster model that has comparable performance. Specifically, we show how to train compact artificial neural nets to *mimic* the function learned by ensemble selection, an ensemble learning method introduced by Caruana et al. [5]. To achieve this, we take advantage of the well known property of artificial neural nets, namely that they are universal approximators: given enough training data, and a large enough hidden layer, a neural net can approximate any function to arbitrary precision. Instead of training the neural net on the original (often small) training set used to train the ensemble, we use the ensemble to label a large unlabeled data set and then train the neural net on this much larger, ensemble labeled, data set. This yields a neural net that makes predictions similar to the ensemble, and which performs much better than a neural net trained on the original training set.

The key difficulty when compressing complex ensembles into simpler models this way is the need for a large unlabeled data set. In some domains, unlabeled data is easy to obtain. In other domains, however, large data sets (labeled or unlabeled) are not available. In these domains, we generate synthetic cases that as closely as possible match the distribution of the original training set. We introduce a new method for generating synthetic cases called MUNGE that outperforms other methods to which we have compared it. Using MUNGE, we are able to train neural nets that are a thousand times smaller and faster than ensemble selection ensembles, but which have nearly the same performance as the far more complex ensembles.

2. MODEL COMPRESSION

In some situations, it is not enough for a classifier or regressor to be highly accurate, it also has to meet stringent time and space requirements. In many cases, however, the best performing model is too slow and too large to meet these requirements, while fast and compact models are less accurate, because either they are not expressive enough, or they overfit to the limited training data. For such situations, we propose using *model compression* to obtain fast, compact yet highly accurate models.

The main idea behind model compression is to use a fast and compact model to approximate the function learned by a slower, larger, but better performing model. Unlike the true function that is unknown, the function learned by a high performing model is available and can be used to label large amounts of pseudo data. A fast, compact and expressive model trained on enough pseudo data will not overfit

and will approximate well the function learned by the high performing model. This allows a slow, complex model such as a massive ensemble to be compressed into a fast, compact model such as a neural net with little loss in performance.

An important question is how do we get the pseudo data. In some domains large amounts of unlabeled data is easy to collect (e.g. in text, web and image domains) and can be used as pseudo data. In other domains, however, unlabeled data is not readily available and synthetic cases need to be generated. This is more difficult than it might seem at first. It is important that the synthetic data match well the distribution of the real train and future test cases. Usually real data lay in a small submanifold of the complete attribute space. If the synthetic data is drawn from a distribution that has little overlap to this manifold, the labeled synthetic points will fail to capture the target function in the region of interest. On the other hand, if the distribution from which the synthetic data is sampled is too broad, only a fraction of the points will be drawn from the true manifold and many more samples will be necessary to adequately sample the region of interest. The best case is when the synthetic distribution is very similar to the true distribution. Then a minimum number of samples will be necessary to adequately sample the target function.

We experiment with three methods of generating pseudo data: RANDOM, generate data for each attribute independently from its marginal distribution; NBE, estimate the joint density of attributes using the Naive Bayes Estimation algorithm [12] and then generate samples from this joint distribution; and MUNGE, a new procedure we propose that samples from a non-parametric estimate of the joint density.

2.1 RANDOM

The simplest way to generate pseudo data is to independently sample the value of each attribute from the marginal distribution of that attribute. This is the procedure predominantly used in the literature whenever there is a need for artificial data (e.g. [13, 6]). Usually the nominal attributes are generated from a multinomial distribution whose parameters are estimated from the training data. The continuous attributes are usually modeled using a uniform distribution, a Gaussian distribution with mean and variance estimated from the training set, or via kernel density estimation [15].

The RANDOM method for generating pseudo data uses a nonparametric bootstrap approach. For each attribute, a value is selected uniformly at random from the multiset (bag) of all values for that attribute present in the train set.[1]

When the attribute values are generated independently, all conditional structure is lost and the pseudo examples are generated from a distribution that is usually much broader than the true distribution of the data. As a consequence many of the generated pseudo examples will cover uninteresting parts of the space, and this may prevent the mimic model from focusing on the important regions.

2.2 NBE

Another approach to generating pseudo data is to estimate the joint distribution of attributes using the training set, then sample pseudo examples from this joint distribution. Assuming that the true joint distribution can be estimated well, the conditional structure of the domain would be preserved and the new artificial examples would cover well the interesting regions of the space.

One way to estimate the joint distribution of a set of variables is to use mixture model algorithms. These algorithms model the data as coming from a mixture of components, each component with a different distribution. The most well known algorithm in this category, used in domains with only continuous attributes is the mixture of Gaussians [7], where each component consists of a Gaussian distribution with a different mean and covariance matrix.

A mixture model algorithm that handles both discrete and continuous attributes, NBE (Naive Bayes Estimation), was recently introduced by Lowd and Domingos [12]. We used NBE to estimate the joint distribution of the attributes because it handles mixed attributes, it is simple to use, it performs as well as learning a Bayesian Network from the same data [12], and it is readily available.

2.3 MUNGE

Estimating a full joint distribution is difficult when there are many attributes and few training cases. Instead of trying to reliably estimate a joint distribution, we have developed a new algorithm that samples directly from a non-parametric estimate of the joint distribution.

Algorithm 1 MUNGE

Require: set of training examples T, size multiplier k, probability parameter p, local variance parameter s
Returns: unlabeled training set D of size $k \times size(T)$

$D \leftarrow \emptyset$
loop k times
  $T' \leftarrow T$
  for all examples e in T' **do**
    $e' \leftarrow$ the closest example of e from T'
    for all attributes a of example e (excluding the label attribute) **do**
      if a is continuous **then**
        with probability p: $e_a \leftarrow norm(e'_a, sd)$, and $e'_a \leftarrow norm(e_a, sd)$, where $sd \leftarrow |e_a - e'_a|/s$, and $norm(a, b)$ is a random value taken from the normal distribution with mean a and standard deviation b.
      else
        with probability p: swap the values of attribute a for examples e and e'
      end if
    end for
  end for
  $D \leftarrow D \bigcup T'$
end loop
Return D

Starting from the original training set, we visit each example once and determine its closest neighbor. To measure distance between cases, we use euclidean distance for continuous attributes and hamming distance for nominal attributes. Continuous attributes are linearly scaled to [0,1].

Given example e and its closest other example e', the values for each noncontinuous attribute are swapped between e and e' with probability p and are left unchanged with probability $1 - p$. For each continuous attribute a, with

[1]For nominal attributes this is equivalent to generating the values from the multinomial distribution. For continuous attributes this procedure is slightly different than previously proposed ones, but generates similar values in practice.

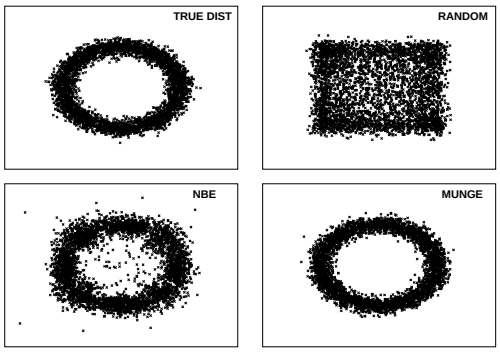

Figure 1: Synthetic data generated for a simple 2D problem.

probability p, e_a is assigned a random value drawn from a normal distribution with mean e'_a and standard deviation $sd = |e_a - e'_a|/s$, and e'_a is assigned a random value drawn from the normal distribution with mean e_a and the same standard deviation sd. We call this approach to generating artificial data by swapping values between neighboring cases *MUNGE*.[2] The method is presented in Algorithm 1.

Figure 1 shows samples generated from a simple 2D distribution (TRUE DIST), and the distributions learned by RANDOM, NBE and MUNGE from a train set of 4000 points drawn from TRUE DIST. As expected, the samples generated by RANDOM cover an area much larger than the true distribution, so only relatively few of the samples overlap with the region of interest. NBE does a better job at approximating the true distribution, but still has problems, especially in the "corners". Of the three methods, MUNGE clearly approximates the true distribution the best.

3. EXPERIMENTAL EVALUATION

We evaluate the effectiveness of model compression on eight binary classification problems. ADULT, COVTYPE and LETTER are from the UCI Repository [1]. COVTYPE has been converted to a binary problem by treating the largest class as positive and the rest as negative. We converted LETTER to a binary problem in two ways. LETTER.p1 treats the letter "O" as positive and the remaining 25 letters as negative, yielding a very unbalanced binary problem. LETTER.p2 uses letters A-M as positives and the rest as negatives, yielding a difficult, but well balanced, problem. HS is the IndianPine92 data set [10] where the difficult class Soybean-mintill is the positive class. SLAC is a problem from the Stanford Linear Accelerator. MEDIS and MG are medical data sets. See Table 1 for characteristics of these problems.

3.1 Experimental Setup

We experiment with using neural networks to compress the models built using the ensemble selection algorithm proposed by Caruana et al. in [5]. The ensemble models generated by ensemble selection are very large, complex models that have very good generalization performance, thus they are a perfect candidate for model compression.

[2]The dictionary defines munge as *"To imperfectly transform information"* or *"To modify data in a way that cannot be described succinctly"*.

Table 1: Description of problems.

PROBLEM	#ATTR	TRAIN SIZE	TEST SIZE	%POZ
ADULT	14/104	4000	35222	25%
COVTYPE	54	4000	25000	36%
HS	200	4000	4366	24%
LETTER.P1	16	4000	14000	3%
LETTER.P2	16	4000	14000	53%
MEDIS	63	4000	8199	11%
MG	124	4000	12807	17%
SLAC	59	4000	25000	50%

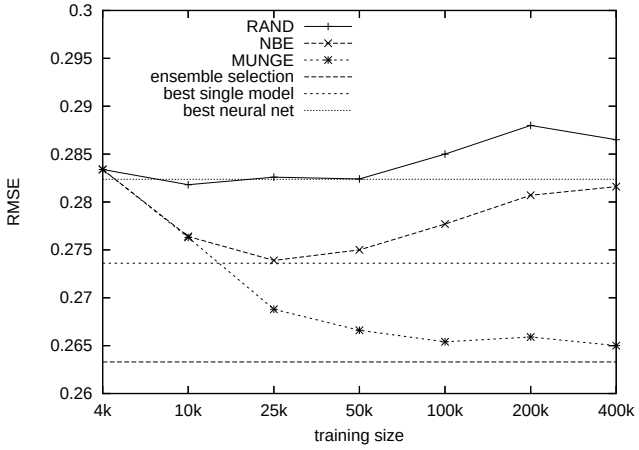

Figure 2: Average perf. over the eight problems.

Ensemble selection first builds a library of diverse base-level modes using many different learning algorithms and parameter settings. After the library is built, the basic ensemble selection procedure builds the ensemble model by greedily selecting at each iteration the model from the library that when added to the ensemble improves the performance of the ensemble the most. Caruana et al. also propose a number of enhancements to the basic ensemble selection algorithm that improve its performance, but as a side effect increase the size of the ensemble by increasing the number of base-level models it contains.

For all problems, a training set of 4000 points is used to train the base-level models, and a validation set of 1000 points is used as hill climb set for ensemble selection. For compression, the 4000 training points are used as a training set for the three algorithms for producing artificial data: RANDOM, NBE and MUNGE. The artificial data generated with each algorithm is then labeled by the ensemble model and used to train a neural net model that will mimic the ensemble. When necessary, the 1000 points validation set is used for early stopping.

We compare the performance of the compressed models with the performance of the target ensemble selection models on the eight test problems. We also show the performance of the best single base-level model from the ensemble selection library, selected using the same 1000 points validation sets, and the best neural network that could be trained on the original 4000 points training sets, using the 1000 points validation sets for early stopping and for selecting the number of hidden units. All the reported results reflect the root-mean-squared-error (RMSE) of models predictions to the binary 0/1 targets on large independent final test sets.

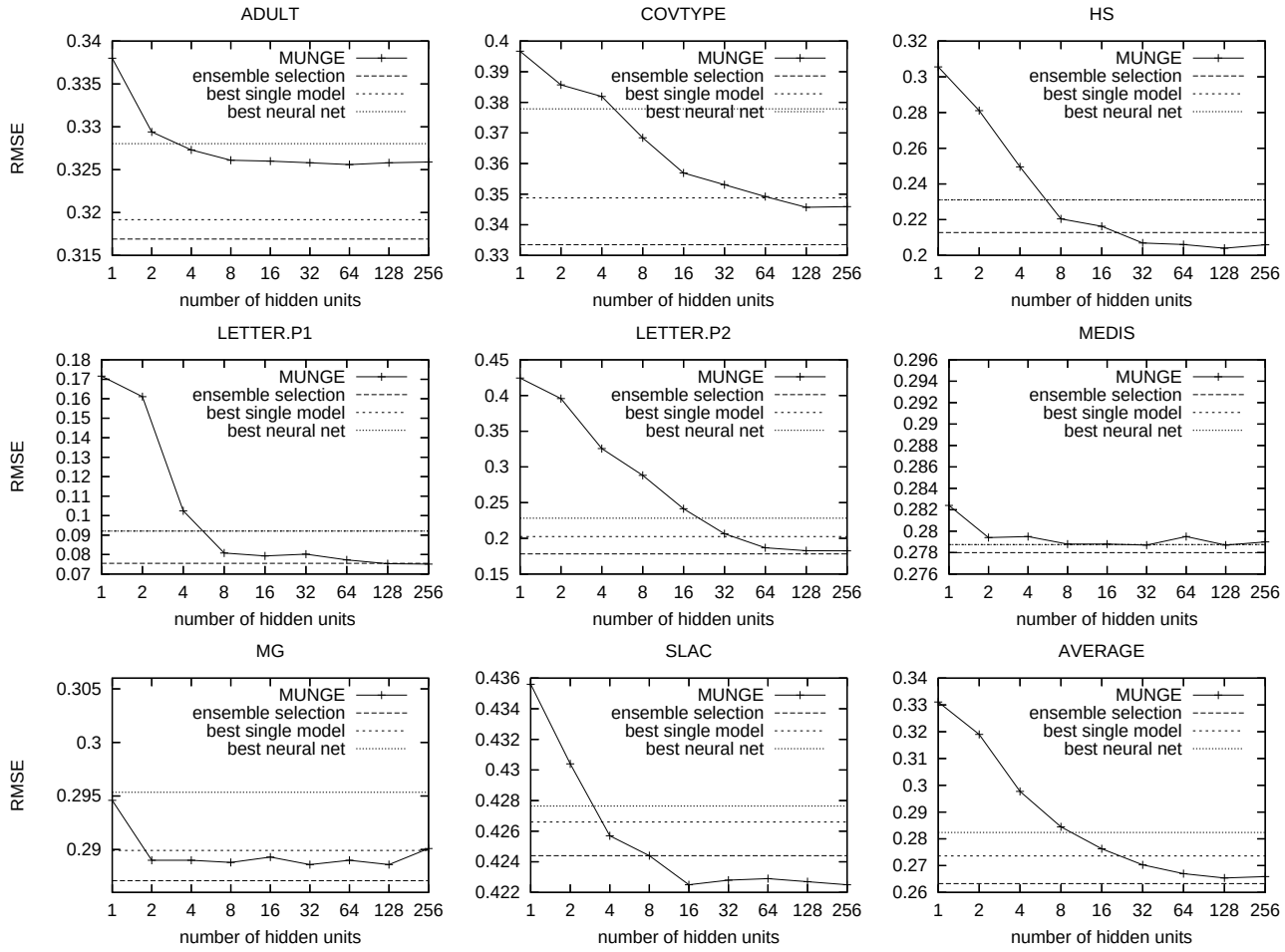

Figure 3: Performance of compressed models vs compressed models complexity.

3.2 Results

Figure 2 shows the average RMSE performance on the eight test problems. Lower RMSE represents better performance. The top horizontal line in the figure shows the performance of the best neural nets we could train on the original 4k train set. The bottom horizontal line shows the performance of ensemble selection trained on the same data. Note that the models trained with ensemble selection perform considerably better than the neural net models. The horizontal line in the middle is the average performance of the best single base-level models from the ensemble selection libraries, before an ensemble has been selected. This line represents the best performance we could achieve with any of the following learning methods: SVMs, bagged trees, boosted trees, boosted stumps, simple decision trees, random forests, neural nets, logistic regression, k-nearest neighbor, and naive Bayes.

The other lines in Figure 2 show the performance of mimic neural nets trained on different amounts of pseudo data labeled by the ensemble models. The lines for RANDOM, NBE, and MUNGE correspond to neural nets with 128 hidden units trained on pseudo data generated using the three methods described in Section 2. The graph starts at 4k where there is no pseudo data added to the train sets. Because of this, the performance of all three methods is similar to the performance of the best neural nets we could train on the original train set. Performance of the mimic nets is slightly worse because they are restricted to using only 128 hidden units, which is not always optimal.

As the size of the train data increases beyond 4k, more pseudo data is being added to the train sets. At 400k, the train set contains 396k of pseudo data and the original 4k train data. For mimic nets trained with pseudo data generated by RANDOM, performance improves slightly at 10k, and at 100k and beyond performs worse than a neural net trained on just the original 4k train set. The mimic neural nets trained on pseudo data generated by NBE perform better, though the overall pattern is similar to the graph for RANDOM. The peak performance of the NBE trained nets occurs when the train set contains about 20k of pseudo data and 4k of the original data, then degrades as more artificial data is added to the train set.

The mimic neural nets trained on pseudo data generated with MUNGE dominate the nets trained with RANDOM and NBE data. Moreover, the performance with MUNGE does not degrade as more data is added to the pseudo train set. On average, once the pseudo training set contains 100k or more data, the mimic neural nets perform considerably better than the best individual models in the ensemble libraries, and nearly as well as the target ensemble itself. This is remarkable given that the mimic neural nets are 100-100,000 times smaller than the ensembles, and 100 to 10,000

Table 2: RMSE results.

	MUNGE	ENSEMBLE	ANN	SINGLE	RATIO
ADULT	0.325	0.317	0.328	0.319	0.29
COVTYPE	0.340	0.334	0.378	0.349	0.84
HS	0.204	0.213	0.231	0.231	1.47
LETTER.P1	0.075	0.075	0.092	0.092	1.01
LETTER.P2	0.179	0.178	0.228	0.203	0.98
MEDIS	0.277	0.278	0.279	0.279	2.29
MG	0.288	0.287	0.295	0.290	0.88
SLAC	0.422	0.424	0.428	0.427	1.69
AVERAGE	0.264	0.263	0.282	0.274	0.97

Table 3: Time in seconds to classify 10k cases.

	MUNGE	ENSEMBLE	ANN	SINGLE
ADULT	7.88	8560.61	3.94	48.31
COVTYPE	4.46	3440.99	1.05	37.31
HS	12.09	1817.17	3.85	3.85
LETTER.P1	2.59	1630.21	0.25	0.25
LETTER.P2	2.59	2651.95	0.74	526.34
MEDIS	4.78	190.18	2.85	2.85
MG	6.98	1220.04	1.80	53.58
SLAC	3.60	23659.03	2.85	74.48
AVERAGE	5.62	5396.27	2.17	93.37

times faster to execute. It suggests that much smaller high performing models are possible if we only knew how to train them on the original training data. In summary, it appears that MUNGE is the preferred method for generating pseudo data. In the experiments in the remainder of this section we will examine results for MUNGE only.

Figure 3 presents the performance of the mimic neural nets trained with MUNGE data as a function of the number of hidden units in the trained network. Performance is shown for all eight problems. The graph in the bottom right shows the average performance across all eight problems. All MUNGE neural nets are trained on pseudo data containing 100k samples (4k original data + 96k MUNGE data). The graphs also show the performance of the best neural nets trained on the original data, the performance of the best single models from the ensemble libraries and the performance of the target ensemble selection models.[3]

Looking at the graph that averages the eight problems, the overall trend is that performance improves until around 128 hidden units when it levels off. Looking at graphs for each individual problem, we see that for some of the problems a small number of hidden units is enough for obtaining good performance. For MEDIS and MG, performance does not improve if we use networks with more than 2 hidden units, and for SLAC 16 hidden units are enough. Since the training set is large enough to prevent overfitting, performance does not degrade as more hidden units are added.

The ADULT and COVTYPE problems are the only ones where there is a significant difference between the performance of the MUNGE neural nets and the performance of the ensemble selection models. A more detailed discussion of the results on these datasets will follow later in this section, and in Section 4.

Figures 2 and 3 show that, on average, performance of the mimic networks trained on the MUNGE data improves with more hidden units and more pseudo data. As a final experiment, we compress the ensemble selection models using 256 hidden unit nets and 400k MUNGE data for every problem. Table 2 shows, for each of the eight problems, the performance of mimic neural nets trained on the MUNGE data, the target ensemble selection model, the best neural net trained on the original data, and the best single model from the ensemble library. The performance of the mimic neural nets is as good as or better than the performance of the ensemble models they are trained to mimic on six of the eight problems, and always better than the performance of neural nets trained on the original 4k data.

The values in the last column of the table indicate how effective compression is at retaining the performance of the target ensemble selection models. These values are the ratio between the improvement in performance the mimic nets provide over the best neural nets and the improvement in performance the target ensemble selection models provide over the best neural nets. For example, if the mimic neural net has performance half way between the original neural net and the ensemble, the ratio is 0.5. If the mimic neural net has performance equal to the target ensemble, the ratio is 1.0. The only problem on which the ratio is less than 0.8 is ADULT. (The results on this problem are discussed in the next paragraph.) For a few problems the ratio is better than 1.0, indicating that the mimic neural net outperforms the ensemble. Note, however, that in two of the cases where the ratio is much larger than 1 (SLAC at 1.69 and MEDIS at 2.29), the range in performance is very small so this large ratio does not actually indicate a very large increase in performance. The ratio in the bottom row is the ratio calculated for the average RMSE performances in the table (not the average of the ratios, which would be inflated by the two problems with artificially high ratios). On average, model compression with MUNGE is able to achieve 97% of the performance increase that could at best be expected.

The only problem for which compression is ineffective is ADULT. On this problem the mimic net performs only a little better than a neural net trained on the original 4k data, and the mimic net does not perform as well as the best single model in the ensemble selection library. Interestingly, ADULT is the only data set that has high-arity nominal attributes. The three attributes with the highest arity have 14, 16, 41 unique values. To train a neural net on ADULT, these attributes must first be converted to 14, 16, and 41 distinct binary attributes. The ADULT problem has only 14 attributes to begin with, yet these three attributes alone expand to 71 sparsely coded binary inputs. It is possible that neural nets are not well suited to this kind of problem, and this may prevent the mimic neural net from learning the ensemble target function. An alternate possibility is that the MUNGE procedure is not effective at generating pseudo data for this kind of problem.

Table 3 shows the time in seconds required to classify 10,000 test cases for the mimic neural nets with 256 hidden units, the target ensemble models trained by ensemble selection, the best neural nets trained on the original 4k train set and the single best model in the ensemble library. There

[3]On HS, LETTER.p1 and MEDIS problems the best single model from the ensemble library is actually a neural net so the lines for best neural net and best single model overlap.

Table 4: Size of the models in MB.

	MUNGE	ENSEMBLE	ANN	SINGLE
ADULT	0.45	1234.72	0.22	3.95
COVTYPE	0.23	1108.16	0.03	3.41
HS	0.79	74.37	0.12	0.12
LETTER.P1	0.08	1.23	0.01	0.01
LETTER.P2	0.08	325.80	0.04	0.07
MEDIS	0.27	5.24	0.14	0.14
MG	0.50	25.75	0.03	3.25
SLAC	0.25	1627.08	0.13	0.30
AVERAGE	0.33	550.29	0.09	1.41

is significant variability in the speed of the best single model because different kinds of models are best for different problems and some of the models (e.g. boosted trees) are much more expensive than others (e.g. logistic regression). As expected, the ensemble is extremely expensive. On average, the ensemble takes about 0.5 seconds to classify a single training case (on a single workstation) and, on the SLAC problem, it takes 2.4 seconds per test case! The mimic neural nets, however, are very fast and take on average only about 0.5 milliseconds per test case.[4]

Table 4 shows the size in megabytes for the different models in Table 2. A similar picture emerges as with execution times: on average ensembles are about 500 megabytes, about 500 times larger than the best single models, and the largest ensembles are more than a gigabyte. The mimic neural nets, however, are four times smaller than the best single models, and more than 1000 times smaller than the ensembles.

4. DISCUSSION

The number of hidden units needed in the compression neural net to mimic the ensemble function provides an indication of how complex the function learned by the ensemble actually is. For some problems, only a few hidden units were required to learn the ensemble function with high fidelity. For other problems (e.g. COVTYPE) the target ensemble function is more complex, and requires using 128 hidden units or more. Note that one cannot always use the size of a neural net trained on the *original* training data as a measure of function complexity because of the interaction between overfitting and network size and train set size. With compression, however, we can make the train set size arbitrarily large, so that overfitting is not an issue, and thus reliably measure the effect of network size on performance.

Generating unlabeled pseudo data is not computationally expensive. Labeling large amounts of pseudo data, however, can be expensive if the target model is a large, complex ensemble. Training the neural net can also be expensive, particularly if the problem has many input features, requires a net with many hidden units, and/or requires a large pseudo training set. In the worst case, it can be more expensive

[4]The speed of different models depends significantly on how they are implemented. The times reported here are for typical implementations. For example, we use the SNNS neural net package [18], the IND decision tree package [4], the SVMlight SVM package [11], WEKA [16] random forests, etc. With care, some of these numbers probably could be improved by a factor of 10 or more, though we suspect the overall picture would not change substantially.

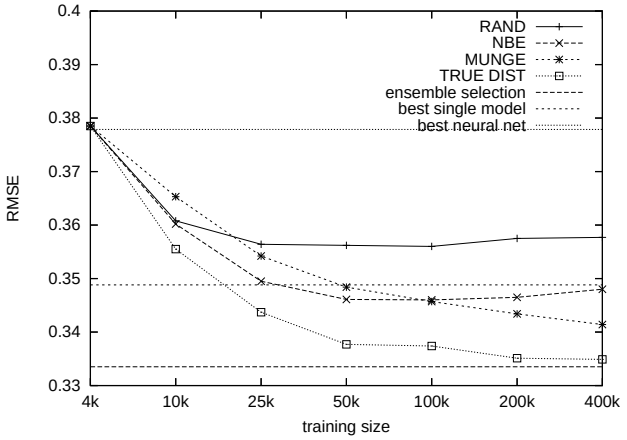

Figure 4: Perf. vs train set size for COVTYPE.

to label the pseudo data and train the mimic neural net than it was to train the original ensemble library and build the ensemble model. The expense of model compression is justified only when high performing models must be used in applications with limited storage or computational power, in applications where predictions are needed in real time, or where there will be very many test cases.

The amount of pseudo data needed to train a high-fidelity mimic model depends on the effectiveness of the method for generating artificial cases. It is clear from Figure 2 that MUNGE is more effective than RANDOM and NBE, but it would be interesting to assess the sample efficiency of MUNGE compared to unlabeled data drawn from the true distribution. Fortunately, more than 500,000 cases are available for the COVTYPE problem. Figure 4 shows the performance of the mimic neural nets when pseudo data is generated using RANDOM, NBE and MUNGE, as well as the performance of the mimic nets when real unlabeled test cases are used instead of pseudo data (TRUE DIST). Data drawn from the true distribution appears to be about 2-8 times as efficient as MUNGE data: MUNGE needs about 25k to match TRUE DIST at 10k, about 50k to match TRUE DIST at 15k, and about 200k to match TRUE DIST at 25k. This suggests that it may be possible to make further improvements to MUNGE, and that in domains where unlabeled data is available model compression will work even better.

Interestingly, on the HS problem, the performance of the mimic neural net trained on MUNGE pseudo data is better than the performance of the ensemble model it is trying to mimic. See HS in Figure 3. We suspect that the ensemble has overfit the data, and that the neural net may provide a beneficial form of smoothing/regularization.

5. RELATED WORK

Zeng and Martinez also used neural nets to approximate ensembles of classifiers [19]. In the experiments they presented, Zeng and Martinez tryed to approximate only ensembles of ten neural nets. The mimic nets were trained on synthetic examples generated using an algorithm similar to RANDOM. We found that, although it yields an improvement on a couple of problems, on average, generating synthetic data using RANDOM does not provide a significant improvement over simply training the neural nets on the original data. Generating synthetic data using NBE

or MUNGE works much better. Also, Zeng and Martinez showed that only a small number of synthetic examples, comparable to the size of the initial train set, is sufficient to obtain good results. This might be an indication that the functions that were approximated, ensembles of ten neural nets, were not hard enough to learn by a single neural net. In contrast, we tried to approximate much harder functions generated by ensemble selection, and as a consequence we needed a lot more synthetic data to obtain the best results.

TREPAN was used to extract tree-structured representations of trained neural nets [6]. The TREPAN approach also made use of additional artificial data generated from the same train set as the one used for training the nets. New examples were generated as needed by randomly selecting new values for each attribute, while ensuring that some constraints are satisfied. For discrete attributes the values were generated the same way as RANDOM. For continuous attributes, a kernel density estimator [15] was used. Although TREPAN models sometimes were more compact than the neural nets they explained, and occasionally outperformed decision trees trained directly on the original training data, the goal in TREPAN was not to train compact models or high performing models.

CMM is a meta-learner that learns one single model from a bagged ensemble of the same type of models [8]. Again, the approach makes use of additional, artificially created data and labeled according to the bagged model. The method used to generate synthetic data was specific to the base models used in the ensemble. The base models were decision rule sets. For each decision rule in the rule set, some examples were generated randomly from the hyperspace classified by the decision rule. Although CMM generates smaller models, they are not optimized for size or performance, but for comprehensibility instead.

DECORATE [13] uses artificial data to increase diversity so that better ensembles can be trained. Before DECORATE trains a new base learner, it generates and labels new data the opposite way to the predictions of the current ensemble. The new trained model is added to the current ensemble only if the addition is beneficial to the ensemble. This approach generates the synthetic data randomly from the distribution of the train set. For continuous attributes, the new values are generated from a Gaussian distribution that has the same mean and standard deviation as the set of values for that attribute. For discrete attributes, frequency counts are used, which is similar to RANDOM.

6. CONCLUSIONS

Some of the highest performing models currently available are complex ensembles containing hundreds or thousand of base-level classifiers. These models have excellent performance, but can be so large and slow that it may be infeasible to use them when memory or computational power is limited, when predictions are needed in real time, or when test sets are extremely large.

We present a method for model compression that is able to train fast, compact models to mimic better performing, but slow and complex models with little loss in performance. Compression works by labeling a large unlabeled data set with the target model, and then training a neural net using the newly labeled data. Where unlabeled data is not available, we present a new method called MUNGE for generating pseudo data from a distribution similar to that of the true data. We show that MUNGE compares favorably with other methods for generating artificial cases. MUNGE, however, is not as effective as drawing unlabeled data from the true distribution, so using true unlabeled data is preferred when it is available.

We present experiments where complex ensemble models are compressed using neural networks. Results on eight test problems show that, on average, the loss in performance due to compression is usually negligible, yet the mimic neural nets are 1000 times smaller and 1000 times faster.

Acknowledgments

This work was supported by NSF Award 0412930.

7. REFERENCES

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

[2] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[4] W. Buntine and R. Caruana. Introduction to IND and recursive partitioning. Technical Report FIA-91-28, NASA Ames Research Center, 10 1991.

[5] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Proc. 21st International Conference on Machine Learning*, 2004.

[6] M. W. Craven and J. W. Shavlik. Extracting tree-structured representations of trained networks. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 24–30. The MIT Press, 1996.

[7] A. P. Dempster, N. M. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 1(39):1–38, 1977.

[8] P. Domingos. Knowledge acquisition from examples via multiple models. In *Proc. 14th International Conference on Machine Learning*, pages 98–106. Morgan Kaufmann, 1997.

[9] P. Domingos. Bayesian averaging of classifiers and the overfitting problem. In *Proc. 17th International Conf. on Machine Learning*, pages 223–230. Morgan Kaufmann, San Francisco, CA, 2000.

[10] A. Gualtieri, S. R. Chettri, R. Cromp, and L. Johnson. Support vector machine classifiers as applied to aviris data. In *Proc. Eighth JPL Airborne Geoscience Workshop*, 1999.

[11] T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods*, 1999.

[12] D. Loyd and P. Domingos. Naive Bayes models for probability estimation. In *Proceedings of the 22nd International Conference on Machine Learning (ICML'05)*, Bonn, Germany, 2005.

[13] P. Melville and R. Mooney. Constructing diverse classifier ensembles using artificial training examples. In *Proceedings of the IJCAI-2003*, pages 505–510, Acapulco, Mexico, 2003.

[14] R. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2001.

[15] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[16] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[17] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[18] A. Zell, N. Mache, R. Huebner, M. Schmalzl, T. Sommer, and T. Korb. SNNS: Stuttgart neural network simulator. Technical report, University of Stuttgart, Stuttgart, 1992.

[19] X. Zeng and T. R. Martinez. Using a neural network to approximate an ensemble of classifiers. *Neural Processing Letters*, 12(3):225–237, 2000.