Neural Network Ensembles from Training Set Expansions

Debrup Chakraborty

Computer Science Department, CINVESTAV-IPN, Av. IPN No. 2508, Col. San Pedro Zacatenco, Mexico, D.F. 07360, Mexico
debrup@cs.cinvestav.mx

Abstract. In this work we propose a new method to create neural network ensembles. Our methodology develops over the conventional technique of *bagging*, where multiple classifiers are trained using a single training data set by generating multiple bootstrap samples from the training data. We propose a new method of sampling using the k-nearest neighbor density estimates. Our sampling technique gives rise to more variability in the data sets than by bagging. We validate our method by testing on several real data sets and show that our method outperforms bagging.

1 Introduction

The goal of constructing an ensemble of classifiers is to train a diverse set of classifiers from a single available training data set, and to combine their outputs using a suitable aggregation function. In the past few years there have been numerous proposals for creating ensembles of classifiers, and in general, it have been noticed that an ensemble of classifiers have better generalization abilities than a single classifier. Two of the well known proposals for creating classifier ensembles are *bagging* [2] and *boosting* [12]. Ample theoretical and experimental studies of Bagging, Boosting and their variants have been reported in the literature, and these studies clearly point out why and under which scenarios ensembles created by these methods can give better predictions [2,9,10,13].

Bagging is a popular ensemble method which can significantly improve generalization abilities of "unstable" classifiers [2]. In bagging, given a training data set $\mathcal{L}_x = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\} \subset \Re^n$ with the associated class labels, α independent bootstrap samples [5] are drawn from $\mathcal{L}_x$ each of size m. In other words, from the original training set $\mathcal{L}_x$, α different sets $\mathcal{B}_1, \mathcal{B}_1, \ldots, \mathcal{B}_\alpha$ are obtained each containing m points with their associated labels. These α different sets thus obtained are used to train α different classifiers. In the discussions that follow we shall call a single member of the ensemble as a *candidate*. The final decision is made by an aggregation of the outputs of the candidates. The type of aggregation depends on the type of the output, i.e., whether it is a numerical response or a class label. Generally, for classification a majority voting type aggregation is applied, whereas in case of regression (function approximation) type problems

E. Bayro-Corrochano and J.-O. Eklundh (Eds.): CIARP 2009, LNCS 5856, pp. 629–636, 2009.

an average or a weighted average is used. This simple procedure can decrease the classification error and give better classifiers with good generalization abilities. The intuitive reason of why bagging works is that each candidate learns a slightly different decision boundary, and thus the combination of all the different decision boundaries learned by the candidate classifiers give rise to less variance in the classification error. In [2] Leo Breiman provided theoretical justification of the fact that one can obtain significant improvement in performance by bagging unstable classifiers. It was also noted in [2] that supervised feed-forward neural networks like the multilayer perceptron (MLP) are unstable, i.e., it is not necessary that for a trained MLP, small changes in the input will produce small changes in the output. Thus it is expected that bagging can decrease classification errors in MLP classifiers to a large extent

Right from the early nineties neural network ensembles has been widely studied [8,14]. A class of studies regarding neural network ensembles are directed towards adapting suitably the general ensemble techniques in case of neural networks [4]. Other studies have been focussed on developing heuristics to choose better candidates for an ensemble such that each candidate has good prediction power along with that the selected candidates have better diversity [3,6], which is known to affect the performance of an ensemble [9,10].

In this paper we propose a new method to create neural network ensembles based on bagging. As discussed earlier, in bagging a bootstrap sample of a given training set is used to train a candidate classifier. A bootstrap sample is generated by sampling with replacement, so the difference among the various bootstrap samples is that there may be some data points missing or some data points may get repeated. In the proposed method we aim to achieve more diversity in each of the training set which would be used to train the candidates of the ensemble. In the ideal scenario it can be assumed that the training data gets generated from a fixed but unknown time-invariant probability distribution. It would have been the best if the different training sets for the candidates could have been independently generated following the same probability distribution from which the training data was generated. But, as this distribution is unknown, so such a method cannot be developed in practice. One of the closely related options can be to estimate the probability distribution of the training data and thus draw different training sets from this estimated distribution. Our work is motivated by this approach. The problem of this approach is that generally the number of available training data is too small to have a reasonable estimate of the distribution. So, in this work we do not attempt to estimate the true probability distribution of the training set, but we propose a method to generate new data points such that the new points are generated according to the spatial density of the training set, i.e., more points are generated in the dense regions of the data and less points in the sparse regions.

The heart of our method is the k-nearest neighbor (k-NN) density estimation and classification procedure. The new data points that are generated for training the candidates in a sense follows the k-NN density estimate of the original training data. This technique has been successfully used for data condensation

in [11]. But we use it for a completely different goal. We generate new points for each candidate and mix these new points with the original training data and train the candidate with this data. Thus, it is expected that the training sets used for the candidates are more diverse than the bootstrap samples. Our experiments demonstrate that this technique when applied to MLP ensembles can give better results than conventional bagging.

2 k Nearest Neighbor Density Estimation

Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m \in \Re^n$ be independently generated from a continuous probability distribution with density f. The nearest neighbor density estimation procedure finds the density of a point $\boldsymbol{z}$. We describe the methodology in brief next.

Let $||\boldsymbol{x} - \boldsymbol{z}||$ denote the Euclidian distance between points $\boldsymbol{x}$ and $\boldsymbol{z}$. A n dimensional hyper-sphere centered at $\boldsymbol{x}$ with radius r is given by the set $S_{\boldsymbol{x},r} = \{\boldsymbol{z} \in \Re^n : ||\boldsymbol{x} - \boldsymbol{z}|| \leq r\}$. We call the volume of this sphere as $V_r = \mathsf{Vol}(S_{\boldsymbol{x},r})$. Let $k(N)$ be a sequence of positive integers such that $\lim_{N\to\infty} k(N) = \infty$ and $\lim_{N\to\infty} k(N)/N = 0$. Suppose we have a sample $\mathcal{L}_x = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\} \subset \Re^n$, and we fix a value of $k(N)$. Let $r_{k(N),\boldsymbol{z}}$ be the Euclidian distance of $\boldsymbol{z}$ from its $(k(N)+1)$-th nearest neighbor in $\mathcal{L}_x$. Then the density at $\boldsymbol{z}$ is estimated as

$$\hat{f}(\boldsymbol{z}) = \frac{k(N)}{N} \times \frac{1}{V_{r_{k(N),\boldsymbol{z}}}} \tag{1}$$

It has been shown that this estimate is asymptotically un-biased and consistent, but it is known that this estimate suffers from the curse of dimensionality, i.e., the estimate gets unstable for high dimensional data. We shall use this density estimation technique to generate new training points, which we describe next.

3 Expanding a Training Set

Our basic motivation is to increase the variability of the individual training sets which we shall use to train each candidate classifier. The idea is to create new training points which are similar to the ones in the original training set. Ideally, we want to generate points from the same probability distribution from which the training data was generated. As that distribution is unknown to us and obtaining a reasonable estimate from a small training set is not feasible we shall apply some heuristic to generate new points following the rule that more points should be generated in the denser regions of the distribution.

Given a labeled data set $\mathcal{L} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) : \boldsymbol{x}_i \in \Re^n, \boldsymbol{y}_i \in \{1, 2, \ldots, c\}, i = 1, \ldots m\}$, we shall call the set of the input vectors as $\mathcal{L}_x = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$. We shall denote the label (or output) associated with $\boldsymbol{x}$ as $\ell(\boldsymbol{x})$. For each $\boldsymbol{x}_i$ we compute the distance of its k-th nearest neighbor in $\mathcal{L}_x$. We call this distance as d_i. From eq. (1), it is clear that the density at a point $\boldsymbol{x}_i$ is inversely related to the volume of the hypersphere centered at $\boldsymbol{x}_i$ with radius d_i. So, it can be inferred that points with higher values of d_i lies in less dense areas and the points

which low values of d_i lies in denser areas. Our objective is to generate new points following the density of the original data, i.e., our method must be such that more points are generated in the denser regions and less points in sparse regions. To achieve this, for each $\boldsymbol{x}_i \in \mathcal{L}_x$ we define a quantity p as follows:

$$p(\boldsymbol{x}_i) = \frac{1}{Z} e^{-d_i} \tag{2}$$

where

$$Z = \sum_{i=1}^{m} e^{-d_i} \tag{3}$$

This definition of p guarantees that for a point $\boldsymbol{x}_i$ the value of $p(\boldsymbol{x}_i)$ would be large if d_i is small and vice versa. Also, because of the way we define p it is obvious that for all $\boldsymbol{x}_i \in \mathcal{L}_x$, $0 \leq p(\boldsymbol{x}_i) \leq 1$, and also $\sum_{i=1}^{m} p(\boldsymbol{x}_i) = 1$. Thus p can be treated as a discrete probability distribution on the set $\mathcal{L}_x$. To generate a single new point we first sample a point randomly from $\mathcal{L}_x$ according to the probability distribution p. The roullet wheel selection technique can be used for this purpose. Let $\boldsymbol{x} \in \mathcal{L}_x$ be the sampled point. As $\boldsymbol{x}$ has been sampled according to the probability p, with high probability it will lie in a dense region of the training data. Let $\{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_k\}$ be the k nearest neighbors of the sampled point $\boldsymbol{x}$. Let $\mathsf{NBRS}(\boldsymbol{x})$ be the set containing the k nearest neighbors of $\boldsymbol{x}$ along with $\boldsymbol{x}$, i.e,

$$\mathsf{NBRS}(\boldsymbol{x}) = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_k\} \cup \{\boldsymbol{x}\}.$$

We now generate the new point $\tilde{\boldsymbol{x}}$ as a random convex combination of the points in $\mathsf{NBRS}(\boldsymbol{x})$. In other words, let each λ_j, for $j = 1, \ldots, k+1$, be generated independently from a uniform random distribution over $[0, 1]$, we compute $\tilde{\boldsymbol{x}}$ as

$$\tilde{\boldsymbol{x}} = \frac{\sum_{j=1}^{k} \lambda_j \boldsymbol{z}_j + \lambda_{k+1} \boldsymbol{x}}{\sum_{j=1}^{k+1} \lambda_j}. \tag{4}$$

The new point will thus lie within the convex hull of the points in $\mathsf{NBRS}(\boldsymbol{x})$, and thus cannot be very atypical of the points already present in the training set.

The new point $\tilde{\boldsymbol{x}}$ was not present in the training set, so to incorporate it into the training set we need to label this point, i.e., assign a target output to this point. The most natural label of $\tilde{\boldsymbol{x}}$ would be that label which the majority of its neighbors have. Note, that $\mathsf{NBRS}(\boldsymbol{x})$ are the $k+1$ neighbors of $\tilde{\boldsymbol{x}}$ (including $\boldsymbol{x}$ itself). Thus, the label of $\tilde{\boldsymbol{x}}$ is calculated as

$$\ell(\tilde{\boldsymbol{x}}) = \underset{j=1,\ldots,c}{\operatorname{argmax}} \sum_{\boldsymbol{z} \in \mathsf{NBRS}(\boldsymbol{x})} \delta(j, \ell(\boldsymbol{z})),$$

where $\delta(a, b) = 1$ if $a = b$, and $\delta(a, b) = 0$ if $a \neq b$.

The method described above can be repeated to obtain the desired number of new points. The algorithm in Fig. 1 summarizes the procedure described above. The algorithm Expand as described in Fig. 1 takes as input the training set $\mathcal{L}$, along with the parameters k and ν, where ν is the number of points that

Algorithm Expand($\mathcal{L}$,k,ν)
1. $Z \leftarrow 0$;
2. **for** $i = 1$ to m;
3. $d_i \leftarrow$ Distance of the k-th nearest neighbor of $\boldsymbol{x}_i$ in $\mathcal{L}_x$;
4. $p(\boldsymbol{x}_i) \leftarrow e^{-d_i}$;
5. $Z \leftarrow Z + p(\boldsymbol{x}_i)$
6. **end for**
7. **for** $i = 1$ to m,
8. $p(\boldsymbol{x}_i) \leftarrow p(\boldsymbol{x}_i)/Z$;
9. **end for**
10. NewPoints $\leftarrow \emptyset$;
11. **while** |NewPoints| $< \nu$,
12. Select $\boldsymbol{x}$ from $\mathcal{L}_x$ with probability $p(\boldsymbol{x})$
13. $\{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_k\} \leftarrow k$ nearest neighbors of $\boldsymbol{x}$ in $\mathcal{L}_x$;
14. /* Let NBRS($\boldsymbol{x}$) $= \{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_k\} \cup \boldsymbol{x}$ */
15. $\lambda_1, \lambda_2, \ldots, \lambda_{k+1} \sim U[0,1]$; $\Lambda \leftarrow \sum_{i=1}^{k+1} \lambda_i$
16. $\tilde{\boldsymbol{x}} \leftarrow (\sum_{i=1}^{k} \lambda_i \boldsymbol{z}_i + \lambda_{k+1}\boldsymbol{x})/\Lambda$;
17. $\ell(\tilde{\boldsymbol{x}}) \leftarrow \text{argmax}_{j=1,\ldots,c} \sum_{\boldsymbol{z} \in \text{NBRS}(\boldsymbol{x})} \delta(j, \ell(\boldsymbol{z}))$;
18. NewPoints $\leftarrow$ NewPoints $\cup \{(\tilde{\boldsymbol{x}}, \ell(\tilde{\boldsymbol{x}}))\}$
19. **end while**
20. **return** NewPoints;

Fig. 1. Algorithm to expand a training set

are required to be generated. It gives as output a set called NewPoints, which contains ν many new points generated by the procedure.

4 Creating the Ensemble

Our strategy of creating the ensemble closely follows bagging, except the fact that instead of using bootstrap samples for training the candidates of the ensemble we use the algorithm Expand of Fig. 1 to create new points and mix them with the original training set. Given a training set $\mathcal{L}$ we decide upon the size of the ensemble, i.e., the number of candidate classifiers. Let us call this as α. We fix two integers k and ν and call Expand($\mathcal{L}, k, \nu$) α times. By this way we obtain $S_1, S_2, \ldots, S_\alpha$ as output, where each S_i contains ν points. For training the i-th candidate we train a multilayered perceptron using $\mathcal{L} \cup S_i$. Thus obtaining α trained networks. For using the network, we feed a test point to all the α networks and decide the class of the test point by a majority vote. The algorithm for creating the ensemble is depicted in Fig. 2.

The algorithm Create_Ensemble takes as input the training set $\mathcal{L}$, k, the number of new points to be used for each candidate ν and the size of the ensemble α. The algorithm calls a function Train, which takes as input a training set and a variable $\mathcal{A}$ which contains the parameters necessary to fix the architecture of a network. The algorithm Train outputs a vector $\boldsymbol{W}$ which contains the weights

Algorithm Create_Ensemble($\mathcal{L}$,k,ν,α)
1. **for** $i = 1$ to α;
2. $S_i \leftarrow$ Expand($\mathcal{L}$,k,ν);
3. $\boldsymbol{W}_i \leftarrow$ Train($\mathcal{L} \cup S_i, \mathcal{A}_i$)
4. **end for**;
5. return $(\boldsymbol{W}_1, \mathcal{A}_1), \ldots, (\boldsymbol{W}_\alpha, \mathcal{A}_\alpha)$;

Fig. 2. Algorithm for creating the ensemble

and biases of the network. Thus $\mathcal{A}$ and $\boldsymbol{W}$ together will specify a trained network. The output of Create_Ensemble is α trained networks. The decision on a test point is taken by a majority vote of these α networks.

The algorithm Train takes in two user defined parameters, k and ν. k is the parameter for the k nearest neighbor density estimation procedure. Choosing a proper value of k is a classical problem which do not yet have an well accepted solution, but there exist solutions (some very complicated) which solves this problem [7]. In the current work we do not attempt to solve this problem. In the next section we present some simulation results using this algorithm, we tested with numerous small values of k, we found that the performance do change with the change of k, but we did not find any significant pattern which shows a conclusive dependence of the parameter k with the performance. Based on experiments we suggest a value of k near 5. The parameter ν decides the number of new points that are to be included in each training set which is used for training the candidates. A small value of ν will mean little variation among the training set, and a big value of ν will mean more variability. But, the new points generated by Expand are noisy versions of the original training set, so a very big value of ν is not recommended. Our experiments suggest that ν being 10% of the size of the original training data gives good results.

The computational overhead in creating the ensemble is same as bagging except that it has the additional overhead of the function Expand. Expand requires finding the k nearest neighbors of each data point for computing the value d_i, this operation is computationally costlier than other operations involved. But the computation of the values d_i are a one time operation and they are not required to be repeated when Expand is called on the same training data multiple times. Thus, the total computational cost in creating the ensemble is not significantly more than that of conventional bagging.

5 Experimental Results

We tried our method on six real data sets from the UCI repository [1]. The data sets used are Iris, Wine, Liver-Disorder(Liver), Waveform-21(Wave), Pima-Indian-Diabetes (Pima), and Wisconsin Breast cancer (WBC). For the experiments we used the multilayered perceptron implementation of MATLAB. In particular we used the Levenberg-Marquardt backpropagation algorithm implemented as 'trainlm' method in MATLAB for training.

Table 1. The results

Data set	Single network	Conventional Bagging	Proposed Method			
			$k = 3$	$k = 5$	$k = 7$	$k = 9$
Iris	91.26±6.11	96.08±2.66	96.46±0.54	97.00±0.47	96.67±0.44	96.86±0.32
Wine	92.02±4.86	97.18±1.88	**98.93±0.32**	**98.70±0.38**	**98.70±0.38**	**99.04±0.53**
Liver	64.85±3.21	67.60±1.74	68.63±1.26	68.95±2.24	68.78±1.28	**68.95±1.70**
Wave	62.74±5.93	84.10±1.88	**86.09±0.18**	**86.43±0.32**	**85.98±0.26**	**85.70±0.24**
Pima	66.35±5.14	75.11±1.06	**77.03±0.83**	**76.66 ± 0.53**	**76.97±0.52**	**76.94±0.48**
WBC	95.71±0.54	96.37±0.44	95.98±0.41	96.06±0.34	96.10±0.33	95.86±0.36
Glass	62.06±3.53	67.66±1.78	**70.70±1.67**	**70.42±1.66**	**69.75±1.82**	**70.18±1.55**

Each of the results reported are for an MLP with 10 nodes in a single hidden layer. Each node has a sigmoidal activation function. Though we agree that this is not supposed to be 'optimal' for all cases. We could have used a validation set for determining the proper number of hidden unit for each data set. But, here our objective is to show that our method performs better than conventional bagging. So we decided to keep the number of hidden units and the number of hidden layer to be fixed across runs irrespective of the data sets. Same decision was taken with respect to the number of candidates in the ensemble. We fixed the number of members in the ensemble to be 10 for all cases. For all the data sets we take ν equal to 10% of the size of the training data.

The performance results reported are for a 10 fold cross validation repeated 10 times. The figures in Table 1 give the average performance and the standard deviation (in percentage) for six different scenarios. The performance of a single network, that of conventional bagging and that of our proposed method using $k = 3, 5, 7, 9$.

Table 1 clearly shows that the proposed method gives better results than conventional bagging for almost all data sets. The amount of improvement for some data sets are statistically significant. The figures are shown in bold if the performance of the proposed method is significantly better than conventional bagging[1].

6 Conclusion

We demonstrated a new method of creating ensembles. Our experiments demonstrates that the method shows improvements over conventional bagging for most of the data sets tried. We plan to address the following problems in future:

1. The procedures Expand and Train are quite general and can be used to train other kinds of classifiers other than a MLP. We plan to apply the method for other classifiers, in particular decision trees seem to be a good alternative.
2. Quantify the diversity among the candidates that this method yeilds.

[1] These results are based on a studentized t-test with 95% confidence.

Acknowledgements. This work was partially supported by CONACYT project I0013/APOY-COMPL-2008/90775.

References

1. Asuncion, A., Newman, D.J.: UCI machine learning repository (2007)
2. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
3. Chen, R., Yu, J.: An improved bagging neural network ensemble algorithm and its application. In: Third International Conference on Natural Computation, vol. 5, pp. 730–734 (2007)
4. Drucker, H., Schapire, R.E., Simard, P.: Improving performance in neural networks using a boosting algorithm. In: Hanson, S.J., Cowan, J.D., Giles, C.L. (eds.) NIPS, pp. 42–49. Morgan Kaufmann, San Francisco (1992)
5. Efron, B., Tibshirani, R.: An Introduction to the Bootstrap. CRC Press, Boca Raton (1993)
6. Georgiou, V.L., Alevizos, P.D., Vrahatis, M.N.: Novel approaches to probabilistic neural networks through bagging and evolutionary estimating of prior probabilities. Neural Processing Letters 27(2), 153–162 (2008)
7. Ghosh, A.K.: On optimum choice of k in nearest neighbor classification. Computational Statistics & Data Analysis 50(11), 3113–3123 (2006)
8. Hansen, L.K., Salamon, P.: Neural network ensembles. IEEE Trans. Pattern Anal. Mach. Intell. 12(10), 993–1001 (1990)
9. Kuncheva, L.I.: Diversity in multiple classifier systems. Information Fusion 6(1), 3–4 (2005)
10. Kuncheva, L.I., Whitaker, C.J.: Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. Machine Learning 51(2), 181–207 (2003)
11. Mitra, P., Murthy, C.A., Pal, S.K.: Density-based multiscale data condensation. IEEE Trans. Pattern Anal. Mach. Intell. 24(6), 734–747 (2002)
12. Schapire, R.E.: A brief introduction to boosting. In: Dean, T. (ed.) IJCAI, pp. 1401–1406. Morgan Kaufmann, San Francisco (1999)
13. Schapire, R.E.: Theoretical views of boosting. In: Fischer, P., Simon, H.U. (eds.) EuroCOLT 1999. LNCS (LNAI), vol. 1572, pp. 1–10. Springer, Heidelberg (1999)
14. Zhou, Z.-H., Wu, J., Tang, W.: Ensembling neural networks: Many could be better than all. Artificial Intelligence 137(1-2), 239–263 (2002)