# Choosing the Best Auto-Encoder-Based Bagging Classifier: An Empirical Study

Yifan Nie[1], Wenge Rong[2,3], Yikang Shen[1], Chao Li[2,3], and Zhang Xiong[2,3]

[1] Sino-French Enginnering School, Beihang University, Beijing, China
[2] School of Computer Science and Engineering, Beihang University, Beijing, China
[3] Research Institute of Beihang University in Shenzhen, Shenzhen, China
{yifan.nie,yikang.shen}@ecpk, {w.rong,licc,xiongz@}buaa.edu.cn

**Abstract.** Feature learning plays an important role in many machine learning tasks. As a common implementation for feature learning, the auto-encoder has shown excellent performance. However it also faces several challenges among which a notable one is how to reduce its generalization error. Different approaches have been proposed to solve this problem and Bagging is lauded as a possible one since it is easily implemented while can also expect outstanding performance. This paper studies the problem of integrating different prediction models by bagging auto-encoder-based classifiers in order to reduce generalization error and improve prediction performance. Furthermore, experimental study on different datasets from different domains is conducted. Several integration schemas are empirically evaluated to analyse their pros and cons. It is believed that this work will offer researchers in this field insight in bagging auto-encoder-based classifiers.

**Keywords:** bagging, auto-encoders, feature learning, neural networks.

## 1 Introduction

When it comes to feature learning, there are two commonly used approaches, i.e., handcrafting features learning and unsupervised features learning [15]. Compared with manually selecting features, which is arduous and requires several validation tests to determine which features are the most representative [10], unsupervised features learning employs learning models to obtain appropriate features. In recent years neural network based feature learning model has been attached much importance as it allows users to perform unsupervised feature learning with unlabelled data [12] and one of the popular learning models is stacked auto-encoder [7].

To build a stacked auto-encoder-based feature learning model, a common approach is to add one classification layer on the top of stacked auto-encoder and then take the learned features as input for the classification layer [9]. Though the auto-encoder-based classification model has many advantages, it still faces several challenges among which a notable one is how to reduce generalization error [4]. To meet this challenge, a lot of approaches have been proposed among which widely adopted ones are ensemble methods [13].

Bagging (Bootstrap Aggregation) is a popular ensemble method which consists in bootstrapping several copies of the training set and then employing them to train several separate models. Afterwards it combines the individual predictions together by a voting scheme for classification applications [5]. As each bootstrapped training set is slightly different from each other, each model trained on theses training sets has different weights and different focus, thereby having different generalization error. By combining them together, the overall generalization error is expected to decrease to some extent.

Previous works have shown that bagging works well for unstable predictors [14]. Considering neural-network-based models are also unstable predictors [19], it is intuitive to assume that applying bagging methods to feature learning based models could also probably improve classification performance. Therefore a fundamental question is arisen accordingly: is it possible to integrate feature learning with ensemble method such as bagging? If so, how can we effectively integrate them? In this research, we thoroughly investigate the possibility of integrating feature learning with bagging ensemble method and further provide different integration schemas and empirically evaluate their pros and cons.

The remainder of this paper is organised as follow. In section 2 we introduce the background about auto-encoder and bagging techniques. Section 3 presents the proposed evaluation architecture. In section 4 we will present the experimental settings and also discuss the experimental results. Finally section 5 concludes the paper and points possible future work.

## 2 Background

### 2.1 Auto-Encoder

Auto-encoder has been widely used as an unsupervised feature learning tool [1]. Typically, a basic auto-encoder consists of three layers, i.e, input layer, hidden layer (or code layer), and output layer [17] which has the same number of units as the input layer. This structure can achieve unsupervised feature learning by feeding the unlabelled data into the input layer and forcing the output layer to reconstruct the input. The encode phase is implemented by mapping the input $x \in \mathbb{R}^m$ to a hidden representation (code) $h \in \mathbb{R}^n$, which has the form:

$$h = f(x) = s(Wx + b_h) \tag{1}$$

where $s$ is the sigmoid activation function $s(z) = \frac{1}{1+e^{-z}}$, $m$ is the number of input units, $n$ is the number of hidden units, $x$ is the input feature vector, $h$ is the extracted code and the encoder is parametrised by the $n \times m$ weight matrix $W$, and $b_h$ is the bias vector. Afterwards the decoder maps the hidden representation back to a reconstruction $\hat{x} \in \mathbb{R}^m$:

$$\hat{x} = g(h) = s(W^T h + b_{\hat{x}}) \tag{2}$$

where $s$ is the sigmoid activation function $s(z) = \frac{1}{1+e^{-z}}$, $m$ is the number of output units, $n$ is the number of hidden units, $h$ is the extracted code, $\hat{x}$ is the reconstructed input feature vector, and the parameters are a bias vector $b_{\hat{x}}$ and $W^T$ is the tied weight matrix. The training process consists in finding the parameters $\theta = \{W, b_h, b_{\hat{x}}\}$, which aim to minimise the cost function:

$$J(\theta) = \sum_{x \in trainset} L(x, g(f(x))) \tag{3}$$

where $x$ is a feature vector and $f$, $g$ are encode and decode function in Eq. 1 and Eq. 2. The squared error is often used as the reconstruction error $L$, i.e., $L(x, \hat{x}) = \|x - \hat{x}\|$. Once the training process is completed, it is able to take the code in the hidden layer as unsupervised learned features.

### 2.2 Bagging

Bagging is a widely used integration technique to combine several prediction models in order to improve accuracy [6]. Firstly, bootstrapped training sets are generated from the original training set $X = \{x_1, x_2, ..., x_m\}$, and this procedure is conducted by sampling with replacement $m$ elements with equal probability from the original training set $X$. Because the sampling is conducted with replacement, there will be repeated examples in the bootstrapped training set. As a result if $m$ is large, asymptotically the fraction of unique examples is $lim_{m \to \infty} 1 - (1 - \frac{1}{m})^m = 63.2\%$. As there are repeated data, the unique data in each bootstrapped training set are randomly different. As such the models trained on these bootstrapped training sets have different focuses. Suppose $x$ is a test example, $y$ is one possible label, $Y$ is the set containing all possible labels, $h_i$ is a basic prediction model and $h^*$ is the combined prediction function, the combining process will combine each trained classifier in the following way [2]:

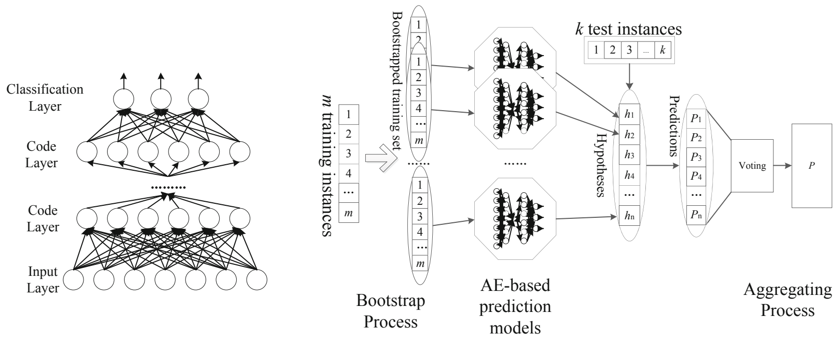
$$h^*(x) = \arg\max_{y \in Y} \sum_{i: h_i(x) = y} 1 \tag{4}$$

## 3 Evaluation Architecture

In order to validate the possibility and evaluate the performance of bagging different feature learning models, in this paper an evaluation architecture is proposed and its architecture is presented in Fig. 1(b).

In this architecture, each model is an auto-encoder-based classification model which is chosen to integrate unsupervised feature learning into prediction models. The model is composed by two parts: 1) stacked auto-encoder-based feature learning layers and 2) a classification layer. The stacked auto-encoder-based feature learning part has multiple layers in order to obtain higher-level representation of the data and performs feature learning [3]. The classification layer is on top of the feature learning layers, and it takes the learned features as input and performs classification, as shown in Fig. 1(a).

The stacked auto-encoder is configured with 0 sparsity penalty and no denoising treatment. All activation functions are sigmoid function and for every



(a) Auto-encoder Structure (b) Evaluation Architecture

**Fig. 1.** Auto-encoder-based Bagging Prediction Architecture

dataset, we try 1, 2, 3 code layers in the proposed model, respectively, as indicated in [18]. As for training process, a two-stage training approach is adopted. Firstly the proposed model performs a layer-wise unsupervised pre-training with unlabelled data [7] . This procedure aims to obtain unsupervised learned features in code layers. Once each code layer is pre-trained, the whole network can be further fine-tuned to include the classification layer with labelled data by traditional back-propagation algorithm [16].

Afterwards $n$ bootstrapped training sets are generated and fed into $n$ auto-encoder-based classification models. Finally through bagging the $n$ basic models together, the overall performance will be evaluated. In this research the number $n$ is set to vary from 1 to 20. The decision of limiting the number of trained basic models is both important in order to complete the experimental study in a reasonable time and for practical applications as indicated in [2].

## 4 Experimental Study

### 4.1 Dataset and Evaluation Metrics

In this research, five commonly used datasets[1,2] are employed and listed in Table 1(a). And the raw performance improvement and the relative performance improvement [2,5] over the average single model are utilised to evaluate the bagging auto-encoder performance:

$$raw\ improvement = (ACC_{bag} - ACC_{avg}) * 100\% \tag{5}$$

$$relative\ improvement = \frac{ACC_{bag} - ACC_{avg}}{100\% - ACC_{avg}} * 100\% \tag{6}$$

[1] `http://yann.lecun.com/exdb/mnist/`

[2] `http://archive.ics.uci.edu/ml/`

where $ACC_{bag}$ stands for the accuracy after bagging, and $ACC_{avg}$ stands for the average accuracy of single model.

**Table 1.** Datasets and Configuration

(a) Datasets

| Dataset | #training instances | #test instances | #features | #classes |
|---|---|---|---|---|
| MNIST | 60000 | 10000 | 784 | 10 |
| Optdigit | 3822 | 1000 | 64 | 10 |
| Yeast | 1000 | 484 | 8 | 10 |
| CNAE9 | 990 | 90 | 856 | 9 |
| Semeion | 1400 | 193 | 256 | 10 |

(b) Layer Size Configuration

| Dataset | 1 Code Layer | 2 Code Layers | 3 Code Layers |
|---|---|---|---|
| MNIST | 784-100-10 | 784-100-100-10 | 784-100-100-50-10 |
| Optdigit | 64-32-10 | 64-50-25-10 | 64-50-40-30-10 |
| Yeast | 8-50-10 | 8-50-25-10 | 8-100-80-40-10 |
| CNAE9 | 856-100-9 | 856-428-214-9 | 856-428-214-107 |
| Semeion | 256-128-10 | 256-128-64-10 | 256-128-64-32-10 |

## 4.2 Experimental Result and Discussion

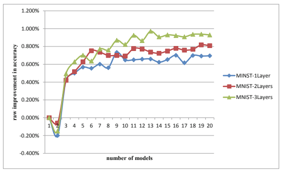
In this research, the number of units in each code layer is chosen by empirical evaluation to achieve best single model accuracy for each dataset [8]. The layer size configuration and experimental results on the five different datasets are summarised in Table 1(b) and Fig. 2. The performance in the best case is presented in Table 2.

**Table 2.** Performance in the Best Configuration

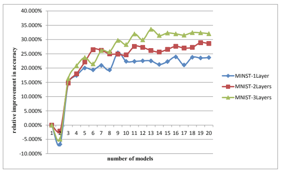
| datasets | single avg | bagging | raw improvement | relative improvement | number of models |
|---|---|---|---|---|---|
| MNIST-1L | 97.107% | 97.840% | 0.733% | 25.346% | 9 |
| MNIST-2L | 97.173% | 97.990% | 0.817% | 28.909% | 19 |
| MNIST-3L | 97.108% | 98.080% | 0.972% | 33.599% | 13 |
| Optdigit-1L | 95.650% | 96.272% | 0.621% | 14.286% | 6 |
| Optdigit-2L | 96.034% | 96.828% | 0.795% | 20.031% | 18 |
| Optdigit-3L | 95.727% | 96.494% | 0.767% | 17.958% | 19 |
| Yeast-1L | 48.347% | 51.033% | 2.686% | 5.200% | 2 |
| Yeast-2L | 49.403% | 54.959% | 5.556% | 10.980% | 9 |
| Yeast-3L | 51.033% | 55.372% | 4.339% | 8.861% | 8 |
| CNAE9-1L | 92.222% | 94.444% | 2.222% | 28.571% | 7 |
| CNAE9-2L | 97.259% | 97.778% | 0.519% | 18.919% | 15 |
| CNAE9-3L | 97.556% | 97.778% | 0.222% | 9.091% | 10 |
| Semeion-1L | 90.271% | 92.746% | 2.476% | 25.444% | 9 |
| Semeion-2L | 90.587% | 94.301% | 3.713% | 39.450% | 12 |
| Semeion-3L | 90.748% | 94.301% | 3.553% | 38.400% | 14 |

The experimental result has illustrated that it is possible to integrate feature learning with ensemble method such as bagging and the relative improvement in accuracy is satisfactory for big training set with many input features. Even for small training set with fewer input features, bagging still works and yields reasonable boost in the best configuration. Particularly the study on five datasets with different configurations reveal several lessons, which may give some insight for researchers in the domain for future bagging auto-encoder-based classifiers.

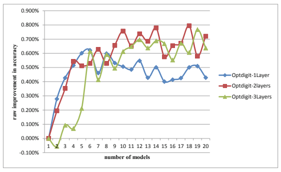
1. Bagging too few models (e.g., $n < 3$) is fruitless and sometimes the overall performance is even worse than the single model performance. It is observed



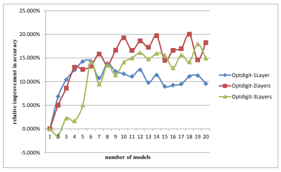
(a) Raw improvement after bagging for MNIST



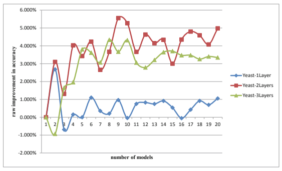
(b) Relative improvement after bagging for MNIST



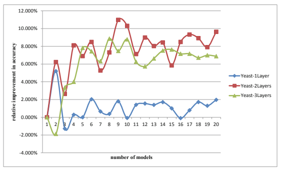
(c) Raw improvement after bagging for Optdigit



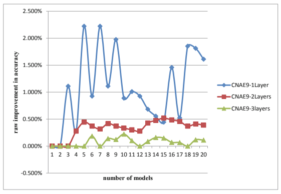
(d) Relative improvement after bagging for Optdigit



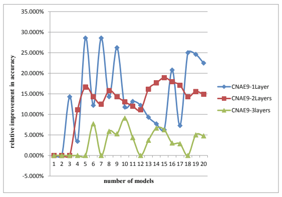
(e) Raw improvement after bagging for Yeast



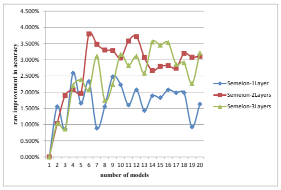
(f) Relative improvement after bagging for Yeast



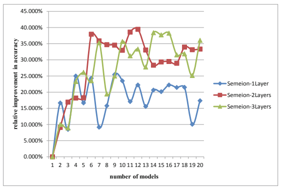
(g) Raw improvement after bagging for CNAE9



(h) Relative improvement after bagging for CNAE9



(i) Raw improvement after bagging for Semeion



(j) Relative improvement after bagging for Semeion

**Fig. 2.** Bagging Improvement for Five Datasets

from the experimental study that for some datasets (MNIST, Optdigit and Yeast) this will cause a negative boost and for other datasets (CNAE9 and Semeion), the boost is not quite significant compared with the performance of a single model. One possible explanation for this phenomenon is that each basic model trained on different bootstrapped training set has different focus. And given the too small number of models, when a controversial case comes, the high disagreement during bagging between the models will probably result in finally worse performance.

2. As the number of models increases, bagging begins to show its promising capability. The boost increases and seems to oscillate around a limit value. The possible reason for this phenomenon is that when the number of models

becomes bigger, there must be some overlaps between each bootstrapped training set. Since each trained model has different focus, the difference of theses focuses could not always be novel and might be repeated. As a result there could be a limit for the bagging performance. Bagging too many models does not help neither.

3. For text-based dataset (CNAE9), multiple code layers does not work well, as shown in Fig 2(g), single code layer configuration outperforms multiple code layers configuration. Whereas for image-based datasets (MNIST, Optdigit, and Semeion), multiple code layers configuration outperforms single code layer configuration. This may be due to the fact that texts themselves are already in a highly abstract form, more code layers could not extract further abstract features. However, for image-based training examples, the neighbouring features represent neighbouring pixels which are continuous and correlated. Multiple code layers can help extract more abstract features and thus increase performance.
4. For datasets which have too few features (Yeast with only 8 features), the boost with multiple code layers compared to single code layer configuration is much significant than datasets with more features (MNIST, Optdigit and Seimeion). This phenomenon might be due to the fact that for datasets with few features, more code layers can extract higher-level features and salient patterns which will improve the discriminative power of the classifier.

## 5 Conclusion and Future Work

The auto-encoder-based classification model has many advantages, but still faces several challenges. One of the challenges is how to reduce the model's generalization error. There are many ways to achieve this goal among which bagging is one of possible solutions. This study empirically studied the mechanism of integrating feature learning based classification models through bagging. By analysing the experimental result on different datasets with different configuration, some lessons are also summarised to show the pros and cons of the bagging oriented auto-encoder classifications. Besides auto-encoder, there exists other common feature learning structures among which DBN [11] is gaining more and more popularity. In the future, it is worthwhile to investigate bagging on these DBN based prediction models.

**Acknowledgements.** This work was partially supported by the National Natural Science Foundation of China (No. 61332018), the National High Technology Research and Development Program of China (No. 2013AA01A601), and the Fundamental Research Funds for the Central Universities. We are grateful to Shenzhen Key Laboratory of Data Vitalization (Smart City) for supporting this research.

## References

1. Baldi, P.: Autoencoders, unsupervised learning, and deep architectures. Journal of Machine Learning Research-Proceedings Track 27, 37–50 (2012)

2. Bauer, E., Kohavi, R.: An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. Machine Learning 36(1-2), 105–139 (1999)
3. Bengio, Y.: Learning deep architectures for ai. Foundations and Trends® in Machine Learning 2(1), 1–127 (2009)
4. Bengio, Y.: Deep learning of representations: Looking forward. In: Dediu, A.-H., Martín-Vide, C., Mitkov, R., Truthe, B. (eds.) SLSP 2013. LNCS (LNAI), vol. 7978, pp. 1–37. Springer, Heidelberg (2013)
5. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
6. Bühlmann, P.: Bagging, boosting and ensemble methods. In: Handbook of Computational Statistics, pp. 985–1022. Springer (2012)
7. Erhan, D., Bengio, Y., Courville, A., Manzagol, P.A., Vincent, P., Bengio, S.: Why does unsupervised pre-training help deep learning? Journal of Machine Learning Research 11, 581–616 (2010)
8. Erhan, D., Manzagol, P.A., Bengio, Y., Bengio, S., Vincent, P.: The difficulty of training deep architectures and the effect of unsupervised pre-training. In: Proceedings of the 12th International Conference on Artificial Intelligence and Statistics, pp. 153–160 (2009)
9. Glorot, X., Bordes, A., Bengio, Y.: Domain adaptation for large-scale sentiment classification: A deep learning approach. In: Proceedings of the 28th International Conference on Machine Learning, pp. 513–520 (2011)
10. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. The Journal of Machine Learning Research 3, 1157–1182 (2003)
11. Kang, S., Qian, X., Meng, H.: Multi-distribution deep belief network for speech synthesis. In: ICASSP, pp. 8012–8016 (2013)
12. Le, Q.V., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G.S., Dean, J., Ng, A.Y.: Building high-level features using large scale unsupervised learning. arXiv preprint arXiv:1112.6209 (2011)
13. Li, L., Hu, Q., Wu, X., Yu, D.: Exploration of classification confidence in ensemble learning. Pattern Recognition 47(9), 3120–3131 (2014)
14. Liang, G., Zhu, X., Zhang, C.: An empirical study of bagging predictors for different learning algorithms. In: Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (2011)
15. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning. In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning, vol. 2011 (2011)
16. Örkcü, H.H., Bal, H.: Comparing performances of backpropagation and genetic algorithms in the data classification. Expert Systems with Applications 38(4), 3703–3709 (2011)
17. Rifai, S., Mesnil, G., Vincent, P., Muller, X., Bengio, Y., Dauphin, Y., Glorot, X.: Higher order contractive auto-encoder. In: Gunopulos, D., Hofmann, T., Malerba, D., Vazirgiannis, M. (eds.) ECML PKDD 2011, Part II. LNCS (LNAI), vol. 6912, pp. 645–660. Springer, Heidelberg (2011)
18. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research 11, 3371–3408 (2010)
19. Yu, L., Wang, S., Lai, K.K.: A neural-network-based nonlinear metamodeling approach to financial time series forecasting. Applied Soft Computing 9(2), 563–574 (2009)