A Unifying Framework for Learning the Linear Combiners for Classifier Ensembles

Hakan Erdogan and Mehmet Umut Sen
Faculty of Engineering and Natural Sciences,
Sabanci University, Orhanli Tuzla, 34956,
Istanbul, Turkey.
{*haerdogan,umutsen*}*@sabanciuniv.edu*

***Abstract*—For classifier ensembles, an effective combination method is to combine the outputs of each classifier using a linearly weighted combination rule. There are multiple ways to linearly combine classifier outputs and it is beneficial to analyze them as a whole. We present a unifying framework for multiple linear combination types in this paper. This unification enables using the same learning algorithms for different types of linear combiners. We present various ways to train the weights using regularized empirical loss minimization. We propose using the hinge loss for better performance as compared to the conventional least-squares loss. We analyze the effects of using hinge loss for various types of linear weight training by running experiments on three different databases. We show that, in certain problems, linear combiners with fewer parameters may perform as well as the ones with much larger number of parameters even in the presence of regularization.**

***Keywords*-classifier fusion, linear combiners, stacked generalization, linear classifier learning.**

I. Introduction

Combining multiple classifiers or experts has been a research area of interest lately. It has been observed in many different applications that fusing multiple classifier outputs improves the overall accuracy of classification. This approach is also termed as decision fusion or combining ensemble of classifiers in some works. A prominent and early example of this kind of information fusion is majority voting of multiple experts [1].

There is much interest in obtaining different classifiers that try to solve the same classification problem in a different way. Diversity across classifiers is desired since clearly there is a higher chance of success if the classifiers are independent [1]. In this paper, we are not interested in how the classifiers are chosen. Our goal is to combine available classifier results in an optimal fashion using linear combiners and learn how to combine them automatically.

For classifier combination, there may be fixed combiners or trainable combiners. Fixed combiners are sum rule, product rule, max rule, min rule, and others [2]. They are based on combining the classifier outputs with some fixed rules. One of the most successful combiners is the linear combiner (or weighted combiner) which takes a linear combination of classifier outputs to arrive at a combined decision [3]–[5]. The weight parameters of the linear combiner can be learned from data. The linear combiner was shown to outperform other trainable nonlinear combiners in [6]. As we describe clearly in this paper, there may be multiple types of linear combination [5], [7], [8]. Each combination type has its advantages and disadvantages which we explore.

Learning the linear combiner has attracted some interest in the literature. Usually, a simple sum rule (which corresponds to using equal weights for each classifier) is considered to be successful enough for satisfactory results since it is the optimal solution for unbiased independent classifiers [9], or heuristic hand-assigned weights are given to each classifier output. Although these simple approaches may be helpful in certain situations, in general we show that it is helpful to learn the weight parameters in the linear combiner. In earlier work, a secondary linear classifier is trained for combining all first level discriminants in a blind fashion using stacking [3]. Several authors observed that using a class-conscious approach may speed up the computation and improve results as well [5]–[8]. In [10], hard categorical results of classifiers are combined using Bayesian techniques which may not be trivial to generalize to continuous outputs. In [4], [11], linear combiners are analyzed theoretically without providing a method for estimating the weights.

As mentioned above, for speed and accuracy, it may not be desired to use all classifier outputs at a time and certain tying and fixing of weight factors are helpful [5]–[8]. In our work, we provide a new combined framework for all types of linear weight learning and use previously unused regularized hinge loss for learning the weights.

This paper is organized as follows. In section II, we describe the inputs to the combiner, the idea of stacked generalization, types of linear combiners and a framework to unify them. We explain how to learn the weights using regularized loss minimization in section III. Section IV discusses experimental results.

II. Problem definition

A. Input to the combiner

Assume that we have M classifiers in an N-way classification problem. For a given sample, the outputs of each classifier is a set of scores (discriminants or posterior probabilities) for each class. Namely, we get N numbers from a classifier which indicate the relative belief that the sample comes from each class. We can arrange all outputs of the classifiers in a matrix called a *decision profile matrix* for which the (i, j) entry is given by [1]:

$$D_{ij}(\boldsymbol{x}) = p_{i,j}(\boldsymbol{x})$$

where $p_{i,j}(\boldsymbol{x})$ is the score for class i in classifier j for classifying sample $\boldsymbol{x}$. The matrix is illustrated in Figure 1. We drop dependence on $\boldsymbol{x}$ in the following discussion for brevity. We can re-arrange the information in the decision profile matrix in a vector form by concatenating the columns of the $\boldsymbol{D}$ matrix. The resulting vector is of size $NM \times 1$ and we call it the vector of scores $\boldsymbol{f}$. Clearly $\boldsymbol{f}_{(j-1)N+i} = p_{i,j}$ for $i = 1, \ldots, N$ and $j = 1, \ldots, M$.

B. Stacked generalization

An important point while training the combiner is that the data used for training the first level classifiers should not contain the

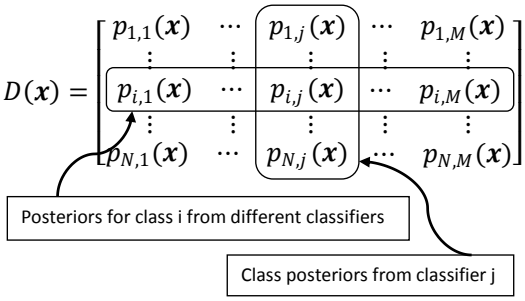

Figure 1. Decision profile matrix illustrated.

sample for which the scores are obtained. This is called stacked generalization [3]. The reason for stacked generalization comes from the fact that the combiner should consider the performance of classifiers on unseen data, not on their own training data. In practice, it may not be possible to train a new classifier for each sample by leaving out only that sample, so we perform L-fold cross validation to obtain the posteriors. We split the training data into L equi-sized parts and train L different classifiers using $L-1$ subparts at a time. For each training sample, we use the classifiers which were trained from the subparts excluding the subpart that contains the sample. During testing, we use the classifiers trained over all training data and the combiner trained using stacked generalization described above.

C. Types of weights

In its most general form, a linear combiner in this context is a set of weight vectors $\{\boldsymbol{w}_i : i = 1, \ldots, N\}$ multiplying the vector of scores $\boldsymbol{f}$. The resultant score for class i is

$$r_i = \boldsymbol{w}_i^T \boldsymbol{f}, \text{ for } i = 1, \ldots, N.$$

We choose the highest scoring class by $\hat{i} = \arg\max_i r_i$. Each $\boldsymbol{w}_i$ is of dimension MN. We form a full size column vector by concatenating the weight vectors for each class and call it $\boldsymbol{w} = [\boldsymbol{w}_1^T, \ldots, \boldsymbol{w}_N^T]^T$. $\boldsymbol{w}$ contains all the weights to be estimated and it has a dimension of MN^2. This type of linear combination is termed as *type 3* in [5] and it can be considered as a black box application of linear classification to the vector of scores $\boldsymbol{f}$. This approach is called stacking or stacked generalization in some papers [8] referring to Wolpert's work [3], although we reserve that term for the cross-validated training approach we explained above. Clearly, when the number of classes and classifiers are large, the dimensionality of the weight matrix is quite high and the resulting weights become harder to interpret. It may be hard to train these weights as well in some situations.

On the other extreme, we have *type 1* linear combination which combines discriminants of a class with a single weight per classifier. This approach is also named as the *weighted sum rule* or *weighted averaging* since it is similar to the sum-rule of combining classifiers with a weight for each classifier. Here, the discriminants of other classes are not used in obtaining the combined score of a class. This can be interpreted as weighting individual classifiers according to their confidences. In this case, the combined score is obtained as:

$$r_i = \sum_{j=1}^{M} \tilde{w}_j p_{i,j}, \text{ for } i = 1, \ldots, N.$$

Note that the weights are independent of the class, but depend only on the classifier. These weights cannot be obtained by a black box training from a reduced feature vector since each weight is multiplied with a different score for each class. The weights for this kind of combination are usually learned using grid search over validation data, or other heuristics. When the number of classifiers are large, principled ways to train these weights should be considered. This combiner has N parameters to learn. Note that, this method corresponds to tying some of the weights in the weight vector $\boldsymbol{w}$ and fixing others to a value of zero.

Another option, which is named *type 2* is to use class-dependent classifier weights, that is

$$r_i = \sum_{j=1}^{M} \tilde{w}_j^i p_{i,j}, \text{ for } i = 1, \ldots, N.$$

Here we have MN parameters to learn. Each classifier is allowed to have a different weight to achieve the combined score for each class. However, still the scores from other classes are not used, hence the weights corresponding to them are fixed to zero. This approach is termed *StackingC* in [7]. It is considered as a fast and accurate alternative to type 3 weighting [6]–[8].

D. Unifying framework

Now, we show that these three different types of linear weighting (and much more kinds that can be defined) can be combined under the same umbrella. To accomplish tying and fixing of weight parameters, we write the full weight vector as

$$\boldsymbol{w} = \boldsymbol{A}\tilde{\boldsymbol{w}} + \boldsymbol{b},$$

where $\tilde{\boldsymbol{w}}$ is what we call the unique weight vector. For type 1 and type 2, the matrix $\boldsymbol{A}$ can be obtained easily and $\boldsymbol{b} = 0$. When $N = 3$ and $M = 2$, we get the $\boldsymbol{A}$ matrices given in Figure 2. For type 3, clearly we have $\boldsymbol{w} = \tilde{\boldsymbol{w}}$ which implies $\boldsymbol{A} = I$ and $\boldsymbol{b} = 0$. If we would like to fix some values of the weight vector to some nonzero value, we can use a nonzero $\boldsymbol{b}$ entry with a corresponding row of zeros in $\boldsymbol{A}$.

$$A_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, \tilde{\boldsymbol{w}}_1 = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}, A_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tilde{\boldsymbol{w}}_2 = \begin{bmatrix} w_1^1 \\ w_2^1 \\ w_1^2 \\ w_2^2 \\ w_1^3 \\ w_2^3 \end{bmatrix}$$

Figure 2. Illustration of A matrix for type 1 and type 2 tying.

We also define $\boldsymbol{A}_k$ and $\boldsymbol{b}_k$ as the sub matrices corresponding to N consecutive rows of $\boldsymbol{A}$ and $\boldsymbol{b}$ such that

$$\boldsymbol{w}_k = \boldsymbol{A}_k\tilde{\boldsymbol{w}} + \boldsymbol{b}_k.$$

Our goal is to learn the unique weights $\tilde{\boldsymbol{w}}$ from training data.

III. Learning the weights

We propose using regularized empirical loss minimization for learning the weights [12]. Assume that we are given $(\boldsymbol{f}_i, y_i)$ pairs, for $i = 1, \ldots, n_t$, where n_t is the number of training samples, $\boldsymbol{f}_i$ is the vector of scores of the ith training sample and $y_i \in \{1, \ldots, N\}$ is the label for sample i. We focus on multi-class classification in this paper, thus $N > 2$. This approach requires to minimize a generalized objective function

$$\phi(\tilde{\boldsymbol{w}}) = \frac{1}{n_t} \sum_{i=1}^{n_t} \sum_{y=1}^{N} l(\boldsymbol{f}_i, y_i, y, \tilde{\boldsymbol{w}}) + R(\tilde{\boldsymbol{w}}). \tag{1}$$

Here $R(\tilde{\boldsymbol{w}})$ is a regularization function on $\tilde{\boldsymbol{w}}$ and can be the L_1, L_2 or L_p norms or any mixed norms [13] of the weight vector. l is the loss function incurred for classifying sample x_i as class y and is a function of the correct class y_i as well. This generalized objective function includes a lot of important linear (and nonlinear) classification methods such as binary and multi-class support vector machines (SVMs) [14]–[16] with hinge loss or squared hinge loss, the LASSO method [8], [17], logistic regression and many other linear classification methods. Regularization is important in learning. L_2 norm is essential in the SVM formulation when using the hinge loss. L_1 norm induces sparsity in the weight vector which yields desirable weights in some situations. The loss function has been traditionally the least squares error criterion, but recently superiority of other loss functions such as hinge loss or logistic loss has been shown [12].

Due to the superior performance of SVMs in many classification tasks, we focus on using the hinge loss with L_2 regularization. When the parameters are tied, this corresponds to using the following hinge loss function

$$l(\boldsymbol{f}_i, y_i, y, \tilde{\boldsymbol{w}}) = \max\left(0, 1 - \delta(y, y_i) - \boldsymbol{f}_i^T(\Delta_y^i \tilde{\boldsymbol{w}} + \boldsymbol{\beta}_y^i)\right), \tag{2}$$

where we define $\Delta_y^i = \boldsymbol{A}_{y_i} - \boldsymbol{A}_y$ and $\boldsymbol{\beta}_y^i = \boldsymbol{b}_{y_i} - \boldsymbol{b}_y$ and $\delta(y, z) = 1$ if $y = z$ and zero otherwise. We also consider the least squares (LS) loss function [5], [8] as an alternative

$$l(\boldsymbol{f}_i, y_i, y, \tilde{\boldsymbol{w}}) = \left(\delta(y, y_i) - \boldsymbol{f}_i^T(A_y \tilde{\boldsymbol{w}} + \boldsymbol{b}_y)\right)^2. \tag{3}$$

Using LS loss is equivalent to solving a regression problem [6], [17] where the targets for regression are one or zero depending on the target class. Hinge loss and LS loss are convex losses and we can implement convergent algorithms for them.

The L_2 regularization function is of the form

$$R(\tilde{\boldsymbol{w}}) = \lambda ||\boldsymbol{A}\tilde{\boldsymbol{w}} + \boldsymbol{b}||_2^2, \tag{4}$$

where λ is the regularization parameter. Assume we deal with type 3 weighting (namely, $\boldsymbol{A} = \boldsymbol{I}$ and $\boldsymbol{b} = 0$). In this case, when we use hinge loss defined in (2) and the regularization function in (4), the optimization problem is a close relative of the Crammer-Singer formulation for multi-class SVMs [14]. The difference between this algorithm and the one in [14] is that, we use a separate slack variable for each sample-class pair instead of a single slack for each sample.

In this paper, we implement a weight tying minimization of the objective function in equation (1) for LS and hinge losses and L_2 regularization. For LS, direct implementation of Newton's method works in one iteration since the objective is quadratic. For the hinge loss, we implemented a primal second order monotonic algorithm [18]. Due to space limitations and since it is outside the scope of this paper, we leave the details of weight tying algorithm implementation for the hinge loss to a broader version of this work. We provide results with the type 1, type 2 and type 3 linear combiners trained using the LS and hinge losses in the experiments section which follows.

IV. Experiments

We experimented with the linear combiner on three different multi-class databases from the UCI repository [19]. Our results are compared with using equal weights (sum rule). We used UCI statlog satellite (sat) data set, UCI image segmentation data set (seg) and UCI optdigit data sets (opt)[1]. For constructing base classifiers, we used 2 different scenarios. In the first scenario, for obtaining complementary information between classifiers, we trained each classifier from a subset of training data, in which, instances that belongs to a particular class are the majority. So, each classifier is a different class' expert, so the classifiers have complementary information. Using stacked generalization, posterior probabilities are obtained for each data instance. For the second scenario, we trained six different types of classifiers from the whole training data. The second scenario was helpful to compare the usefulness of different combination types when base classifiers use the same training data. We may say that the second scenario has more correlated classifiers since they all use the same data for training.

The results for scenario one are presented in Table I. EW denotes *equal weights* and corresponds to the sum rule result. In all cases, we can get improvement upon equal weights by learning the weights from training data. The results indicate the best testing accuracy obtained during the iterations of the algorithm after grid searching for the best λ parameter over the test data, hence a bit optimistic. Hinge loss function yields better rates than the LS loss function in general. Among types 1, 2 and 3, for the hinge loss, we obtain the best results for type 2 over types 1 and 3. This shows that even with regularization, it is beneficial to structurally regularize the weight vector by using class-dependent weights. This is due to the nature of the classifier combination problem and it is interesting to observe it. For the LS loss, type 3 generally performs better than type 1 or 2. However, since LS loss yields worse results as compared to the hinge loss in general, we can say that type 2 is a good choice. For the segment database, LS loss yields an extremely low accuracy rate for type 1. We attribute this to using little amount of data in that database for some classifiers and limited set of weights to adjust for the loss.

Results for scenario two are provided in Table II. Here, we have six different classifiers trained on the same data. The results are somewhat different from scenario one results. Here, we have types 1 and 3 usually perform better than type 2. In this scenario, it appears type 1 weights which do not discriminate between classes and only weight classifiers yield more stable weights as compared to type 2. This is understandable since the classifiers do not favor any class during their training and using class-dependent weights

[1]UCI statlog satellite data set has a training set of 4435, a test set of 2000 with 6 classes and 36 variables. UCI optdigit data set has a training set of 3823, a test set of 1797, 10 classes and 64 variables. UCI image segmentation data set has a training set of 210, a test set of 2100, 7 classes and 19 variables.

do not help as much. In general, type 3 is better than type 1 as well since it has more parameters to tune and with proper amount of regularization, we can get good results. For the UCI optdigits database, LS loss surprisingly yields the best result with type 3, but for other databases, we consistently get the best result using the hinge loss.

Table I
LINEAR COMBINER ACCURACY RESULTS IN SCENARIO ONE.

DB	EW	LS			HINGE		
		type1	type2	type3	type1	type2	type3
opd	96.7	96.82	98.44	98.44	97.49	98.72	98.60
sat	89.7	88.95	89.45	90.20	89.95	90.90	90.20
seg	80.4	17.57	73.04	82.19	82.19	85.55	81.33

Table II
LINEAR COMBINER ACCURACY RESULTS IN SCENARIO TWO.

DB	EW	LS			HINGE		
		type1	type2	type3	type1	type2	type3
opd	95.1	95.93	94.65	97.44	97.21	96.04	96.82
sat	86.2	83.65	85.40	91.70	91.10	91.15	92.00
seg	91.7	92.04	91.52	91.66	92.19	91.95	92.38

V. CONCLUSION

We have introduced methods for linear combination of classifier outputs. We handle different types of linear combination under the same umbrella by introducing a tying/fixing matrix and a vector. This framework enables one to estimate weight parameters which are related to each other in various ways which we have not explored to the full potential yet and plan to explore further in our future work. We learn weights of types 1, 2 and 3 using the introduced method by minimizing a regularized empirical loss function. We have experimented with hinge loss and LS loss on three different databases and under two different scenarios and show that hinge loss yields better results in general. We plan to perform more extensive experiments in the future. We also explain the results we get and conclude that each type of weight combination can be beneficial in certain cases and that one needs to consider the needs of their own problem and choose an appropriate method accordingly.

REFERENCES

[1] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.

[2] J. Kittler, "Combining classifiers: A theoretical framework," vol. 1, no. 1, pp. 18–27, 1998.

[3] D. H. Wolpert, "Stacked generalization," *Neural Netw.*, vol. 5, no. 2, pp. 241–259, 1992.

[4] G. Fumera and F. Roli, "A theoretical and experimental analysis of linear combiners for multiple classifier systems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 6, pp. 942–956, June 2005.

[5] N. Ueda, "Optimal linear combination of neural networks for improving classification performance," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 2, pp. 207–215, 2000.

[6] K. M. Ting and I. H. Witten, "Issues in stacked generalization," *Journal of Artificial Intelligence Research*, vol. 10, pp. 271–289, 1999.

[7] A. K. Seewald, "How to make stacking better and faster while also taking care of an unknown weakness," in *ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 554–561.

[8] S. Reid and G. Grudic, "Regularized linear models in stacked generalization," in *MCS '09: Proceedings of the 8th International Workshop on Multiple Classifier Systems*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 112–121.

[9] K. Tumer and J. Ghosh, "Linear and order statistics combiners for pattern classification," in *Combining Artificial Neural Nets*. Springer-Verlag, 1999, pp. 127–162.

[10] Z. Ghahramani and H. chul Kim, "Bayesian classifier combination," 2003.

[11] G. Fumera and F. Roli, "Linear combiners for classifier fusion: Some theoretical and experimental results," in *In Proc. Int. Workshop on Multiple Classifier Systems (LNCS 2709*. Springer, 2003, pp. 74–83.

[12] Y. Lecun, S. Chopra, R. Hadsell, F. J. Huang, and M. A. Ranzato, *A Tutorial on Energy-Based Learning*. MIT Press, 2006.

[13] J. Duchi and Y. Singer, "Efficient learning using forward-backward splitting," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds., 2009, pp. 495–503.

[14] K. Crammer and Y. Singer, "Ultraconservative online algorithms for multiclass problems," *Journal of Machine Learning Research*, vol. 3, p. 2003, 2001.

[15] S. S. Keerthi, S. Sundararajan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin, "A sequential dual method for large scale multi-class linear svms," in *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2008, pp. 408–416.

[16] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *J. Mach. Learn. Res.*, vol. 9, pp. 1871–1874, 2008.

[17] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning*, corrected ed. Springer, July 2003. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0387952845

[18] H. Erdogan, "Ongoing work on monotonic algorithms for svms," 2010.

[19] A. Asuncion and D. Newman, "UCI machine learning repository," 2007. [Online]. Available: http://www.ics.uci.edu/∼mlearn/MLRepository.html