ATLANTIS PRESS
International Journal of Computational Intelligence Systems, Vol. 11 (2018) 962–978

Sparsity-driven weighted ensemble classifier

Atilla Özgür [1] Fatih Nar [2] Hamit Erdem [3]

[1] *Logistics Engineering, Jacobs University,*
Campus Ring 1 28759 Bremen, Germany
E-mail: a.oezguer@jacobs-university.de

[2] *Computer Engineering, Konya Food and Agriculture University,*
Dede Korkut Mah. Beyşehir Cad. No:9 ,
Meram / Konya / Turkey
E-mail: fatih.nar@gidatarim.edu.tr

[3] *Electrical Engineering, Başkent University,*
Bağlıca Kampüsü Fatih Sultan Mahallesi Eskişehir Yolu 18. km ,
Ankara 06790, Turkey
E-mail: herdem@baskent.edu.tr

Received 20 April 2017

Accepted 5 April 2018

Abstract

In this study, a novel sparsity-driven weighted ensemble classifier (SDWEC) that improves classification accuracy and minimizes the number of classifiers is proposed. Using pre-trained classifiers, an ensemble in which base classifiers votes according to assigned weights is formed. These assigned weights directly affect classifier accuracy. In the proposed method, ensemble weights finding problem is modeled as a cost function with the following terms: (a) a data fidelity term aiming to decrease misclassification rate, (b) a sparsity term aiming to decrease the number of classifiers, and (c) a non-negativity constraint on the weights of the classifiers. As the proposed cost function is non-convex thus hard to solve, convex relaxation techniques and novel approximations are employed to obtain a numerically efficient solution. Sparsity term of cost function allows trade-off between accuracy and testing time when needed. The efficiency of SDWEC was tested on 11 datasets and compared with the state-of-the art classifier ensemble methods. The results show that SDWEC provides better or similar accuracy levels using fewer classifiers and reduces testing time for ensemble.

Keywords: Machine Learning, Ensemble, Convex Relaxation, Classification, Classifier Ensembles

1. Introduction

The accuracy of classification can be improved by using more than one classifier. This process is known by different names in different domains such as classifier fusion, classifier ensemble, classifier combination, mixture of experts, committees of neural networks, or voting pool of classifiers etc [1].

Ensembles can be categorized as weak or strong depending on the used classifier type [2]. The weak classifiers use machine learning algorithms with fast training times and lower classification accuracy. Due to fast training times, weak classifier ensembles contain high number of classifiers, such as 50–200 classifiers. On the other hand, strong classifiers have slow training times and high gener-

alization accuracy individually. Due to slow training times, strong classifier ensembles contain as low as 3–7 classifiers.

Although using more classifiers increases generalization performance of ensemble classifier, this degrades after a while. To put it in another way, similar classifiers do not contribute to overall accuracy very much. This deficiency can be removed by increasing the classifier diversity [1, 3, 4]. Therefore, finding new diversity measurements [5] and improving existing ones [4] are an ongoing research effort in ensemble studies.

Research in the ensembles can be categorized into two groups according to their construction methods: (a) Combining pre-trained classifiers. (b) Constructing classifiers and ensemble together.

Methods in the first group (a) are the easiest to understand and the mainly used methods to create ensembles. The classifiers are trained using training set and combined in an ensemble. The simplest method to ensemble classifiers is majority (plurality) voting. In the majority voting method, every classifier in an ensemble gets a single vote for result. The output is the most voted result. A well-known approach that uses majority voting in its decision stage is Bootstrap aggregating algorithm (Bagging) [6]. Bagging trains weak classifiers from same dataset using uniform sampling with replacement, then classifiers are combined using simple majority voting [7]. Instead of using a single vote for every classifier, weighted voting might be used [7]. Standard Weighted majority voting (WMV) algorithm [7] uses accuracy of individual classifiers for finding weights. Classifiers that have better accuracies in training step get higher weights for their votes, and become more effective in voting.

Kuncheva and Rodriguez [8] proposed a probabilistic framework for classifier ensembles. This framework shows relationships between four combiners: majority voting, weighted voting, recall voting, and naive bayes voting. According to the experiments of Kuncheva and Rodríguez [8] on 73 benchmark datasets, there is no definite best combiner among those four. These results conform to "no free lunch theorem" [9]. No universal classifier exists that is suitable for every problem. Numerous other methods has been proposed for finding weights to combine pre-trained classifiers, Table 1. Methods in Table 1 are also summarized in Section 1.1. Similar to approaches in Table 1, main focus of this study is to present a new approach for finding weights in an ensemble that uses pre-trained classifiers using convex optimization techniques.

In the second categorization (b), ensemble construction and classifier construction affect each other. Adaboost [10] is a well known example for this categorization that trains weak classifiers iteratively and adds them to ensemble. Different from bagging, subset creation is not randomized in boosting. At each iteration, subsets are obtained using results of previous iterations. That is miss-classified data in previous subsets are more likely included. In classifier ensemble, standard weighted majority voting is used.

Gurram and Kwon [11] used similar approach to classify remote sensing images. Randomly selected features were used to train weak SVM classifiers. Optimization process of training and combination of these classifiers were done together. Lee et al. [12] combined neural network weak classifiers in their ensemble. Genetic algorithms were used for finding weights for neural network neurons and increase diversity among neural networks. Then, these diverse neural networks were combined using negative correlation rule. Neural networks were trained and combined in one step. Tian and Fend [13] proposed an approach that combines feature sub-selection and ensemble optimization. They proposed three-term cost function: a classification accuracy term, a diversity term and a feature size term. They solved this ensemble cost function using population based heuristics optimization. Zhang et al. [14] used Kernel sparse representation based classifiers for ensemble in face recognition domain. Features were projected to higher dimensions using kernels, then sparse representation of these features were found using optimization techniques. Similarly, Kim et al. [15] proposed ensemble approach for biological data. Their approach were similar to boosting but they selected sparse features in their weak classifiers. Özgür and Erdem [16] used Genetic algorithms to select features and find weights for clas-

Table 1: Ensemble weights finding studies that use pre-trained classifiers

Study	Year	Classifiers	Method	Size	Sparse	Cost Function	Regularizer	Notes
Sylvester and Chawla [17]	2006	12 Different Classifiers	Genetic Algorithms	120	No	No Information	No Information	
Li and Zhou [18]	2009	Decision Tree	Quadratic Programming	100	Yes	Hinge Loss	L_1	
Kim et al. [19]	2011	Decision Tree	Matrix Decomposition	64	No	Indicator Loss	No Regularization	
Mao et al. [20]	2011	Decision Tree, SVM[1]	Matching Pursuit	100	Yes	Sign Loss	No Regularization	
Zhang and Zhou [21]	2011	KNN[2]	Linear Programming	100	Yes	Hinge Loss	L_1	
Goldberg and Eckstein [22]	2012	No Information	Linear Programming	NI	Yes	Indicator Loss	L_0	[a]
Santos et al. [23]	2012	SVM,[1] MLP[3]	Genetic Algorithms	6	No	No Cost Function	No Regularization	
Yin et al. [24]	2012	Neural Networks	Genetic Algorithms	100	Yes	Square Loss	L_1	[b]
Meng and Kwok [25]	2013	Decision Tree, SVM,[1] KNN[2]	Domain Heuristic	3	No	No Cost Function	No Regularization	
Tinoco et al. [26]	2013	SVM,[1] MLP[3]	Linear Programming	6	Yes	Hinge Loss	L_1	[d]
Hautamaki et al. [27]	2013	Logistic Regression	Nelder–Mead	12	Yes	cross-entropy [28]	$L_1, L_2, L_1 + L_2$	[c]
Şen and Erdoğan [29]	2013	13 Different Classifiers	Convex Opt.	130	Yes	Hinge Loss	L_1 , Group Sparsity	
Mao et al. [30]	2013	Decision Tree	Singular Value Decomposition	10	No	Absolute Loss	No Regularization	
Yin et al. [31]	2014	Neural Networks	Genetic Algorithms	100	Yes	Square Loss	L_1	[e]
Yin et al.[32]	2014	Neural Networks	Quadratic Programming	100	Yes	Square Loss	L_1	[f]
Zhang et al. [3]	2014	5 Different Classifiers	Differential Evolution	5	No	No Cost Function	No Regularization	
Mao et al. [33]	2015	Decision Tree	Quadratic Form	200	No	Square Loss	L_1	

[1] SVM Support Vector Machines.
[2] KNN K-Nearest Neighbor
[3] MLP Multi Layer Perceptron.
[a] No experimental results.
[b] Diversity Term Yule's Q Statistic is used
[c] Improved version of [23]
[d] 3 regularizers are compared
[e] Journal version of [24]
[f] Convex Quadratic model of [31] and [24]

sifier ensemble in their study. They combined different strong classifiers and experimented on NSL-KDD dataset.

1.1. Related works: ensembles that combine pre-trained classifiers

Focus of this study is to combine pre-trained classifiers so that combined accuracy of the ensemble is better than individual classifiers. This study aims to accomplish this objective in a sparse manner so that not all of the classifiers are used in ensemble; therefore, weak decision tree classifiers are used in the experiments. Although some of the other sparse approaches [11, 14, 15, 34] are mentioned before, in this section, only ensemble classifiers that proposed methods to find weights for base classifiers are investigated.

Sylvester and Chawla [17] proposed differential evolution to find suitable weights for ensemble base classifiers. Similar to most heuristic solution techniques, they did not explicitly define cost function, but use classification accuracy for fitness function. ID3 decision trees, J48 decision trees (C4.5), JRIP rule learner (Ripper), Naive Bayes, NBTree (Naive Bayes trees), One Rule, logistic model trees, logistic regression, decision stumps, multi-layer perceptron (MLP), SMO (support vector machine), and 1BK (k-nearest neighbors) classifiers from Weka toolbox [35] were used in the experiments.

Li and Zhou [18] modeled ensemble weights finding problem using cost function that consists of hinge loss and L_1 regularization. This cost function were minimized using Quadratic programming. Decision tree weak classifiers and UCI datasets were used for experiments. A semi-supervised version was also suggested.

Zhang and Zhou [21] formulated weights finding problem using three different cost functions: LP1 uses a cost function that consists of Hinge loss only. LP2 uses a cost function that consists of Hinge loss and L_1 regularization. LP3 allows weights to be negative. These cost functions were minimized using linear programming. They used K-Nearest neighbor (KNN) algorithm as base classifiers and UCI datasets in their experiments.

Kim et al. [19] proposed an approach similar to boosting. They considered two weight vectors, one for classifier and one for instances. Hard to classify instances get more weight and correspondingly they affect weight vector more. Different from boosting, their approach works with pre-trained classifiers. Weights for ensemble was found using matrix decomposition and an iterative algorithm. Decision tree weak classifiers and UCI datasets were used for experiments.

Mao et al. [20] proposed matching pursuit al-

gorithm to find weights for ensemble base classifiers. Since matching pursuit is a sparse approximation algorithm [36], their approach include sparsity. Decision Tree and SVM weak classifiers and UCI datasets were used for experiments.

Goldberg and Eckstein [22] modeled weights finding problem with indicator loss function and L_0 regularization function. According to Goldberg and Eckstein [22], this problem is NP-hard in special cases. They gave different relaxation strategies to solve this problem and gave their relaxation bounds. Different from other studies, this study was purely theoretical.

Santos et al. [23] combined MLP and SVM algorithms to classify remote sensing images. They did not give any explicit cost function but used genetic algorithms for finding weights. An improved version of their studies [26] modelled weights finding problem using hinge loss and L_1 regularization. This cost function were minimized using linear programming. In both versions, remote sensing images were classified using ensemble of MLP and SVM classifiers.

Mong and Kwok [25] combined Decision Tree(J48), K-Nearest Neighbor and SVM classifiers. They suggest using following domain heuristic for weights of classifiers: "...weighted ranking (precision of false alarm $>$ recall of false alarm $>$ classification accuracy) is an appropriate and correct way to decide the weight values with high confidence in ensemble selection..." [25].

Hautamaki et al. [27] investigated using sparse ensemble in speaker verification domain. Ensemble weights finding problem were modeled using cross-entropy loss function and three different regularization functions: L_1, L_2, and L_1+L_2. These cost functions were minimized using Nelder–Mead method. Logistic regression classifiers were used in experiments.

Yin et al. [24] modeled ensemble weights finding problem with a cost function that consists of the terms a square loss, L_1 regularization and a diversity based-on Yule's Q statistics. They used neural network classifiers on 6 UCI datasets in their experiments. In their first study [24], the proposed cost function were minimized using genetic algorithms. In their second study [31], the Pascal 2008 webspam dataset were added to their experiments. Finally, convex optimization techniques [32] were used to minimize the same cost function.

Sen and Erdogan [29] modeled ensemble weights finding problem using a cost function that consists Hinge loss and two different regularization functions, L_1 and group sparsity. This cost function were minimized using convex optimization techniques. In their experiments, 13 different classifiers were compared on 12 UCI datasets and 3 other datasets using CVX Toolbox [37, 38].

Zhang et al. [3] proposed Differential Evolution for finding suitable weights for ensemble base classifiers. Similar to most heuristic solution techniques, they did not explicitly define cost function, but use classification accuracy for fitness function. Decision Tree (J4.8), Naive Bayes, Bayes Net, K-Nearest Neighbor, and ZeroR classifiers from Weka toolbox [35] were used in the experiments.

Mao et al. [30] modeled ensemble weights finding problem using a cost function that consists of absolute loss only. Proposed cost function was minimized using 0–1 matrix decomposition. In a later study [33], Mao et al. proposed a cost function that consists of square loss and L_1 regularization function. This cost function was minimized using quadratic form approximation. Both studies used decision tree weak classifiers and UCI datasets in experiments.

As can be seen from Table 1, numerous approaches exist for finding weights in ensemble classification. Inspired from studies of [16, 21, 30, 33, 39], sparsity-driven weighted ensemble classifier (SDWEC) has been proposed. SDWEC can use both strong classifiers or weak classifiers for classifier ensemble. In this study, decision tree as a weak classifier is used as the base classifier since choosing fewer number of classifiers among large number of weak classifiers leads to high accuracy with shorter testing time. Proposed cost function consists of the following terms: (1) a data fidelity term with sign function aiming to decrease misclassification rate, (2) L_1-norm sparsity term aiming to decrease the number of classifiers, and (3) a non-negativity constraint on the weights of the classifiers. Cost func-

tion proposed in SDWEC is hard to solve since it is non-convex and non-differentiable; thus, (a) the sign operation is convex relaxed using a novel approximation, (b) the non-differentiable L_1-norm sparsity term and the non-negativity constraint are approximated using *log-sum-exp* and Taylor series. Contributions of SDWEC can be summarized as follows:

1. A new cost function is proposed for ensemble weights finding problem.
2. This cost function is minimized using novel convex relaxation and approximation techniques for sign function and absolute value function.
3. SDWEC provides similar or better classification accuracy, while minimizing the number of classifiers used in the ensemble.
4. According to sparsity level of SDWEC, number of classifiers used in the ensemble decreases; thus, the testing time for whole ensemble decreases.
5. The sparsity level of SDWEC allows trade-off between testing accuracy and testing time when needed.
6. Computational Complexity of SDWEC is analyzed theoretically and experimentally, which is linear in number of data rows, number of classifiers and number of algorithm iterations.

2. Sparsity-driven weighted ensemble classifier

An ensemble consists of l number of classifiers which are trained using training dataset. We aim to increase ensemble accuracy on test dataset by finding suitable weights for classifiers using validation dataset. Ensemble weights finding problem is modeled with the following matrix equation.

$$\underbrace{sgn\left(\begin{bmatrix} -1 & -1 & \dots & +1 \\ +1 & -1 & \dots & -1 \\ \vdots & \vdots & \vdots & \vdots \\ -1 & \dots & \dots & +1 \\ +1 & \dots & \dots & -1 \end{bmatrix}\right.}_{\mathbf{H}_{mxl}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{l-1} \\ w_l \end{bmatrix}}_{w_{lx1}} \underbrace{\left.\right)}_{} \approx \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{m-1} \\ y_m \end{bmatrix}}_{y_{mx1}}$$

$\mathbf{H}$	classifiers results $\{-1,1\}^{mxl}$
m	number of samples in the validation dataset
l	number of individual classifiers
w	classifier weights
y	true labels $\{-1,1\}^{mx1}$ for the validation dataset

In this matrix equation, classifiers predictions are weighted so that obtained prediction for each data row becomes approximately equal to expected results. Matrix $\mathbf{H}$ consists of l classifier predictions for m data rows that are drawn from validation dataset. Vector y contains the labels for the validation dataset. Our aim is to find suitable weights for w in a sparse manner while preserving condition of $sgn(\mathbf{H}w) \approx y$ (sign function). For this model, the following cost function is proposed:

$$J(w) = \frac{\lambda}{m}\sum_{s=1}^{m}(sgn(H_s w) - y_s)^2 + \frac{1}{l}||w||_1^1 \quad \text{subject to } w \geqslant 0 \tag{1}$$

λ	data fidelity coefficient ($\lambda > 0$)
H_s	s_{th} row vector of matrix $\mathbf{H}$
y_s	s_{th} label for vector y

In equation 1, the first term acts as a data fidelity term and minimizes the difference between true labels and ensemble predictions. Base classifiers of ensemble give binary predictions (-1 or 1) and these predictions are multiplied with weights through sign function which leads to $\{-1,0,1\}$ as an ensemble result. To make this term independent from data size, it is divided to m (number of data rows).

The second term is a sparsity term [40] that forces weights to be sparse [39]; therefore, minimum number of classifiers are utilized. In sparsity term, any L_p-norm ($0 \leqslant p \leqslant 1$) can be used. Weights become more sparse as p gets closer to 0.

However, when $(0 \leqslant p < 1)$, sparsity term becomes non-convex and thus the problem becomes harder to solve. When p is 0 (L_0-norm) then problem becomes NP-hard [41]. Here, L_1-norm is used as a convex relaxation of L_p-norm [40, 42]. Similar to the data fidelity term, this term is also normalized with division by l (number of individual classifiers).

The third term is used as a non-negativity constraint. Since base binary classifiers use values of -1 and 1 for class labels, negative weights change sign of prediction; thus they change class label of prediction. To avoid this problem, the constraint term is added to force weights to be non-negative.

Using the $|x| = max(-x,x)$ as the definition of L_1-norm and using the penalty method [43] for transforming the constraint in the equation 1 to a penalty term (i.e. $w \geqslant 0 \rightarrow max(-w_r,0), 1 \leqslant r \leqslant l$), below unconstrained cost function is obtained:

$$\begin{aligned} J^{(n)}(w) = & \frac{\lambda}{m}\sum_{s=1}^{m}(sgn(H_s w) - y_s)^2 \\ & + \frac{1}{l}\sum_{r=1}^{l} max(-w_r, w_r) \\ & + \frac{\beta^{(n)}}{l}\sum_{r=1}^{l} max(-w_r, 0) \end{aligned} \tag{2}$$

In equation 2, n is the iteration number since constrained cost function in equation 1 is converted into series of unconstrained problems using penalty method. Due to employed penalty method approach, the constraint $w \geqslant 0$ is better satisfied as the penalty coefficient $\beta^{(n)}$ is increased in each iteration where $\beta^{(1)} > 0$ in the first iteration. Equation 2 is a non-convex function, since *sgn* function creates jumps on cost function surface. In addition, *max* function is non-differentiable. Functions *max* and *sgn* in Equation 2 are hard to minimize. Therefore, we propose a novel convex relaxation for *sgn* as given in equation 3. Figure 1 shows approximation of sign function using Equation 3.

$$sgn(H_s w) \approx \frac{H_s w}{|H_s \widehat{w}| + \varepsilon} = S_s H_s w \tag{3}$$

where

$$S_s = (|H_s \widehat{w}| + \varepsilon)^{-1} \tag{4}$$

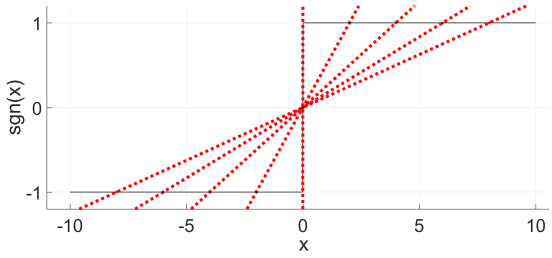

Figure 1: Sign function approximation using equation 3. Dotted Lines are approximations using Equation 3 at various points.

In this equation, ε is a small positive constant. We also introduce a new constant $\widehat{w}$ as a proxy for w. Therefore, $S_s = (|H_s \widehat{w}| + \varepsilon)^{-1}$ is also a constant. However, this *sgn* approximation is only accurate around introduced *constant* $\widehat{w}$. Therefore, the approximated cost function needs to be solved around $\widehat{w}$. Additionally, *max* function is approximated with *log-sum-exp* [44] as follows:

$$max(-w_r, w_r) \approx \frac{1}{\gamma} log(e^{-\gamma w_r} + e^{\gamma w_r}) \tag{5}$$

Accuracy of *log-sum-exp* approximation becomes better as γ, a positive constant, increases. In double-precision floating-point format [45], values up to 10^{308} in magnitude can be represented. This means that $\gamma|w_r|$ should be less than 710 where $exp(709) \approx 10^{308}$, otherwise exponential function will produce infinity (∞). At $w_r = 0$, there is no danger of numerical overflow in exponential terms of a *log-sum-exp* approximation; thus, large γ values can be used. But as $|w_r|$ gets larger, there is a danger of numerical overflow in exponential terms of *log-sum-exp* approximation, since $e^{\gamma|w_r|}$ may be out of double precision floating point upper limit.

To remedy this numerical overflow issue, a novel adaptive γ approximation is proposed, where γ_r is adaptive form of γ and defined as $\gamma_r = \gamma(|\widehat{w}_r| + \varepsilon)^{-1}$. The accuracy of approximation can be improved by decreasing ε or increasing γ. Figure 2 shows proposed adaptive γ and resulting approximations for

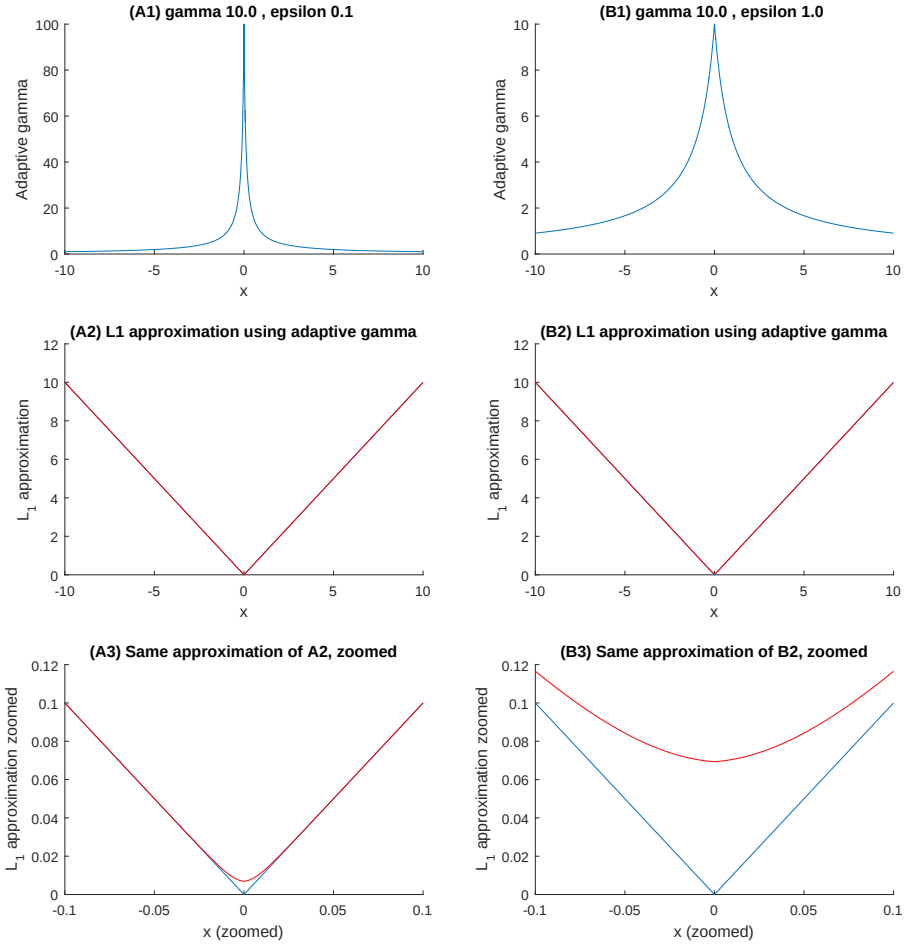

Figure 2: Adaptive gamma (γ_1) L_1 Approximation with different ε values.

two different set of values $(\gamma = 10, \varepsilon = 0.1)$ and $(\gamma = 10, \varepsilon = 1)$.

Validity of the approximation can be checked by taking the limits at $-\infty$, 0, and $+\infty$ with respect to w_r. These limits are $-x$, $\frac{\varepsilon \log 2}{\lambda_r}$, and x when w_r goes to $-\infty, 0$, and $+\infty$. As $|x|$ gets larger, dependency to γ decreases; thus, proposed adaptive γ approximation is less prone to numerical overflow compared to standard *log-sum-exp* approximation.

Regularization term given in equation 6 is added to the unconstrained cost function (equation 2) since approximated cost function needs to be solved around $\widehat{w}$. This new regularization term forces solution to be around $\widehat{w}$ by imposing a quadratic penalty between solution and $\widehat{w}$. Due to this new regularization term, solution in each iteration will be changed slowly; thus, this new term is called as slow-step regularization. The main drawback of penalty method is the need to increase penalty coefficient in each iteration, theoretically up to infinity, that leads to ill-conditioning in the minimization of the cost function. However, increase of $\beta^{(n)}$ in each iteration is not needed since during the minimization changes in the solution will be small. These small changes is accomplished due to employed slow-step regularization. Therefore, penalty coefficient is used as a constant β with a small value (i.e $\beta < 10^2$) for all iterations. Note that, using a small value for penalty coefficient β leads to numerically well-posed minimization problem.

$$\frac{1}{l}\sum_{r=1}^{l}(w_r-\widehat{w}_r)^2 \tag{6}$$

Application of adaptive γ approximation leads to the following equations:

$$max(-w_r,w_r)\approx\frac{1}{\gamma_r}log(e^{-\gamma_r w_r}+e^{\gamma_r w_r}) \tag{7}$$

$$\beta max(-w_r,0)\approx\frac{\beta}{\gamma_r}log(e^{-\gamma_r w_r}+1)=P(w_r) \tag{8}$$

Use of slow-step regularization in equation 6 and *log-sum-exp* approximation with adaptive γ leads to the cost function shown in equation 9.

$$\begin{aligned}J^{(n)}(w)=&\frac{\lambda}{m}\sum_{s=1}^{m}(S_sH_sw-y_s)^2\\&+\frac{1}{l}\sum_{r=1}^{l}\frac{1}{\gamma_r}log(e^{-\gamma_r w_r}+e^{\gamma_r w_r})\\&+\frac{1}{l}\sum_{r=1}^{l}\frac{\beta}{\gamma_r}log(e^{-\gamma_r w_r}+1)\\&+\frac{1}{l}\sum_{r=1}^{l}(w_r-\widehat{w}_r)^2\end{aligned} \tag{9}$$

In order to achieve a second-order accuracy and to obtain a linear solution, after taking the derivative of the cost function, equation 9 is expanded as a second-order Taylor series centered on $\widehat{w}_r$, leading to equation 10.

$$\begin{aligned}J^{(n)}(w)=&\frac{\lambda}{m}\sum_{s=1}^{m}(S_sH_sw-y_s)^2\\&+\frac{1}{l}\sum_{r=1}^{l}(A_r+B_rw_r+C_rw_r^2)\\&+\frac{1}{l}\sum_{r=1}^{l}(w_r-\widehat{w}_r)^2\end{aligned} \tag{10}$$

In equation 10, A_r represents constants terms while B_r and C_r are the coefficients of the terms w_r and w_r^2, respectively. If w_r values differ significantly from constant point, $\widehat{w}_r$, Taylor approximation diverges from true cost function. In proposed method, employed slow-step regularization also ensures the accuracy of Taylor approximations.

Equation 10 can be written in a matrix-vector form as follows:

$$\begin{aligned}J^{(n)}(w)=&\frac{\lambda}{m}(\mathbf{SH}w-y)^\intercal(\mathbf{SH}w-y)\\&+\frac{1}{l}(v_A^\intercal\vec{1}+v_B^\intercal w+w^\intercal\mathbf{C}w)\\&+\frac{1}{l}(w-\widehat{w})^\intercal(w-\widehat{w})\end{aligned} \tag{11}$$

$\mathbf{S}$	matrix form of S_s
$\vec{1}$	vector of ones
v_A	vector form of A_r
v_B	vector form of B_r
$\mathbf{C}$	diagonal matrix form of C_r

Equation 11 is strictly convex (see appendix for the details) thus it has a unique global minimum. Therefore, to minimize $J^{(n)}(w)$ in Equation 11, the derivative with respect to w is taken and is equalized to zero. This leads to system of linear equations:

$$\begin{aligned}&\mathbf{M}w=b\\&where\\&\mathbf{M}=\frac{2\lambda}{m}(\mathbf{SH})^\intercal(\mathbf{SH})+\frac{2}{l}(\mathbf{C}+\mathbf{I})\\&b=\frac{2\lambda}{m}(\mathbf{SH})^\intercal y+\frac{2\widehat{w}-v_B}{l}\end{aligned} \tag{12}$$

In Equation 12, $\mathbf{M}$ is dense, symmetric, real, and positive definite matrix with size of $l\times l$.

Final model is solved using algorithm 1 iteratively. Due to the employed numerical approximations and using constant β, small negative weights may occur around zero. Since our feasible set is $w\geqslant 0$, back projection to this set is performed after solving linear system at each iteration in algorithm 1. This kind of back-projection to feasible domain is commonly used [46]. Additionally, small weights in ensemble do not contribute to overall accuracy; therefore, these small weights are thresholded after iterations are completed.

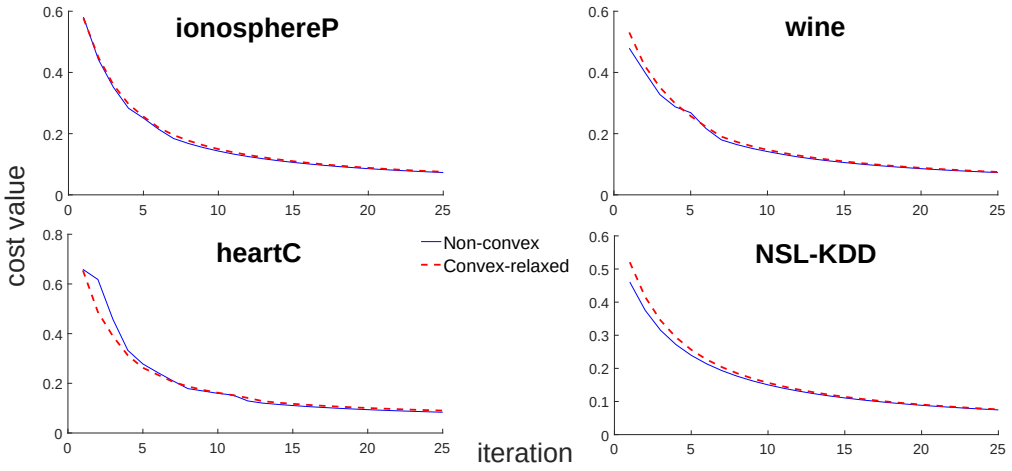

Figure 3: Cost function minimization for 4 datasets (Non-convex equation 2 vs convex-relaxed equation 11).

Algorithm 1 SDWEC Pseudo code

$\mathbf{H}, y, \lambda, \beta, \gamma, \varepsilon$ are initialized
$w \leftarrow \vec{1}$
$m, l \leftarrow size\ of\ \mathbf{H}_{mxl}$
$k \leftarrow 25$ ▷ Maximum Iteration
for n = 1 to k **do**
  $\widehat{w} \leftarrow w$
  $\gamma_r \leftarrow \frac{\gamma}{|\widehat{w}|+\varepsilon}$
  construct $\mathbf{S}$ as diagonal form of S_s
  construct v_B and $\mathbf{C}$
  $\mathbf{M} \leftarrow \frac{2\lambda}{m}(\mathbf{SH})^{\intercal}(\mathbf{SH}) + \frac{2}{l}(\mathbf{C}+\mathbf{I})$
  $b \leftarrow \frac{2\lambda}{m}(\mathbf{SH})^{\intercal} y + \frac{2\widehat{w}-v_B}{l}$
  solve $\mathbf{M}w = b$
  $w = max(w, 0)$ ▷ Back projection to $w \geqslant 0$
end for
$w_{threshold} = argmin_{w_r}(P(w_r) - 10^{-3})^2$
$w = \begin{cases} w & if\ w > w_{threshold} \\ 0 & otherwise \end{cases}$

An example run of Algorithm 1 can be seen in Figure 3, where cost values for equations 2 and 11 decrease steadily. As seen in Figure 3, the difference between non-convex cost function and its convex relaxation is minimal especially in the final iterations. This shows that two functions converge to very similar values. Since non-convex Equation 2 and convex Equation 11 are converged to similar points, this converged points are within close proximity of the global minimum. Non-convex Equation 2 and convex-relaxed Equation 11 are close to each other due to the slow-step regularization term and employed iterative approach for numerical minimization. These results show success of the proposed approximations.

3. Experimental results

The performance of SDWEC has been tested on 11 datasets; 10 UCI datasets and NSL-KDD [47]. NSL-KDD is a popular database for intrusion detection [47, 48, 49]. In all ensemble methods, 200 base decision tree classifiers, Classification And Regression Trees (CART) [50] are used. SDWEC has been compared with the following algorithms : Single decision tree classifier (CART) [50], bagging [6], WMV [7], and state-of-the-art ensemble QFWEC [33]. Each dataset is divided to training (80%), validation (10%), and testing (10%) datasets. This process has been repeated 10 times for cross validation. Mean values have been used in Table 2. The accuracy values for QFWEC in Table 2 are higher than original publication [33] since weights are found using validation dataset instead of training dataset, which provides better generalization.

SDWEC finds weights of ensemble for pre-trained classifiers; thus, it is divided into 3 sub tasks.

1. *Training base classifiers on training dataset*: This sub task is common for the ensemble methods which aims to combine pre-trained classifiers. Employed pre-trained classifier can be a weak classifier or a strong classifier where generally weak classifiers are faster to train with lower accuracy and strong classifiers are slower to train with higher accuracy. Training time of base classifiers depends on training complexity of the method which is dependent to the number of data in the training dataset (p), number of features (d), number of classes (i.e. binary, multi-label), and data characteristics. Computational (time) complexity of base classifier training are independent from the proposed SDWEC method; thus, one can use a base classifier of his choice. SDWEC aims to use few number of classifiers among trained l base classifiers; therefore, weak decision tree classifiers (CART) [50] are used in the experiments.

2. *Finding ensemble weights on validation dataset (SDWEC training)*: SDWEC finds the ensemble weights of base classifiers using y and $\mathbf{H}$. Here, y consists of true labels and $\mathbf{H}$ consists of l classifier predictions for m data rows for the validation dataset. Prediction speed of creating the matrix $\mathbf{H}$ depends on the choice of base classifier, number of data in the validation dataset (m), number of features (d), number of classes (i.e. binary), and data characteristics. So, this study only investigates the computational complexity (see Table 4) and execution time (see Figure 6) of the proposed SDWEC training method (see Algorithm 1) for the ensemble weight finding. Note that, computational complexity of the SDWEC training only depends on number of data in validation set (m), number of classifiers (l), and number of algorithm iteration (k) (see table 3).

3. *Applying ensemble on real-world data (i.e. test dataset)*: Prediction time of SDWEC for test data (or unseen real-world data) depends on base classifiers prediction speed and number of base classifiers selected by SDWEC method (Algorithm 1). As the weights (w) of ensemble becomes more sparse (fewer non-zero elements in the solution w) fewer base classifiers are used in testing phase. Thus, execution time of the testing time decreases as the weights become sparser independent of the employed base classifier. In this study, weak decision tree classifier is used as a base classifier since it is fast in training and prediction; thus, testing time of the SDWEC mostly depends on the sparsity of the obtained ensemble weights.

3.1. Experimental results: sparsity

The principle of parsimony (sparsity) states that simple explanation should be preferred to complicated ones [40]. Sparsity mostly used for feature selection in machine learning. In our study, principle of sparsity is used for selecting subset of classifiers among weak classifiers. During experiments, sparsity definition given in equation 13 is used where $\mathcal{S}(w) = 0$ corresponds to least sparse solution while solution becomes more sparse as $\mathcal{S}(w)$ gets closer to 1. According to dataset and hyper-parameters used, SDWEC achieves different sparsity levels. When SDWEC applied to 11 different datasets, sparsity levels between 0.80 and 0.88 has been achieved (Figure 4). This means that among 200 weak classifiers, 24 classifiers (sparsity level of 0.88) to 40 classifiers (sparsity level of 0.80) are used in ensembles.

$$\mathcal{S}(w) = 1 - \frac{1}{l}||w||_0 \tag{13}$$

where

$$||w||_0 = \#(r|w_r \neq 0), \quad (1 \leqslant r \leqslant l) \tag{14}$$

Here, $||w||_0$ is the L_0-norm of a vector w. Mathematically speaking, L_0-norm is not a proper norm since it is not absolutely homogeneous while it satisfies other norm properties. In practice, L_0-norm is a cardinality function which has its definition in the form of L_p-norm for counting the number of non-zero elements in a given vector.

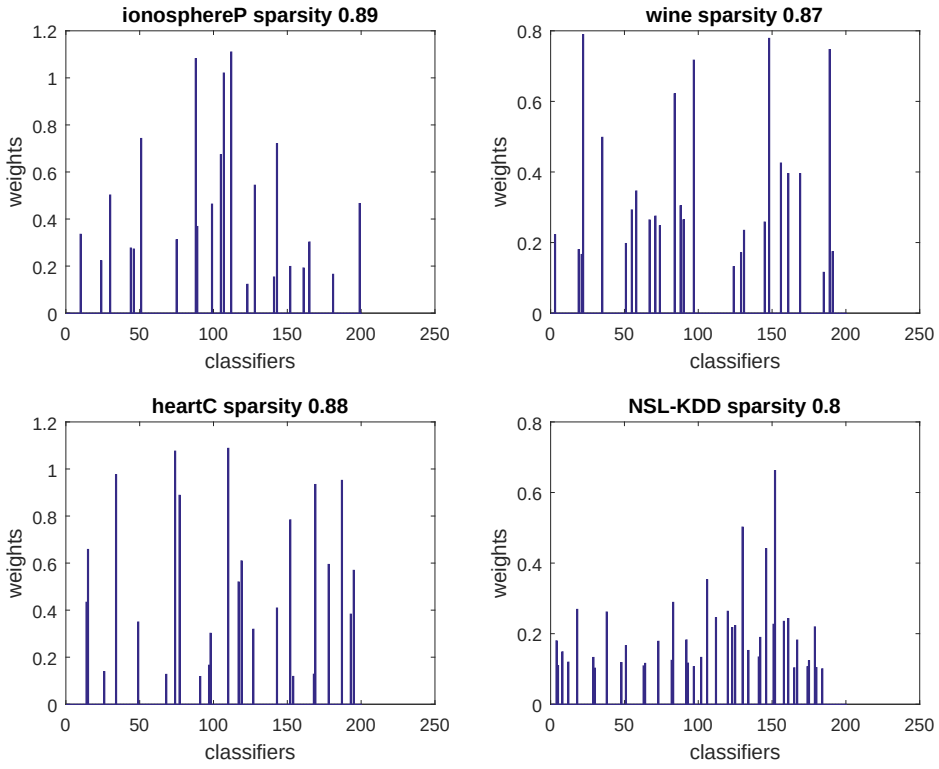

Figure 4: 4 Datasets and their sparsity levels ($\lambda = 1, \beta = 10, \gamma = 20, \varepsilon = 0.1$).

Two different results with different sparsity values (A and B), chosen from Figure 5 have been provided in Table 2. SDWEC-A has no sparsity, all 200 base classifiers have been used in ensemble; thus, it has superior performance at the cost of testing time. SDWEC-A has best accuracy values in 4 out of 10 datasets and it is very close to top performing ones in others. QFWEC is only slightly more accurate in 4 datasets comparing to SDWEC-A while SDWEC-A is only slightly more accurate comparing to QFWEC in other 4 datasets. SDWEC provides similar accuracies with the best performing method (QFWEC) since both QFWEC and SDWEC-A use all base classifiers. SDWEC-B has 0.90 sparsity, 20 of 200 base classifiers have been used in ensemble; nonetheless, it has best accuracy values in 2 out of 10 datasets. Besides, its accuracy values are marginally lower (about 2%) but its testing time is significantly better (90%) than the other approaches. Testing time of the methods in Table 2 is defined as $(1-\mathcal{S}(.))\mathcal{T}(l)$ where $\mathcal{S}(.)$ is the sparsity provided by the ensemble method (see equation 13) and $\mathcal{T}(l)$ is the testing time for all base classifiers. SDWEC-B has 10 times faster testing time comparing to QFWEC since $\mathcal{S}(.)$ is 0 for QFWEC and 0.9 for SDWEC-B.

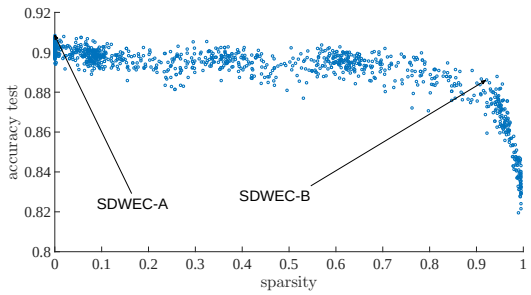

Figure 5: **Sparsity vs accuracy of SDWEC**. The sparsity and accuracy values come from the mean of 11 datasets. Corresponding values can be seen in Table 2.

Table 2: **Comparison of accuracies (sparsity values are given in parentheses)**

Datasets	QFWEC	SDWEC-A	SDWEC-B	WMV	bagging	singleC
breast	0.9736	**0.9737** (0)	0.9532 (0.90)	0.9355	0.9722	0.9400
heartC	0.8085	0.8186 (0)	**0.8279** (0.90)	0.8118	0.8118	0.7268
ionosphere	0.9344	0.9371 (0)	**0.9427** (0.92)	0.9371	0.9342	0.8799
sonarP	0.8088	**0.8136** (0)	0.8126 (0.88)	0.7893	0.8088	0.7367
vehicleP	**0.9788**	0.9693 (0)	0.9539 (0.91)	0.9681	0.9670	0.9634
voteP	0.9576	**0.9703** (0)	0.9525 (0.84)	0.8509	0.9703	0.9533
waveform	**0.8812**	0.8652 (0)	0.8600 (0.93)	0.8634	0.8620	0.8220
wdbcP	**0.9595**	0.9507 (0)	0.9418 (0.88)	0.9489	0.9507	0.9138
wine	**0.9722**	0.9722 (0)	0.9605 (0.89)	0.7514	0.9719	0.9500
wpbcP	0.7989	**0.8036** (0)	0.7477 (0.91)	0.7850	0.7750	0.6911
NSL-KDD	0.9828	0.9766 (0)	0.9849 (0.88)	0.9610	0.9613	**0.9976**
SDWEC-A	$\lambda=0.1$ $\beta=35$ $\gamma=5$ $\varepsilon=0.1$, Mean sparsity 0.00					
SDWEC-B	$\lambda=10$ $\beta=15$ $\gamma=15$ $\varepsilon=1.0$, Mean sparsity 0.90					

3.2. *Computational Complexity Analysis*

In this section, computational complexity of SDWEC (Algorithm 1) has been analyzed. First, computational complexity of every pseudo-code line in Algorithm 1 is given in table 3 and then overall computational complexity is determined. In Table 3, m stands for the number of data in the validation dataset, l stands for the number of base classifiers, and k stands for the iteration count.

Table 3: Computational complexity of SDWEC

Line	Code in Alg 1	Complexity	Notes
6	$\widehat{w} \leftarrow w$	$O(l)$	
7	$\gamma_r \leftarrow \frac{\gamma}{\lvert\widehat{w}\rvert+\varepsilon}$	$O(l)$	
8	construct $\mathbf{S}$ as diagonal form of S_s	$O(ml)$	$\mathbf{S} \leftarrow S_s$ sparse diagonal matrix $(m \times m)$ (Eq 4)
9	v_B	$O(l)$	
9	$\mathbf{C}$	$O(l)$	$\mathbf{C}$ sparse diagonal matrix
10	$\mathbf{SH}$	$O(ml)$	$\underset{m\times l}{\mathbf{X}} = \underset{m\times m}{\mathbf{S}} \times \underset{m\times l}{\mathbf{H}}$
10	$\mathbf{X}^\intercal\mathbf{X}$	$O(l^3)$	$\mathbf{X}^\intercal : O(l^2)$, $\mathbf{X}^\intercal\mathbf{X} : O(l^3)$
10	$\mathbf{M} \leftarrow \frac{2\lambda}{m}[\mathbf{X}^\intercal\mathbf{X}] + \frac{2(\mathbf{C}+\mathbf{I})}{l}$	$O(l^3+l^2)$	
11	$\mathbf{X}^\intercal y$	$O(l^2)$	
11	$\frac{2\widehat{w}-V_B}{l}$	$O(l)$	
11	$b \leftarrow \frac{2\lambda}{m}\mathbf{X}^\intercal y + \frac{2\widehat{w}-V_B}{l}$	$O(l^2+l)$	
12	solve $\mathbf{M}w = b$	$O(l^3)$	Cholesky solver
13	$w = max(w,0)$	$O(l)$	

$\mathbf{M}$ is dense, symmetric, real, and positive definite. Cholesky solver is used to solve $\mathbf{M}w = b$, $O(\frac{2}{3}l^3)$.

Computational complexity inside the for loop is $O(ml)+C_1O(l^3)+C_2O(l^2)+C_3O(l)$. Since $l \ll m$, dominant term is $O(ml)$ for the **SH** multiplication in line 10 of the Algorithm 1, where **S** is a diagonal matrix. Our iteration count is k, then final computational complexity of SDWEC is $O(kml)$, that is linear in k, m, and l (see Table 4 and Figure 6). This computational complexity analysis shows the computational efficiency of the proposed numerical minimization.

Table 4 shows training time (weight finding) of SDWEC on various datasets. Note that, **H** is an input to the Algorithm 1 and calculated as a prior step; thus, training times given in Table 4 only corresponds to SDWEC training. In this experiment, execution time is only dependent on number of rows (m) and number of classifiers (l) since fixed iteration count is used ($k = 25$). In training set, NSL-KDD dataset (100778 instances) has 25 times more instances than waveform dataset (4000 instances). And training time of NSL-KDD (25.95) is about 25 times of waveform (0.96). In Figure 6, SDWEC training times are shown for 3 datasets with different number of data (m), different number of classifiers (l), and for fixed iteration count. As seen in Table 4 and Figure 6, practical execution times are in alignment with theoretical computational complexity analysis. Slight differences between theoretical analysis and actual execution times are due to implementation issues and caching in CPU architectures.

Table 4: SDWEC training time on various datasets,

Dataset	**Rows** (m)	**Time (sec.)** l **classifier count**			
		$l = 100$	$l = 200$	$l = 500$	$l = 1000$
breast-cancer	547	0.05	0.10	0.48	1.63
ionosphereP	280	0.04	0.07	0.31	1.01
wpbcP	155	0.03	0.06	0.26	0.89
wdbcP	456	0.05	0.09	0.44	1.34
wine	143	0.03	0.05	0.23	0.91
waveform	4000	0.43	0.96	3.01	7.78
voteP	186	0.03	0.07	0.24	0.97
vehicleP	667	0.06	0.18	0.73	1.83
sonarP	167	0.03	0.06	0.23	0.83
heartC	239	0.03	0.07	0.25	1.02
NSL-KDD	100778	12.73	25.95	80.23	204.59

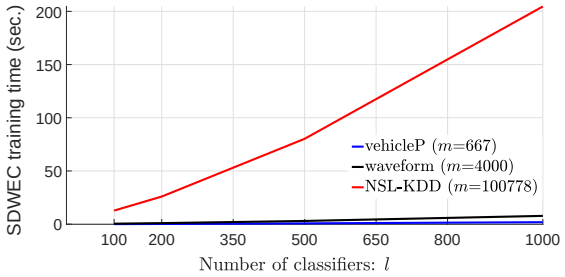

Figure 6: Number of classifier (l) versus SDWEC training time (see Table 4).

4. Conclusion

In this article, a novel sparsity driven ensemble classifier method, SDWEC, has been presented. An efficient and accurate solution for original cost function (hard to minimize, non-convex, and non-differentiable) has been developed. A novel convex relaxation technique for *sign* function, and a novel adaptive *log-sum-exp* approximation for the approximation of *max* function that reduces numerical overflows are proposed. Computational complexity of SDWEC has been investigated theoretically and experimentally. SDWEC training has a linear computational complexity in number of classifier used (l), number of instances in the validation dataset (m), and number of algorithm iterations (k). SDWEC has been compared with other ensemble methods in well-known UCI and NSL-KDD datasets. According to the experiments, SDWEC decreases number of classifiers used in ensemble without significant loss of accuracy. By tuning parameters of SDWEC, a more sparse ensemble –thus, better testing time– can be obtained with a small decrease in accuracy.

Appendix

Optimality conditions can be used to show strict convexity since equation 11 is in matrix-vector form and differentiable.

In equation 12, first derivative of equation 11 is equalized to zero and a close-form solution is obtained as a linear system so first order optimality condition is satisfied.

Let $\mathbf{G}$ be a second derivative (Hessian) of the cost function $J^{(n)}(w)$ given in the equation 11. A symmetric matrix $\mathbf{G} \in \mathbb{R}^{lxl}$ is called positive definite (thus $J^{(n)}(w)$ is strictly convex), denoted by $\mathbf{G} \succ 0$, if $x^{\intercal}\mathbf{G}x > 0$ for every $x \in \mathbb{R}^l$ with $x \neq 0$. Lets take the second derivative of the convex-relaxed cost function $J^{(n)}(w)$ given in equation 11:

$$\frac{\partial^2 J^{(n)}(w)}{\partial w^2} = \frac{2\lambda}{m}(\mathbf{SH})^{\intercal}(\mathbf{SH}) + \frac{2}{l}\mathbf{C} = \mathbf{G} \qquad (15)$$

Lets show that $x^{\intercal}\mathbf{G}x > 0$ for all non-zero x:

$$x^{\intercal}(\frac{2\lambda}{m}(\mathbf{SH})^{\intercal}(\mathbf{SH}) + \frac{2}{l}\mathbf{C})x > 0 \qquad (16)$$

If we distribute $x^{\intercal}$ and x from left and right:

$$\frac{2\lambda}{m}x^{\intercal}(\mathbf{SH})^{\intercal}(\mathbf{SH})x + \frac{2}{l}x^{\intercal}\mathbf{C}x > 0 \qquad (17)$$

Since λ, m, and l are all positive we just need to show that $x^{\intercal}(\mathbf{SH})^{\intercal}(\mathbf{SH})x > 0$ and $x^{\intercal}\mathbf{C}x > 0$:

$$x^{\intercal}(\mathbf{SH})^{\intercal}(\mathbf{SH})x > 0 \rightarrow (\mathbf{SH}x)^{\intercal}(\mathbf{SH}x) > 0 \qquad (18)$$

Lets define z as $z = \mathbf{SH}x$, then $z^{\intercal}z > 0$ since $\mathbf{S}$ is a diagonal matrix with all positive elements (see equation 4), $\mathbf{H}$ contains non-zero elements $\{-1, 1\}$, and x is non-zero vector.

$\mathbf{C}$ is a diagonal matrix with diagonal elements C_r, $1 \leqslant r \leqslant l$, which are defined as below (from second order Taylor approximation):

$$C_r = \frac{\gamma_r(4u_r^2 + 8u_r^3 + 4u_r^4 + \beta u_r + 2\beta u_r^3 + \beta u_r^5)}{2(u_r^3 + u_r^2 + u_r + 1)^2} \qquad (19)$$

where $u_r = e^{\widehat{w}_r\gamma_r}$. Here, β is a positive constant, $\gamma_r = \gamma(|\widehat{w}_r| + \varepsilon)^{-1}$ is always positive since $\gamma > 0$, and u_r is always positive since $\widehat{w}_r\gamma_r \geqslant 0$. Thus, $x^{\intercal}\mathbf{C}x > 0$ is satisfied since C_r is always positive.

Therefore, both first order optimality conditions and second order optimality conditions are satisfied which shows that cost function $J^{(n)}(w)$ given in equation 11 is strictly convex.

References

[1] L. I. Kuncheva, J. C. Bezdek, and R. P. Duin, "Decision templates for multiple classifier fusion: an experimental comparison," *Pattern Recognition*, vol. 34, no. 2, pp. 299 – 314, 2001.

[2] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *European conference on computational learning theory*, pp. 23–37, Springer, 1995.

[3] Y. Zhang, H. Zhang, J. Cai, and B. Yang, "A weighted voting classifier based on differential evolution," *Abstract and Applied Analysis*, vol. 2014, p. 6, 2014.

[4] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine Learning*, vol. 51, no. 2, pp. 181–207, 2003.

[5] B. Krawczyk and M. Woźniak, "Diversity measures for one-class classifier ensembles," *Neurocomputing*, vol. 126, pp. 36 – 44, 2014.

[6] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[7] L. I. Kuncheva, *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2005.

[8] L. I. Kuncheva and J. J. Rodríguez, "A weighted voting framework for classifiers ensembles," *Knowledge and Information Systems*, vol. 38, no. 2, pp. 259–275, 2014.

[9] D. H. Wolpert, *The Supervised Learning No-Free-Lunch Theorems*, pp. 25–42. London: Springer London, 2002.

[10] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *Journal of Japanese Society for Artificial Intelligence*, vol. 14, pp. 771–780, 1999.

[11] P. Gurram and H. Kwon, "Sparse kernel-based ensemble learning with fully optimized kernel parameters for hyperspectral classification problems," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 51, pp. 787–802, Feb 2013.

[12] H. Lee, E. Kim, and W. Pedrycz, "A new selective neural network ensemble with negative correlation," *Applied Intelligence*, vol. 37, no. 4, pp. 488–498, 2012.

[13] J. Tian and N. Feng, "Adaptive generalized ensemble construction with feature selection and its application in recommendation," *International Journal of Computational Intelligence Systems*, vol. 7, no. sup2, pp. 35–43, 2014.

[14] L. Zhang, W.-D. Zhou, and F.-Z. Li, "Kernel sparse representation-based classifier ensemble for face recognition," *Multimedia Tools and Applications*, vol. 74, no. 1, pp. 123–137, 2015.

[15] S. Kim, F. Scalzo, D. Telesca, and X. Hu, "Ensemble of sparse classifiers for high-dimensional biological data," *International Journal of Data Mining and Bioinformatics*, vol. 12, no. 2, pp. 167–183, 2015.

[16] A. Özgür and H. Erdem, "Feature selection and multiple classifier fusion using genetic algorithms in intrusion detection systems," *Journal of the Faculty of Engineering and Architecture of Gazi University*, vol. 33, no. 1, pp. 75–87, 2018.

[17] J. Sylvester and N. V. Chawla, "Evolutionary ensemble creation and thinning," in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pp. 5148–5155, 2006.

[18] N. Li and Z.-H. Zhou, "Selective ensemble under regularization framework," in *Multiple Classifier Systems: 8th International Workshop, MCS 2009*, (Berlin, Heidelberg), pp. 293–303, Springer Berlin Heidelberg, 2009.

[19] H. Kim, H. Kim, H. Moon, and H. Ahn, "A weight-adjusted voting algorithm for ensembles of classifiers," *Journal of the Korean Statistical Society*, vol. 40, no. 4, pp. 437 – 449, 2011.

[20] S. Mao, L. Jiao, L. Xiong, and S. Gou, "Greedy optimization classifiers ensemble based on diversity," *Pattern Recognition*, vol. 44, no. 6, pp. 1245 – 1261, 2011.

[21] L. Zhang and W.-D. Zhou, "Sparse ensembles using weighted combination methods based on linear programming," *Pattern Recognition*, vol. 44, no. 1, pp. 97 – 106, 2011.

[22] N. Goldberg and J. Eckstein, "Sparse weighted voting classifier selection and its linear programming relaxations," *Information Processing Letters*, vol. 112, no. 12, pp. 481 – 486, 2012.

[23] A. B. Santos, A. de A. Araújo, and D. Menotti, "Combiner of classifiers using genetic algorithm for classification of remote sensed hyperspectral images," in *2012 IEEE International Geoscience and Remote Sensing Symposium*, 2012.

[24] X.-C. Yin, K. Huang, H.-W. Hao, K. Iqbal, and Z.-B. Wang, "Classifier ensemble using a heuristic learning with sparsity and diversity," in *Neural Information Processing: 19th International Conference, ICONIP 2012*, (Berlin, Heidelberg), 2012.

[25] Y. Meng and L.-F. Kwok, "Enhancing false alarm reduction using voted ensemble selection in intrusion detection," *International Journal of Computational Intelligence Systems*, vol. 6, no. 4, pp. 626–638, 2013.

[26] S. L. J. L. Tinoco, H. G. Santos, D. Menotti, A. B. Santos, and J. A. dos Santos, "Ensemble of classifiers for remote sensed hyperspectral land cover analysis: An approach based on linear programming and weighted linear combination," in *2013 IEEE International Geoscience and Remote Sensing Symposium - IGARSS*, 2013.

[27] V. Hautamäki, T. Kinnunen, F. Sedlák, K. A. Lee, B. Ma, and H. Li, "Sparse classifier fusion for speaker verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, pp. 1622–1631, Aug 2013.

[28] C. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006.

[29] M. U. Sen and H. Erdogan, "Linear classifier combination and selection using group sparse regularization and hinge loss," *Pattern Recognition Letters*, vol. 34, no. 3, pp. 265 – 274, 2013.

[30] S. Mao, L. Xiong, L. C. Jiao, S. Zhang, and B. Chen, "Weighted ensemble based on 0-1 matrix decomposition," *Electronics Letters*, vol. 49, pp. 116–118, January 2013.

[31] X.-C. Yin, K. Huang, H.-W. Hao, K. Iqbal, and Z.-B. Wang, "A novel classifier ensemble method with sparsity and diversity," *Neurocomputing*, vol. 134, pp. 214 – 221, 2014.

[32] X.-C. Yin, K. Huang, C. Yang, and H.-W. Hao, "Convex ensemble learning with sparsity and diversity," *Information Fusion*, vol. 20, pp. 49 – 59, 2014.

[33] S. Mao, L. Jiao, L. Xiong, S. Gou, B. Chen, and S.-K. Yeung, "Weighted classifier ensemble based on quadratic form," *Pattern Recognition*, vol. 48, no. 5, pp. 1688 – 1706, 2015.

[34] S. Shukla, J. Sharma, S. Khare, S. Kochkar, and V. Dharni, "A novel sparse ensemble pruning algorithm using a new diversity measure," in *2015 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pp. 1–4, Dec 2015.

[35] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009.

[36] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on Signal Processing*, vol. 41, pp. 3397–3415, Dec 1993.

[37] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1." `http://cvxr.com/cvx`, Mar. 2014.

[38] M. Grant and S. Boyd, "Graph implementations for nonsmooth convex programs," in *Recent Advances in Learning and Control* (V. Blondel, S. Boyd, and H. Kimura, eds.), Lecture Notes in Control and Information Sciences, pp. 95–110, Springer-Verlag Limited, 2008. `http://stanford.edu/~boyd/graph_dcp.html`.

[39] F. Nar, A. Özgür, and A. N. Saran, "Sparsity-driven change detection in multitemporal sar images," *IEEE Geoscience and Remote Sensing Letters*, vol. 13, no. 7, 2016.

[40] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski, "Optimization with sparsity-inducing penalties," *Foundations and Trends in Machine Learning*, vol. 4, no. 1, pp. 1–106, 2012.

[41] D. Ge, X. Jiang, and Y. Ye, "A note on the complexity of l p minimization," *Mathematical programming*, vol. 129, no. 2, pp. 285–299, 2011.

[42] J. A. Tropp, "Just relax: convex programming methods for identifying sparse signals in noise," *IEEE Transactions on Information Theory*, vol. 52, pp. 1030–1051, March 2006.

[43] D. Bertsekas, *Nonlinear Programming*. Athena Scientific, 2016.

[44] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[45] "IEEE standard for binary floating-point arithmetic," 1985. Note: Standard 754–1985.

[46] T. Pock, D. Cremers, H. Bischof, and A. Chambolle, "An algorithm for minimizing the mumford-shah functional," in *2009 IEEE 12th International Conference on Computer Vision*, pp. 1133–1140, IEEE, 2009.

[47] A. Özgür and H. Erdem, "A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015," *PeerJ Preprints*, 2016.

[48] M. Albayati and B. Issac, "Analysis of intelligent classifiers and enhancing the detection accuracy for intrusion detection system," *International Journal of Computational Intelligence Systems*, vol. 8, no. 5, pp. 841–853, 2015.

[49] J. Hussain, S. Lalmuanawma, and L. Chhakchhuak, "A two-stage hybrid classification technique for network intrusion detection system," *International Journal of Computational Intelligence Systems*, vol. 9, no. 5, pp. 863–875, 2016.

[50] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.