# Robust Multi-Class Boosting

*Gunnar Rätsch*

Fraunhofer FIRST
Kekuléstr. 7, 12489 Berlin, Germany
Gunnar.Raetsch@first.fhg.de

Max-Planck Institute for Biol. Cybernetics
Spemannstr. 38, 72076 Tübingen, Germany
Gunnar.Raetsch@tuebingen.mpg.de

## Abstract

Boosting approaches are based on the idea that high-quality learning algorithms can be formed by repeated use of a "weak-learner", which is required to perform only slightly better than random guessing. It is known that Boosting can lead to drastic improvements compared to the individual weak-learner. For two-class problems it has been shown that the original Boosting algorithm, called AdaBoost, is quite unaffected by overfitting. However, for the case of noisy data, it is also understood that AdaBoost can be improved considerably by introducing some regularization technique.

In speech-related problems one often considers multi-class problems and Boosting formulations have been used successfully to solve them. I review existing multi-class boosting algorithms, which have been much less analyzed and explored than the two-class pendants. In this work I extend these methods to derive new boosting algorithms which are more robust against outliers and noise in the data and are able to exploit prior knowledge about relationships between the classes.

## 1. Introduction

Boosting is a technique for sequentially training and combining a collection of classifiers in such a way that the later classifiers adjust for the deficiencies of the earlier ones. Many variants exists and a few of them I will briefly review in this work. They all follow the same basic strategy: There is a sequence of iterations, and at each iteration a new classifier is trained on a *weighted training* set. Initially, every example gets the same weight, but in later iterations, the weights of hard-to-classify examples are increased relative to the easy ones.

Experimental results in a variety of classifications tasks have shown AdaBoosts remarkable success in improving the classifier accuracy (cf. [1] and references therein). This is supported by the theoretical analysis in [2] showing that AdaBoost is effective in increasing the *separation margin* between the classes, which plays a crucial role in bounding the generalization error of the resulting combined classifier. Large margin algorithms as Support Vector Machines and Boosting are nowadays the state-of-the-art in machine learning [3].

In speech recognition, however, there are only a few studies applying boosting to acoustic models: in continuous speech recognition [4], in voice-mail transcription [5] and utterance classification [6, 7]. This small number of applications in speech recognition is surprising as it seems particularly simple to use boosting techniques with frequently used hidden Markov models and, hence, to combine the benefits of a well-tuned probabilistic model for speech recognition with the state-of-the-art large margin concept from machine learning.

In some of the mentioned applications remarkable improvements using boosting were observed. In other cases, however, only minor or no improvements could be achieved. I suspect that this is mainly due to used multi-class boosting formulation (AdaBoost.M2). As in the binary case, this formulation implicitly assumes that the data is separable and may overemphasize examples that are hard to classify. For the binary case, robust versions of AdaBoost have been developed: They "give up" on hard-to-learn examples, which led to great improvements in many experiments (cf. [1] and references therein). Those methods still generate large margins, but incorporate the idea of a *soft-margin* which also made the practical success of Support-Vector-Machines [8] possible.

In this work I propose an improvement of current boosting algorithms for the multi-class case by incorporating the soft-margin concept. I start with a brief review in Section 2 of existing boosting methods. In Section 3, I generalize the margin definition, then outline the robustified algorithm and discuss several extensions and modifications which seem appropriate for use in a speech recognition system. Finally I give a few details of the derivation of the algorithm and conclude with a brief discussion in Section 4.

## 2. Multi-Class Boosting

The purpose of boosting is to find a highly accurate classification rule by combining many *weak* or *base hypotheses*, each of which may only be moderately accurate. It is assumed to have access to a separate procedure called the *weak learner* or *weak learning algorithm* for computing the weak hypotheses. The boosting algorithm calls the weak learner repeatedly in a series of rounds and so finds a set of weak hypotheses. Those are then linearly combined into a single rule – the *final* or *combined hypothesis*.

In the simplest version of AdaBoost for multi-class classification, called AdaBoost.M1 [9], the algorithm maintains a set of importance weights over the training examples. These weights are used by the weak learning algorithm whose goal is to find a weak hypothesis with moderately low weighted error. AdaBoost.M1 uses the weights to force the weak learning algorithm to concentrate on the examples which are hardest to classify [10].

Later it was found that it is more appropriate to maintain a distribution on the examples *and the labels*. As boosting progresses *training examples and their corresponding labels* that are hard to predict correctly, get incrementally higher weights than the other examples and labels. The intended effect is to force the weak learner to concentrate on examples and labels that will be most beneficial to the overall goal of finding a highly accurate classification rule [10].

### 2.1. AdaBoost.M2

The latter idea is implemented in AdaBoost.M2 (cf. Algorithm 1) as proposed in [9]. One starts with the uniform distribution on the examples and then calls the weak learner, which returns a hypothesis $\boldsymbol{h}_t : \mathbf{x} \to [0,1]^C$, where $C$ is the number of classes, i.e. for every example it assigns a score in the range $[0,1]$ for each class $c = 1, \ldots, C$. I denote by $\boldsymbol{h}^c(\mathbf{x})$ the $c$-th component of $\boldsymbol{h}(\mathbf{x})$. The quality of the new hypothesis is given by the pseudo-loss $\epsilon_t \in [0,1]$, which is equal to a half minus the weighted average difference of the correct class-score and any

**Algorithm 1** The AdaBoost.M2 algorithm

1. **Input:** $N$ examples $Z = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, Number of iterations $T$
2. **Initialize:** $d_{n,c}^{(1)} = 1/(C-1)$
3. **Do for** $t = 1, \ldots, T$,
   (a) Train classifier with respect to the weighted examples and obtain hypothesis $\boldsymbol{h}_t : \mathbf{x} \mapsto [0,1]^C$ with small pseudo-loss
   $$\epsilon_t = \frac{1}{2} \sum_{n, c \neq y_n} d_{n,c}(1 - \boldsymbol{h}_t^{y_n}(\mathbf{x}_n) + \boldsymbol{h}_t^c(\mathbf{x}_n))$$
   (b) Let $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon}$
   (c) Update
   $$d_{n,c}^{(t+1)} = \frac{d_{n,c}^{(t)}}{Z_t} \exp\left(-\alpha_t (1 + \boldsymbol{h}_t^{y_n}(\mathbf{x}_n) - \boldsymbol{h}_t^c(\mathbf{x}_n))\right)$$
   where $Z_t$ is such that $\sum_{n, c \neq y_n} d_{n,c}^{(t+1)} = 1$.
4. **Output:** Final hypothesis with weights $\boldsymbol{\alpha}^{(T)}$:
   $$\boldsymbol{f}(\mathbf{x}) = \sum_{t=1}^{T} \frac{\alpha_t}{\sum_r \alpha_r} \boldsymbol{h}_t(\mathbf{x})$$

other class-score (cf. step 3a in Algorithm 1). Hence, if in most cases $\boldsymbol{h}_t^{y_n}(\mathbf{x}_n) - \boldsymbol{h}_t^c > 0 \, (c \neq y_n)$ holds, then $\epsilon_t$ will be considerable smaller than $\frac{1}{2}$. It can be shown that if the base learner always returns a hypothesis with pseudo-loss consistently smaller than $\frac{1}{2}$, then the training error of the combined hypothesis will converge fast to zero. For the final hypothesis $\boldsymbol{f} : \mathbf{x} \to \mathbb{R}^C$ holds in this case:

$$\boldsymbol{f}^{y_n}(\mathbf{x}_n) - \max_{c \neq y_n} \boldsymbol{f}^c(\mathbf{x}_n) > 0, \tag{1}$$

for all $n = 1, \ldots, N$, i.e. the data is separated. In the following I will call the left-hand-side of (1) the margin of the example $\mathbf{x}_n$. In binary classification, the margin corresponds to the *separation distance* between the linearly separated classes. From a theoretical point of view it is beneficial, as for large-margin two-class classification, to maximize the separation between the classes if the data is noise-free and separable. AdaBoost.M2 tries to achieve this, but as for binary AdaBoost it could not be shown that it maximizes the separation distance.[1] However, if the data is noisy – as often in speech recognition – then the *pure* large margin strategy turns out to be sub-optimal. In this case AdaBoost overemphasizes some examples in order to classify them right – the algorithm becomes very sensible to such examples and tends to overfit. Here, the soft-margin idea (as proposed for binary classification in [12]) will help making the algorithm more robust against noise.

### 2.2. Logistic Regression

A different algorithm was proposed in [13]. For simplicity let us use the following notation: let $\mathbf{y}_n$ be the $C$-dimensional vector of the form $(0, 0, \ldots, 0, 1, 0, \ldots, 0)$ with a 1 in the $c$-th position if, and only if, $\mathbf{x}_n$ belongs to the $c$-th class. In [13] the log-likelihood function:

$$G = \sum_{n=1}^{N} \sum_{c=1}^{C} \{y_{n,c} \log p(c|\mathbf{x}_n) + (1 - y_{n,c}) \log(1 - p(c|\mathbf{x}_n))\}$$

was considered, where $y_{n,c}$ is the $c$-th component of the vector $\mathbf{y}_n$, and $p(c|\mathbf{x}_n)$ is the model probability that $\mathbf{x}_n$ belongs to the $c$-th class. The standard soft-max representation, $p(c|\mathbf{x}) = \frac{\exp(f^c(\mathbf{x}))}{\sum_{c'=1}^{C} \exp(f^{c'}(\mathbf{x}))}$, where $f^c(\mathbf{x})$ is the $c$-th component of the combined hypothesis, was used to obtain a probability vector from the combined hypothesis $\boldsymbol{f}(\mathbf{x})$. Then using an algorithm similar to Algorithm 1, but using a different weighting scheme and some additional optimization methods, one can iteratively find a combined hypothesis that maximizes the log-likelihood. The clear probabilistic modeling seems nice, but the important concept of the large margin classification is missing in this approach. However, since examples are not overemphasized too much, Logistic Regression worked in noisy cases considerable better than AdaBoost.M2 [13]. But since an appropriate way of complexity control is missing, I expect it to overfit easily as one continues to combine weak hypotheses. The proposed algorithm in Section 3 uses similar weight updates and also exhibits the mentioned robustness against noise, while having the large margin concept incorporated.

### 2.3. Error Correcting Codes

There has recently been extensive activity on recoding multi-class problems as a sequence of binary problems, using the idea of error-correcting codes [14]. A detailed description of the approach can be found in [15] in a general context as well as in the Boosting setup. Two special cases of this general approach are the so-called *one-against-all* and *all-pairs* approaches, which were extensively used prior to the development of the error correcting code approach. In the former case, given a $C$-class classification problem, one constructs $C$ binary classifiers each of which learns to distinguish one class from the rest. The multi-category classifier then uses the highest ranking classifier. In the all-pairs approach, all possible $\binom{k}{2}$ pairs of classification problems are learned and used to form the multi-category classifier using some form of majority voting. Although these techniques seem very promising, to my knowledge those techniques have not yet been applied to speech recognition (at least the all-pairs approach seems infeasible when having many classes). In the next section I will discuss an approach which is based on real-valued codes but their use is differently motivated than in e.g. [14].

## 3. A Robust Multi-Class Algorithm

### 3.1. Margins and Codes

Before I present the new algorithm, I generalize the definition of the margin as used before in (1). Let us consider the problem of $C$ classes and $N$ training examples $(\mathbf{x}_n, y_n) \in \mathcal{X} \times \{1, \ldots, C\}$. Suppose one is given a set of *code vectors* $\boldsymbol{t}_c$, $c = 1, \ldots, C$. For simplicity assume $\boldsymbol{t}_c \in \mathbb{R}^C =: \mathcal{C}$ and $\|\boldsymbol{t}_c\|_2 = 1$. Then one can reformulate the classification problem as follows: Try to find a function $\boldsymbol{f} : \mathcal{X} \to \mathcal{C}$ that maps examples of the $c$-th class close to its code vector $\boldsymbol{t}_c$ and far from all the other vectors. Then it is natural to define the margin $\rho$ of $\boldsymbol{f}$ at example $(\mathbf{x}_n, y_n)$ as follows:

$$\rho(\boldsymbol{f}(\mathbf{x}), y) := \min_{c \neq y} \mathrm{d}(\boldsymbol{t}_c, \boldsymbol{f}(\mathbf{x})) - \mathrm{d}(\boldsymbol{t}_y, \boldsymbol{f}(\mathbf{x})), \tag{2}$$

where $\mathbf{d}$ is some distance measure between code and mapped example. It has been shown in [16] that in order to obtain a convex optimization problem (and thus ensure efficiency in optimization), the distance has to have the following form $\mathrm{d}(\mathbf{a}, \mathbf{b}) = d_1(\mathbf{a}) + d_2(\mathbf{b}) + \mathbf{a}^\top B \mathbf{b}$, where $\mathrm{d}_1$ and $\mathrm{d}_2$ are convex functions and the matrix $B$ is symmetric. The simplest case is the squared Euclidean distance, which I will assume here, but other distance measures might be more appropriate. For the Euclidean distance, (2) simplifies to

$$\rho(\boldsymbol{f}(\mathbf{x}), y) := \min_{c \neq y} \, (\boldsymbol{t}_y - \boldsymbol{t}_c)^\top \boldsymbol{f}(\mathbf{x}), \tag{3}$$

where I used the assumption $\|\boldsymbol{t}_y\| = \|\boldsymbol{t}_c\| = 1$.

Let us consider the case, where the code vectors are simply the unit vectors, i.e. $\boldsymbol{t}_c = \mathbf{e}_c$. Then (3) simplifies to the original

[1] However, along the lines of [11] one may also be able to show that AdaBoost.M2 generates *large* margins.

margin definition in (1). The following derivations and the algorithm are formulated using this more general notion of margin, but it can easily be translated back to the original setting.

But why is it useful to have these code vectors? I can only give an intuition. Suppose one has many different classes (as in speech recognition), then some of these classes will be very similar (e.g. 'here' and 'beer'), while others have nothing in common ('sausage' and 'bread'). Moreover, assume one knows these relationships *a priori* and has a set of code vectors available that reflect these class relations. Then it would be relatively easy to learn a function that maps examples of similar classes into the right area of the code space – even with a very coarse model. What is left is to learn the fine level class separation which I believe is easier if the code vectors appropriately reflect the class relationships.

In [16] we proposed an algorithm that automatically learns such relationships, but the large number of classes in speech recognition would make a direct application of this approach infeasible. However, existing prior knowledge could be used to find a good set of code vectors and if this knowledge is not available, then the default unit code vectors have to suffice.

**Algorithm 2** The AdaBoost.M$\nu$ algorithm

1. **Input:** $N$ examples $Z = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, Number of iterations $T$, Regularization constant $\nu$ (e.g. $\nu = 0.2$), approximation constant $\beta > 0$ (e.g. $\beta = 1/100$), a set of $C$ code vectors $\boldsymbol{t}_c \in \mathbb{R}^C$ (e.g. $\boldsymbol{t}_c = \mathbf{e}_c$)
2. **Initialize:** $d_{n,c}^{(1)} = 1$, $\boldsymbol{\alpha}^{(0)} = \mathbf{e}^{(1)}$ (= 1st unit vector)
3. **Do for** $t = 1, \ldots, T$,
   (a) Train classifier with respect to the weighted examples and obtain hypothesis $\boldsymbol{h}_t : \mathbf{x} \mapsto \mathbb{R}^C$ with small pseudo-loss as in (5)
   (b) Find best trade-off between new and old hypotheses by minimizing $E(\varrho, \boldsymbol{\alpha})$ as defined in (4)
   $[\varrho^{(t)}, \lambda^{(t)}] = \operatorname*{argmin}_{\varrho \geq 0, \lambda \in [0,1]} E(\varrho, \lambda \mathbf{e}^{(t)} + (1-\lambda)\boldsymbol{\alpha}^{(t-1)})$
   (c) Let $\boldsymbol{\alpha}^{(t)} = \lambda^{(t)} \mathbf{e}^{(t)} + (1-\lambda)\boldsymbol{\alpha}^{(t-1)}$
   (d) Let $\rho_{n,c} = (\boldsymbol{t}_{y_n} - \boldsymbol{t}_c)^\top \boldsymbol{f}_{\boldsymbol{\alpha}^{(t)}}(\mathbf{x}_n)$
   (e) Update $d_{n,c} = \dfrac{\exp\left(\frac{\varrho - \rho_{n,c}}{\beta}\right)}{1 + \exp\left(\frac{\varrho - \rho_{n,c}}{\beta}\right)}$
4. **Output:** Final hypothesis with weights $\boldsymbol{\alpha}^{(T)}$:
   $\boldsymbol{f}_{\boldsymbol{\alpha}^{(T)}}(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t^{(T)} \boldsymbol{h}_t(\mathbf{x})$

### 3.2. The Algorithm

The proposed algorithm is outlined in Algorithm 2. It works along the same lines as AdaBoost.M2. The main differences are a) the computation of the hypothesis coefficient $\alpha_t$ and b) a different weighting scheme for the examples. Moreover, it has two additional parameters $\nu$ and $\beta$, which I will discuss below.

a) In AdaBoost.M2, one computes the hypothesis coefficient $\alpha_t$ at iteration $t$ and normalizes the vector at the end. This is equivalent to keeping the length of $\boldsymbol{\alpha}$ equal to 1 and by letting the new weight vector $\boldsymbol{\alpha}^{(t)}$ be a *convex combination* of the old weight vector $\boldsymbol{\alpha}^{(t-1)}$ and the $t$-th unit vector $\mathbf{e}_t$ as in step 3c. The weighting is determined by $\lambda_t$, which is determined in step 3b by minimizing the following loss function:

$$E(\varrho, \boldsymbol{\alpha}) = \beta \sum_{n, c \neq y_n} \log\left[1 + \mathrm{e}^{\frac{\varrho - \rho_{n,c}}{\beta}}\right] - N(C-1)\nu\varrho, \qquad (4)$$

where $\rho_{n,c}$ is as in step 3d. In AdaBoost.M2 there is an explicit formula to compute $\alpha_t$, which only approximates the optimal value when $\boldsymbol{h}$ is real valued [17]. Since this is often the case in speech recognition and every call to the weak learner can be very costly, it seems mandatory to accurately determine the optimal $\alpha_t$ in each iteration to speed up the convergence.[2]

b) The weighting in step 3e works different as in AdaBoost.M2. For simplicity, I use the explicit form instead of the iteratively updated distribution. Now one can readily see the effect on the weights: if the margin $\rho_{n,c}$ is much larger than $\varrho$, then the weight will be close to zero, but if it is much smaller, then it will be close to 1. In the extreme case of $\beta \to 0$ holds: $d_{n,c} = 1$, if $\rho_{n,c} < \varrho$ and $d_{n,c} = 0$, if $\rho_{n,c} > \varrho$. Here one can see that the variable $\varrho$ directly controls how many of the weights are non-zero. I will discuss in the next section that if one minimizes the function $E$ with respect to $\varrho$ (when $\beta \to 0$), then it is chosen such that the fraction of $d$'s which are non-zero is (almost) equal to $\nu$ (the hyper-parameter). Thus, $\nu \in [0, 1]$ allows one to control easily the fraction of margin errors and provides an interpretable way to control the complexity of the ensemble. The interpretability makes model selection often much easier ($\nu = 0.2$ works very well in many cases).

The parameter $\beta$ controls how well this so-called $\nu$-property is satisfied. For too large $\beta$ the approximation will only be fair. As one decreases $\beta$, then the approximation quality improves, but the function $E$ becomes less and less smooth, the optimization harder and the convergence to the optimum slower.

Finally, note that the algorithm only progresses if the base learner is able to return a weak hypothesis with small pseudo-loss. In the most general case, the pseudo-loss uses the code vectors

$$\gamma_t = \sum_{n, c \neq y_n} d_{n,c} (\boldsymbol{t}_c - \boldsymbol{t}_{y_n})^\top \boldsymbol{h}_t(\mathbf{x}_n). \qquad (5)$$

If one uses the trivial code, this definition is up to constants equal to the one used in step 3a in Algorithm 1. One can show, that if $\gamma_t > 0$ then the optimal $\lambda_t$ is equal zero and the algorithm stops making progress.

### 3.3. Remarks

Let me briefly discuss two questions: a) How should one use the weights $d_{n,c}$ in the base learner? b) How can one reduce the large number of weights when training with many classes?

a) In [9] it was proposed to provide the weak learner only with a "marginal weighting" on the examples $D_n = \frac{1}{Z} \sum_{c \neq y_n} d_{n,c}$, where $Z$ is such that $\sum_n D_n = 1$. This strategy seems particularly useful, when the classes are trained separately using ML only using (positive) the examples of the corresponding class. However, this ignores the case when an example is well separated to one class $c_1 \neq y_n$ but not from another class $c_2 \neq y_n$. In order to take this into account, discriminative training seems more appropriate which makes it possible to force the learning algorithm to discriminate better between $c_2$ and $y_n$ for this example. A first step in this direction was done in [6], which led to considerable improvements.

b) As noted before by [5, 6, 18], the computation of the weights $d_{n,c}$ makes it necessary to compute the output of $\boldsymbol{h}$ for each example for every class which can be computationally too expensive. The proposed method is to use a $M$-best list $L_n$ for each example $\mathbf{x}_n$ and then separate only against the classes in this list. This reduces the number of weights from $(C-1)N$ to $MN$ ($M \ll C$). As discussed before, the fraction of non-zero weights $d_{n,c}$ is determined by $\nu$. In order to take the proposed restriction into account one either has to change the constant in (4) from $N(C-1)$

[2] For AdaBoost.M2 there is also such cost function [17], which can be minimized instead of using the explicit solution.

to $NM$ leading to the appropriate meaning of the parameter $\nu$, or one has to make sure that $M > \nu(C-1)$ such that the algorithm is able to satisfy the $\nu$-property.

### 3.4. Sketch of the Derivation of the Algorithm

I started with the following optimization problem, which similarly appeared in [16]:

$$\min_{\boldsymbol{\alpha}\in\mathcal{S}_T,\boldsymbol{\xi}\geq\mathbf{0},\varrho\geq 0} \sum_{n=1}^{N}\sum_{c\neq y_n} \xi_{n,c} - \nu N(C-1)\varrho \quad (6)$$
$$\text{where } (\boldsymbol{t}_{y_n}-\boldsymbol{t}_c)^\top \boldsymbol{f}_{\boldsymbol{\alpha}}(\mathbf{x}_n) \geq \varrho - \xi_{n,c},$$

where one would like to find a convex combination[3] $\boldsymbol{f}_{\boldsymbol{\alpha}}(\mathbf{x}) = \sum_t \alpha_t \boldsymbol{h}_t(\mathbf{x}_n)$ of base hypotheses $\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots$ that has as large as possible margin $(\boldsymbol{t}_{y_n}-\boldsymbol{t}_c)^\top \boldsymbol{f}_{\boldsymbol{\alpha}}(\mathbf{x}_n)$. I use slack variables $\xi_{n,c}$ to allow for smaller margins than $\varrho$ and misclassifications. In (6) one minimizes the sum of all slack variables (the margin errors) and at the same time maximizes the margin $\varrho$.

The solution of (6) has the following property (cf. [19, 11]): *Suppose the solution of* (6) *satisfies* $\varrho > 0$*, then*

1. *$\nu$ upper-bounds the fraction of margin errors ($\xi_{n,c} > 0$).*
2. *$1-\nu$ is an upper bound on the fraction of examples with a margin $\min_{c\neq y_n}(\boldsymbol{t}_{y_n}-\boldsymbol{t}_c)^\top \boldsymbol{f}_{\boldsymbol{\alpha}}(\mathbf{x}_n)$ larger than $\varrho$.*

Hence, the parameter $\nu$ can be interpreted as the fraction of examples (and labels) in the training set, where the separation to one of the other classes is smaller than $\varrho$. For noisy data one would set this parameter larger and for well separable data close to zero (e.g. $\nu < 1/N(C-1)$).

It can easily be verified that the solution of (6) satisfies $\xi_{n,c} = \max(0, \varrho - (\boldsymbol{t}_{y_n}-\boldsymbol{t}_c)^\top \boldsymbol{f}(\mathbf{x}_n))$. This expression can be plugged into (6) and one obtains a non-smooth function in $\varrho$ and $\boldsymbol{\alpha}$. It is beneficial from an optimization point of view to replace $\max(0, \varrho - a)$ with $\beta \log\left[1+\exp\left(\frac{\varrho-a}{\beta}\right)\right]$, where $\beta$ is a parameter which controls the smoothness and the quality of the approximation. It is easily verified that the approximation is exact for $\beta \to 0$. Finally I have the following objective to minimize

$$\beta\sum_{n,c\neq y_n}\log\left[1+\exp\left(\frac{\varrho-(\boldsymbol{t}_{y_n}-\boldsymbol{t}_c)^\top \boldsymbol{f}_{\boldsymbol{\alpha}}(\mathbf{x}_n)}{\beta}\right)\right] - \nu N(C-1)\varrho,$$

subject to $\boldsymbol{\alpha} \in \mathcal{S}_T$ and $\varrho \geq 0$. This objective is to be minimized over the set of all convex combinations of functions which can be generated by the base learner. I conjecture that by using the results in [20], one can show the convergence of Algorithm 2 to the minimum of (4), which approximately solves (6). Note that one could extend Algorithm 2 to appropriately decrease $\beta$ to zero (as e.g. in [11]). Using the standard theory of barrier optimization one can then show the convergence to the solution of the linear program (6) (cf. [11, 21] for more details).[4]

## 4. Conclusion

In this work I proposed a new boosting algorithm for multi-class problems. It works similar as the original AdaBoost.M2 algorithm, but incorporates the soft-margin concept which improved the classification performance in many cases in binary problems. It combines the robustness against noise and outliers in the data as previously seen in Logistic Regression with large margins. Moreover, I discussed how to take prior knowledge about class relationships into account, which seem very important when learning about problems with many classes. It is future work to test and apply the proposed technique in a speech recognizer.

[3] $\mathcal{S}_T := \{\boldsymbol{\alpha} \in \mathbb{R}^T \mid \alpha_t \geq 0, \sum_t \alpha_t = 1\}$.

[4] Following the lines in [11], one can also show that AdaBoost.M2 essentially corresponds to the optimization problem (6) but with the trivial code vectors and without the slack variables $\xi_{n,c}$.

## 5. References


[1] R. Meir and G. Rätsch, "An introduction to boosting and leveraging," in *Adv. Lect. on Machine Learning*. Springer, 2003, pp. 119–184.

[2] R. Schapire, Y. Freund, P. Bartlett, and W. Lee, "Boosting the margin," *Annals of Statistics*, vol. 26, no. 5, pp. 1651–1686, 1998.

[3] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf, "An introduction to kernel-based learning algorithms," *IEEE Trans. Neural Netw.*, vol. 12, no. 2, pp. 181–201, 2001.

[4] H. Schwenk, "Using boosting to improve a hybrid hmm/neural network speech recognizer," in *Proc. ICASSP-99*, 1999, pp. 1009–1012.

[5] G. Zweig and M. Padmanabhan, "Boosting gaussian mixtures in a lvcsr system," in *Proc. ICASSP-00*, Istambul, 2000, pp. 1527–1530.

[6] C. Meyer and P. Beyerlein, "Towards large margin speech recognizers by boosting and discr. training," in *Proc. ICML*. M. Kaufmann, 2002, pp. 419–426.

[7] G. Cook and A. Robinson, "Boosting the performance of connectionist large vocabulary speech recognition," in *Proc. ICSLP-96*, 1996, pp. 1305–1308.

[8] C. Cortes and V. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[9] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. of Comp. & Sys. Sc.*, vol. 55, no. 1, pp. 119–139, 1997.

[10] R. Schapire and Y. Singer, "BoosTexter: A boosting-based system for text categorization," *Machine Learning*, vol. 39, no. 2/3, pp. 135–168, 2000.

[11] G. Rätsch, "Robust boosting via convex optimization," Ph.D. dissertation, University of Potsdam, 2001.

[12] G. Rätsch, T. Onoda, and K.-R. Müller, "Soft margins for AdaBoost," *Machine Learning*, vol. 42, pp. 287–320, 2001.

[13] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *Annals of Statistics*, vol. 2, pp. 337–374, 2000.

[14] T. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," *Journal of Aritifical Intelligence Research*, vol. 2, pp. 263–286, 1995.

[15] E. Allwein, R. Schapire, and Y. Singer, "Reducing multiclass to binary," *Journal of Machine Learning Research*, vol. 1, pp. 113–141, 2000.

[16] G. Rätsch, A. Smola, and S. Mika, "Adapting codes and embeddings for polychotomies," in *Proc. NIPS*, 2003.

[17] R. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, no. 3, pp. 297–336, 1999.

[18] C. Meyer, "Utterance-level boosting of HMM speech recognizers," in *Proc. ICASSP-02*, 2002.

[19] B. Schölkopf, A. Smola, R. Williamson, and P. Bartlett, "New support vector algorithms," *Neural Computation*, vol. 12, pp. 1207 – 1245, 2000.

[20] T. Zhang, "A general greedy approximation algorithm with applications," in *Advances in Neural Information Processing Systems*, vol. 14. MIT Press, 2002.

[21] G. Rätsch, S. Mika, B. Schölkopf, and K.-R. Müller, "Constructing boosting algorithms from SVMs," *IEEE PAMI*, vol. 24, no. 9, pp. 1184–1199, 2002.