Soft Computing
https://doi.org/10.1007/s00500-018-3577-z

METHODOLOGIES AND APPLICATION

Calibrating AdaBoost for phoneme classification

Gábor Gosztolya[1] · **Róbert Busa-Fekete**[2]

Abstract
Phoneme classification is a classification sub-task of automatic speech recognition (ASR), which is essential in order to achieve good speech recognition accuracy. However, unlike most classification tasks, besides finding the correct class, providing good posterior scores is also an important requirement of it. Partly because of this, formerly Gaussian Mixture Models, while recently Artificial Neural Networks (ANNs) are used in this task, while other common machine learning methods like Support Vector Machines and AdaBoost.MH are applied only rarely. In a previous study, we showed that AdaBoost.MH can match the performance of ANNs in terms of classification accuracy, but lags behind it when utilizing its output in the speech recognition process. This is in part due to the imprecise posterior scores that AdaBoost.MH produces, which is a well-known weakness of this method. To improve the quality of posterior scores produced, it is common to perform some kind of posterior calibration. In this study, we test several posterior calibration techniques in order to improve the overall performance of AdaBoost.MH. We found that posterior calibration is a good way to improve ASR accuracy, especially when we integrate the speech recognition process into the calibration workflow.

Keywords Speech recognition · Phoneme classification · Phoneme probability estimation · Posterior calibration · AdaBoost.MH

1 Introduction

In the automatic speech recognition (ASR) task, we are given a sound recording of a speech of an user (an *utterance*), and our aim is to produce its written transcription. Although it is undoubtedly a machine learning task, over the decades researchers have developed dedicated tools for it, most notably the Hidden Markov Model (HMM) (Morgan and Bourland 1995). This is mainly because speech recognition is special in the sense that it has to handle an input of a varying size, and the output is not of fixed length either. To overcome this, the input is usually divided into small, equal-sized portions, but the sequence of these samples still has to be processed jointly. Hence, general purpose machine learning algorithms are rarely used for speech recognition, and when they are, they are usually applied just for phoneme classification. In this task, each small portion of the speech signal (*frame*) has to be identified as one of the possible phonemes in the given language.

A second reason is that, compared to usual learning tasks, speech recognition databases have rather specific machine learning characteristics. Firstly, there are a fair number of classes (e.g. phonemes distinguished in the given language). Secondly, there are usually very few features: the very popular MFCC + Δ + $\Delta\Delta$ feature set (containing the mel-frequency spectral coefficients, their first- and second-order derivatives) consists of only 39 features for a frame (Rabiner and Juang 1993). This set can be expanded with some other attributes; still, the size of the feature set rarely exceeds a few hundred. But most importantly, a typical speech recognition database contains at least several hours of recordings (nowadays hundreds of hours is not uncommon), which leads to literally millions of training examples. This set-up is quite rare in machine learning, and most classification algorithms cannot easily handle such a huge amount of data. It is more typical to have significantly fewer training examples that are described by a large feature vector, and which belong to one

Communicated by V. Loia.

✉ Gábor Gosztolya
ggabor@inf.u-szeged.hu

Róbert Busa-Fekete
busafekete@oath.com

[1] MTA-SZTE Research Group on Artificial Intelligence, Hungarian Academy of Sciences, Tisza Lajos krt. 103, Szeged, Hungary

[2] Yahoo Research, New York, NY, USA

Published online: 25 October 2018

of the few classes, as in Bodnár and Nyúl (2015) and Kaya et al. (2015). Such tasks arise in speech technology as well; e.g. in speaker recognition (van Leeuwen et al. 2006), emotion detection (Gosztolya et al. 2013; Tóth et al. 2012) and laughter detection (Gosztolya 2015; Gosztolya et al. 2016, 2013; Gupta et al. 2013; Neuberger and Beke 2013).

Furthermore, we perform speech recognition to get the transcript of the whole utterance, so the point of phoneme classification is to lead to high-precision transcriptions. To this end, the frame-level results of phoneme classification are combined, which is usually performed via a HMM. For this step, for each frame, their posterior probability estimates are used instead of the resulting classes (e.g. phonemes). This means that phoneme classification has another requirement: besides achieving a good frame-level classification accuracy, it also has to provide precise class conditional probability values. [This is the reason why this task is also called *phoneme posterior estimation* (Imseng et al. 2011).]

For the above reasons, usually Gaussian Mixture Models (GMM) (Duda and Hart 1973) and Artificial Neural Networks (ANN) (Bishop 1995; Morgan and Bourland 1995) are applied to estimate the posteriors in phoneme classification. Support Vector Machines [SVMs (Schölkopf et al. 2001)] and variants of AdaBoost (e.g. Friedman et al. 2000; Schapire and Freund 2012; Schapire and Singer 1999) have typically become more popular in those tasks where the classification accuracy is the main target performance metric. Although these methods tend to provide only inaccurate posterior estimates, they achieved state-of-the-art classification performance in several such tasks, clearly because perfect classification in multi-class classification does not require the precise estimate of posteriors for *every* class, but only the maximum of the posterior estimates should be accurate.

In a previous study (Gosztolya 2014), we applied both SVM and AdaBoost.MH in the phoneme classification task, and we found that they can compete with ANN in terms of frame-level classification accuracy. However, when we combined the frame-level outputs in a HMM and evaluated the utterance-level results, they slightly lagged behind neural networks. This is also in accordance with the general experience that the posterior scores of AdaBoost.MH are quite imprecise. And although sophisticated methods are available to calibrate the raw posterior outputs of AdaBoost.MH into normalized class conditional probability values, we only applied a very simple posterior processing method.

The goal of calibration is to turn the output scores of a classifier into valid posterior class probability estimates. The output scores of a classifier might mainly be uncalibrated if the learning method seeks to minimize quality measures like the zero-one error, which do not demand precise posterior estimates. Consider the classification rule of, for example, AdaBoost.MH which is a multi-class boosting implementation, or multi-class SVM (Crammer and Singer 2001), which consists of predicting the label which has the highest output score. Thus, achieving low multi-class zero-one error does not require good posterior estimates. This kind of inconsistency of output scores has *methodological* roots and various calibration techniques had been devised to handle it (Niculescu-Mizil and Caruana 2005; Platt 2000).

Calibration techniques play an important role in many machine learning applications. For example in classifier combination, the outputs of various classifiers need to be comparable on an absolute scale (Busa-Fekete et al. 2013) or in all learning tasks where the evaluation metric applied requires accurate modelling of posterior probabilities like instance-wise F-measure (Waegeman et al. 2014) or cross-entropy performance. Methodologies that are based on some calibration technique can achieve state-of-the-art results in many cases (Busa-Fekete et al. 2013; Drish 2001; Niculescu-Mizil and Caruana 2005; Zadrozny and Elkan 2001). Motivated by this fact, we decided to apply some popular calibration techniques to AdaBoost.MH in phoneme classification.

The structure of this paper is as follows. First, we introduce the speech recognition problem and explain the importance of the phoneme classification task in its workflow (see Fig. 1). Then, we briefly describe the AdaBoost.MH algorithm and the variant that we applied (AdaBoost.MH.BA). Next, we proceed to posterior calibration, where we describe the standard methods tested. After, we turn to the testing part, where we describe the experimental set-up, then present and analyse the results. Lastly, we draw our conclusions and make some suggestions for future study.

2 The speech recognition task

In the speech recognition problem, we have a speech signal represented by a series of observations (frames) $A = a_1 \dots a_t$ and a set of possible phoneme sequences (words or word sequences) denoted by W. Our task is to find the most probable word or word sequence $\hat{w} \in W$ for the given speech signal, i.e.

$$\hat{w} = \arg\max_{w \in W} P(w|A). \tag{1}$$

This, using Bayes' theorem, is equivalent to the maximization problem

$$\hat{w} = \arg\max_{w \in W} \frac{P(A|w)P(w)}{P(A)}. \tag{2}$$

Noting that $P(A)$ is the same for all $w \in W$, we get

$$\hat{w} = \arg\max_{w \in W} P(A|w)P(w). \tag{3}$$

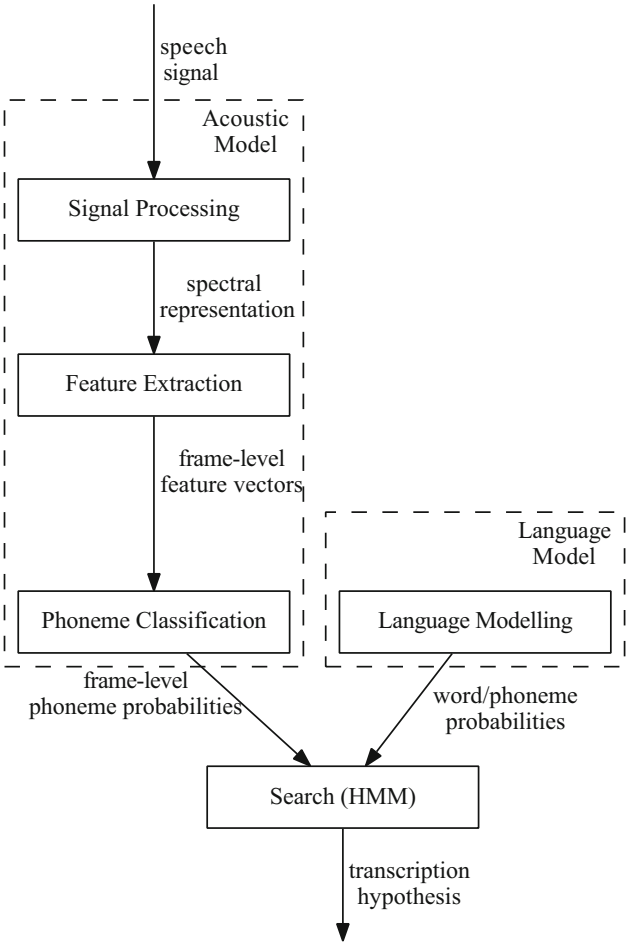

Fig. 1 Flow diagram of automatic speech recognition

The two factors represent two different sub-tasks of our problem: $P(A|w)$ (the *acoustic model*) describes the relation between the speech signal and the word, while $P(w)$ is the probability of w, being completely independent of the utterance A, and it is called the *language model* (Jelinek 1997). These models can be treated separately; in our study, we will focus on the first one.

Defining the word w as a phoneme sequence $o_1, \ldots, o_n$ allows us to divide A into nonoverlapping segments $A_1, \ldots, A_n$, each A_j belonging to the corresponding phoneme o_j. (As $A = a_1 \ldots a_t$, we can define A_j as $a_{t_{j-1}+1} \ldots a_{t_j}$ with $0 = t_0 < t_1 < \cdots < t_n = t$. The small, equal-sized a_i speech signal parts are the frames.) Then, making the common assumption that the phonemes are independent, we have

$$P(A|w) = \prod_{j=1}^{n} P(A_j|o_j). \tag{4}$$

Next, we will also assume that the a_i frames are independent of each other. Although this assumption (owing to the continuous movement of the lips, the vocal chords and so on) is actually false, it allows us to simplify our model further by using the formula

$$P(A_j|o_j) = \prod_{k=t_{j-1}+1}^{t_j} P(a_k|o_j). \tag{5}$$

Combining Eq. (4) and Eq. (5), we can write

$$P(A|w) = \prod_{i=1}^{t} P(a_i|o_j), \tag{6}$$

where j is the index of the phoneme assigned to the ith frame (a_i) (i.e. $t_{j-1} < i \leq t_j$). The sequence of the o_j states which maximized this formula is usually found via a Hidden Markov Model (HMM; Morgan and Bourland 1995).

The general workflow of automatic speech recognition is shown in Fig. 1. Notice that each a_i frame has the same length, so we can extract the same number of features from them, and the $P(a_i|o_j)$ likelihoods can be estimated via standard machine learning methods. But Eq. (6) also implies that the role of acoustic model is not to classify the frames, but to reliably estimate these $P(a_i|o_j)$ conditional likelihoods.

3 The phoneme classification sub-task

Besides the number of training examples and the size of the feature vector, there are other special characteristics of the phoneme classification task. Next we will describe the main points where it differs from a standard classification task and the way we will evaluate the results.

3.1 Tri-state phoneme representation

Firstly, it is common to define multiple classes (called *states*) for the same phoneme. The reason for this is that usually only the middle of an uttered phoneme is clear-cut, whereas the first and last parts are mixed with the neighbouring phonemes. Therefore, it seems sensible to assign a separate class for each part (*tri-state set-up*; Rabiner and Juang 1993). During recognition, these states have to be processed in an increasing order, which also defines a minimal length of each phoneme (Tóth et al. 2005). From a machine learning viewpoint, however, having multiple states introduces a correlation in the *classes*. Generative models such as GMM can easily handle this set-up, as they build a separate model for each class, independently of the others. However, if we shift to a discriminative classification technique, which seeks to find the *only* correct class label, their interpretation is not that plausible any more.

3.2 Dataset division

To perform speech recognition, a quite complex system has to be built that has a lot of parameters. To handle this in a fair way, the set of examples is usually divided into three parts. Phoneme classification methods are trained using the *training* set, and the rest of parameters are tuned on the *development* (or *validation*) set. The accuracy of the speech recognizer configuration is evaluated on the *test* set, using the parameter values obtained previously. In our case, AdaBoost.MH was trained on the training set, and we set the parameters of the calibration techniques on the development set. This set-up also has the advantage that we can measure the robustness of any improvement by examining how much of the improvement achieved on the development set could be transferred to the test set.

3.3 Evaluation

For standard classification tasks, usually just the classification accuracy is used for evaluation, calculated as the ratio of correctly classified examples. If the classes are unbalanced, we can calculate classification accuracy for each class, and then, we take the mean of these values; for information retrieval tasks, usually precision, recall and F-measure are computed (Manning et al. 2008).

However, phoneme classification is performed not to identify frames, but utterances. Although during training it is common to measure (and optimize for) frame-level accuracy, it is feasible to switch to utterance-level accuracy in the latter steps. Another reason for doing this is that the exact bounds between the consecutive phonemes cannot be decided objectively within one-frame precision (which is usually only 10 ms).

For the above reasons, phoneme classification accuracy is calculated at the utterance level. Given the transcriptions of the utterance and the phoneme sequence we got via speech recognition, first we calculate the phoneme-level edit distance (Levenshtein 1966) of the two phoneme sequences; i.e. we construct the resulting sentence from the real transcript by inserting and deleting phonemes, and replacing one phoneme with another one. These operations have some cost (we used the common values of 3, 3 and 4, respectively), so we choose an operation set that has the lowest overall cost. Now we can calculate the accuracy metric as

$$\text{Accuracy} = \frac{N - S - D - I}{N}, \tag{7}$$

where N is the total number of phonemes in all the original utterances, S is the number of substitutions, D is the number of deletions and I is the number of insertions.

4 AdaBoost.MH

Next, we will give a brief description of the AdaBoost.MH algorithm. Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ be the *observation matrix*, where $x_i^{(j)}$ are the elements of the d-dimensional observation vectors $\mathbf{x}_i \in \mathbb{R}^d$. Furthermore, let $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$ be a *label matrix* of dimension $n \times K$, where $\mathbf{y}_i \in \{+1, -1\}^K$. In a *multi-class* classification task, one and only one of the elements of $\mathbf{y}_i$ is $+1$; we will denote the index of the correct class by $\ell(\mathbf{x}_i)$.

The goal of the AdaBoost.MH algorithm (Schapire and Singer 1999) is to return a classifier $\mathbf{f}$: $\mathcal{X} \to \mathbb{R}^K$ with a small *Hamming loss*[1]

$$R_{\mathrm{H}}\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell}^{(1)} \mathbb{I}\left\{\operatorname{sign}\big(f_\ell^{(T)}(\mathbf{x}_i)\big) \neq y_{i,\ell}\right\} \tag{8}$$

by minimizing its upper bound (the *exponential margin loss*)

$$R_{\mathrm{e}}\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell}^{(1)} \exp\left(-f_\ell^{(T)}(\mathbf{x}_i) y_{i,\ell}\right), \tag{9}$$

where $f_\ell(\mathbf{x}_i)$ is the ℓth element of $\mathbf{f}(\mathbf{x}_i)$. The user-defined weights $\mathbf{W}^{(1)} = \left[w_{i,\ell}^{(1)}\right]$ are usually set either uniformly to

$$w_{i,\ell}^{(1)} = 1/(nK), \tag{10}$$

or, in multi-class classification, to

$$w_{i,\ell}^{(1)} = \begin{cases} \frac{1}{2n} & \text{if } \ell = \ell(\mathbf{x}_i) \text{ (i.e. if } y_{i,\ell} = 1), \\ \frac{1}{2n(K-1)} & \text{otherwise (i.e. if } y_{i,\ell} = -1) \end{cases} \tag{11}$$

to create K well-balanced one-against-all classification problems. AdaBoost.MH builds the final classifier $\mathbf{f}$ as a weighted sum of *base classifiers* $\mathbf{h}^{(t)}$: $\mathcal{X} \to \mathbb{R}^K$ returned by a *base learner* algorithm $Base(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)})$ for each iteration t. In general, the base learner should seek to minimize the *base objective*

$$E\big(\mathbf{h}, \mathbf{W}^{(t)}\big) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell}^{(t)} \exp\big(-h_\ell(\mathbf{x}_i) y_{i,\ell}\big). \tag{12}$$

In our tests, we used *discrete* AdaBoost.MH, in which the vector-valued base classifier $\mathbf{h}(\mathbf{x})$ is represented as

$$\mathbf{h}(\mathbf{x}) = \alpha \mathbf{v} \varphi(\mathbf{x}), \tag{13}$$

[1] The indicator function $\mathbb{I}\{A\}$ is 1 if its argument A is true and 0 otherwise.

where $\alpha \in \mathbb{R}^+$ is the *base coefficient*, $\mathbf{v} \in \{+1, -1\}^K$ is the *vote vector* and $\varphi(\mathbf{x})$: $\mathbb{R}^d \to \{+1, -1\}$ is a *scalar* base classifier. It can be shown that to minimize (12), one has to choose a $\mathbf{v}$ and a φ that maximize the *edge*

$$\gamma = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} v_\ell \varphi(\mathbf{x}_i) y_{i,\ell}. \tag{14}$$

The optimal coefficient is then

$$\alpha = \frac{1}{2} \log \frac{1+\gamma}{1-\gamma}. \tag{15}$$

The simplest scalar base learner used in practice is the *decision stump*, a two-leaf decision tree having the form

$$\varphi_{j,b}(\mathbf{x}) = \begin{cases} 1 & \text{if } x^{(j)} \geq b, \\ -1 & \text{otherwise,} \end{cases} \tag{16}$$

where j is the index of a feature and b is the decision threshold. Although boosting decision stumps often yields satisfactory results, state-of-the-art performance is usually achieved by using decision trees as base learners, parametrized by their number of leaves.

In the boosting framework, the decision stump is trained by using an exhaustive search which consists of computing the edge for each possible cut of each feature based on the training data and then picking the cut which maximizes the edge. As this learning procedure might be computationally expensive, we opted for an accelerated version of this training procedure. In this version, not all of the features are explored for each boosting step, but only a subset of them which was found to be useful in previous iteration steps. Since decision trees can be viewed as a set of stumps called in a recursive fashion, this idea can be extended to decision trees as well. We opted for this heuristical variation since its performance was reported to be on par with AdaBoost.MH, but it is an order of magnitude faster.

5 Posterior calibration

We shall assume that a multi-class classification algorithm such as AdaBoost.MH provides vector-valued multi-class discriminant functions in the form $\mathbf{f}$: $\mathcal{X} \to \mathbb{R}^K$, where $\mathcal{X}$ is the input space (in our case the space of phonemes represented by a real-valued feature vector) and K is the number of classes (now phoneme states). Elements of these vector-valued discriminant functions will be denoted by $\mathbf{f}(\mathbf{x}) = \big(f_1(\mathbf{x}), \ldots, f_K(\mathbf{x})\big)$.

A widely used performance evaluation metric in multi-class classification is the zero-one error, which is zero if an instance to be classified is correctly classified and one otherwise. Because of the nonconvexity of the zero-one error, so-called surrogate functions are used that give an upper bound for the error function, and more importantly, they are convex and thus are easy to optimize. For example, in the case of AdaBoost this surrogate function is the exponential loss, whereas Hinge loss is used for SVM. As an artefact of the application of such a surrogate function, the learning algorithms force discrimination by pulling the scores $f_\ell(\mathbf{x})$ away from zero. This means that direct (linear) conversion into class probabilities usually does not produce good estimates (Mease et al. 2007). This phenomenon is especially pronounced in the case of AdaBoost, because exponential loss increases sharply with negative margins (Niculescu-Mizil and Caruana 2005). At the same time, the score vector usually represents the *order* of the probability values rather well, so a simple nonlinear, monotonic function can transform the scores into good probability estimates. The process of learning this nonlinear function from held-out data is called *calibration* (Platt 2000). For the overall scheme of posterior calibration in ASR, see Fig. 2.

Next we will describe several calibration techniques, some of which were inspired by classical techniques tuned for squared error and cross-entropy (Niculescu-Mizil and Caruana 2005; Wu et al. 2004). The output scores of the AdaBoost.MH are real values ($\in \mathbb{R}^K$), but our goal is to obtain probabilities for each class. Therefore, the calibration function we shall consider is of the form $\mathbf{F}$: $\mathbb{R}^K \to [0, 1]^K$; the kth component of $\mathbf{F}$ is denoted by F_k and, with a slight abuse of notation, its argument is denoted here by $f_1, \ldots, f_K \in \mathbb{R}$ (without indicating the dependence on $\mathbf{x}$).

Note that most methods have parameters, which in theory can be set independently for each class. However, there is a large number of classes present in our case, meaning that there is a high risk of overfitting; so we decided to utilize a shared parameter vector for each class. This also simplifies our notation; as $F_1 = F_2 = \cdots = F_K$, we can simply write $F(\mathbf{f})$.

5.1 Linear scaling

Linear scaling consists of a linear mapping of each output score $f_1, \ldots, f_K$. It can be written in the form of

$$F_k^{\text{lin}}(f_1, \ldots, f_K) = \max\{0, af_k + b\}, \tag{17}$$

where $a, b \in \mathbb{R}$ are the parameters to be tuned. Since Eq. (17) is a convex function, the parameters a and b of the linear calibration function can be easily found by minimizing the squared error between the calibrated posterior estimates and the labels.

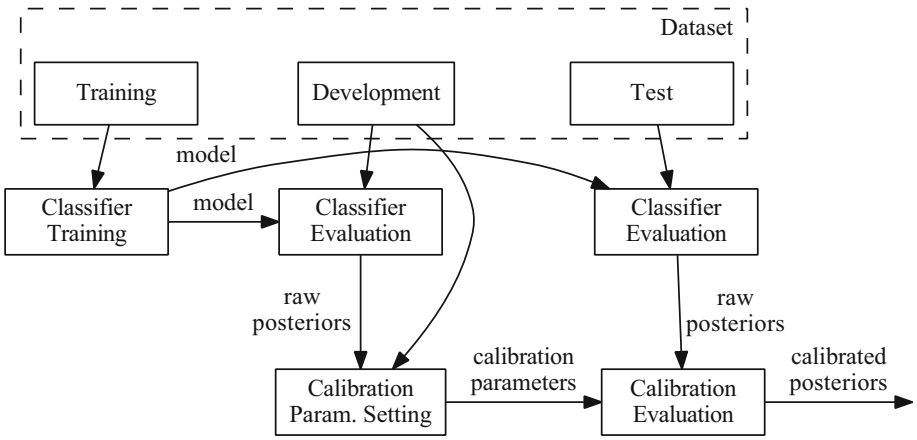

Fig. 2 Workflow of the posterior calibration process

5.2 Logistic correction

Logistic correction specifically targets AdaBoost. As described in Sect. 4, AdaBoost.MH builds its final model as the weighted sum of base classifiers. Assuming that the output of the ith base classifier for input $\mathbf{x}$ [i.e. $\mathbf{h}^{(i)}(\mathbf{x})$] falls in the range $[-1, 1]^K$ and its weight α_i is nonnegative, the output of the boosted model with T base classifiers for class k will be

$$f_k(\mathbf{x}) = \sum_{i=1}^{T} \alpha_i h_k^{(i)}(\mathbf{x}). \tag{18}$$

Friedman et al. (2000) showed that, in the binary case, discrete AdaBoost builds an additive logistic regression model by minimizing the expected exponential loss on the population level. For this, the weak learner needs to find the weak classifier which minimizes the weighted error also at a population level in each boosting iteration, where the weights are exponentially proportional to the negative margin of the current strong classifier. Next we are going to discuss how this result can be translated into the multi-class framework.

The population-level loss optimization of AdaBoost.MH had already been investigated with real-valued base classifiers by Friedman et al. (2000). Their analysis relies on a decoupling technique which consists of adding a virtual feature to the feature space that encodes the class label. Our analysis essentially differs from theirs, because first, we assume discrete multi-class weak classifiers, and second our analysis is tailored to the algorithm presented in Sect. 4 which uses some initial weighting over the labels. We will assume a weighting function in the form of $I\colon \{-1, 1\}^K \times [K] \to \mathbb{R}_+$. However, the main steps of our analysis follow the one given in Theorem 1 of Friedman et al. (2000).

Proposition 1 *Assume an arbitrary weighting function in the form of* $I\colon \{-1, 1\}^K \times [K] \to \mathbb{R}_+$ *which assigns a positive weight to the* ℓ*th component of a label vector* $\mathbf{y}$*. Then, AdaBoost.MH optimizes the multi-class exponential loss*

$$J(\mathbf{f}) = \mathbb{E}\left[\sum_{\ell=1}^{K} I(\mathbf{y}, \ell) \exp(-y_\ell f_\ell(\mathbf{x}))\right] \tag{19}$$

at the population level when the initial weighting was chosen according to $I(.,.)$. □

The proof of Proposition 1 is deferred to Appendix A.

Note that using AdaBoost results in an estimate for $\mathbf{f}^*$, the risk minimizer, using exponential loss based on a finite data. In a recent study (Bartlett and Traskin 2007), consistency of AdaBoost for the binary case has been proved; moreover, the rate of convergence was also computed under mild conditions. This means that the classifier output by AdaBoost is a "good" estimate of f^* on a large enough training data. To extend their result to the multi-class case is beyond the scope of this current study.

We discuss here the two most widely used initial weighting functions: **uniform weighting** can be defined as $I_u(\mathbf{y}, \ell) = 1$ for all $1 \le \ell \le K$, while the **one-against-all initial weighting** can be given as

$$I_o(\mathbf{y}, \ell) = \mathbb{I}\{y_\ell = 1\} + \tfrac{1}{K-1}\mathbb{I}\{y_\ell = -1\}. \tag{20}$$

Note that these weighting functions are the population-level counterparts of those given in Eqs. (10) and (11). Next, we will compute the optimal solution of the multi-class exponential loss with these weighting functions, which can then be used to devise a reasonable calibration function.

Proposition 2 *The multi-class exponential loss*

$$J(\mathbf{f}) = \mathbb{E}\left[\sum_{\ell=1}^{K} I(\mathbf{y}, \ell) \exp(-y_\ell f_\ell(\mathbf{x}))\right]$$

with $I_u(\mathbf{y}, \ell) = 1$ *for all* $1 \leq \ell \leq K$ *is minimized by*

$$f_i(\mathbf{x}) = \frac{1}{2} \log \frac{p_i(\mathbf{x})}{1 - p_i(\mathbf{x})},$$

where $1 \leq i \leq K$. *Thus,*

$$p_i(\mathbf{x}) = \frac{1}{1 + \exp(-2 f_i(\mathbf{x}))}. \tag{21}$$

Furthermore, $J(\mathbf{f})$ *with uniform initial weighting defined in (20) is minimized by*

$$f_i(\mathbf{x}) = \frac{1}{2} \log \frac{p_i(\mathbf{x})}{1 - p_i(\mathbf{x})} + \frac{1}{2} \log(K - 1),$$

where $1 \leq i \leq K$. *Thus,*

$$p_i(\mathbf{x}) = \frac{1}{1 + \exp(\log(K - 1) - 2 f_i(\mathbf{x}))}. \tag{22}$$

Proof One may compute the partial derivative of the conditional expectation written as

$$\mathbb{E}\left[\sum_{\ell=1}^{K} I(\mathbf{y}, \ell) \exp(-y_\ell f_\ell(\mathbf{x})) \middle| \mathbf{x} \right]$$

with respect to $f_\ell(\mathbf{x})$. Then, one can obtain the result by setting the partial derivatives to zero. □

Note that the uniform and one-against-all initial weightings result in very similar probability estimates, given in Eqs. (21) and (22), respectively, the difference being a constant offset in terms of the output score of the strong classifier. This suggests that a logistic correction of the output scores of AdaBoost in the form of Eqs. (21) and (22) results in precise posterior estimates. In practice, it may worth reparametrizing the sigmoid function by introducing a bias term which might lead to better performance. Hence, we can generalize this formula to

$$F_k^{\log}(f_1, \ldots, f_K) = \frac{1}{1 + e^{a f_k(\mathbf{x}) + b}}, \tag{23}$$

where $a, b \in \mathbb{R}$ are parameters. The calibration function was fitted in a standard L2 set-up by minimizing the squared loss.

5.3 Platt scaling

Platt originally used a sigmoid function to map the outputs of an SVM to posterior scores (Platt 2000). A similar solution can be applied for AdaBoost as well. In this formulation, we expect the raw (uncalibrated) outputs of AdaBoost to lie in the range $[-1, -1]$, which can be achieved in a straightforward way by dividing the values in Eq. (18) by the sum of the α_i values. That is,

$$\mathbf{f}(\mathbf{x}) = \frac{\sum_{i=1}^{T} \alpha_i h_i(x)}{\sum_{i=1}^{T} \alpha_i}. \tag{24}$$

To get calibrated probability values, we pass the output of boosting through a sigmoid function; that is,

$$P(y = 1|x) = \frac{1}{1 + e^{a\mathbf{f}(\mathbf{x}) + b}}, \tag{25}$$

where the parameters a and b are fitted using maximum likelihood estimation on a calibration set. Therefore, in the binary case (i.e. y_i is either zero or one), a and b are solutions to the minimization problem

$$\arg\min_{a,b} \left\{ -\sum_{i=1}^{T} y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \right\}, \tag{26}$$

where

$$p_i = \frac{1}{1 + e^{a\mathbf{f}_i + b}}. \tag{27}$$

The straightforward generalization of Eq. (26) for the multiclass case is

$$\arg\min_{a,b} \left\{ -\sum_{i=1}^{T} \log(p_{y_i}) \right\}. \tag{28}$$

5.4 Isotonic regression

Another posterior calibration technique we tested is Isotonic Regression (Robertson et al. 1988; Zadrozny and Elkan 2001). This method just assumes that

$$y_i = m(\mathbf{f}_i) + \epsilon_i, \tag{29}$$

where m is an isotonic (monotonically increasing) function. Given the $\mathbf{f}_i$ and y_i vectors of the training set, we have to find an isotonic function $\hat{m}$ for which the ϵ values are minimal; that is,

$$\hat{m} = \arg\min_{m} \sum (y_i - m(\mathbf{f}_i))^2. \tag{30}$$

A piecewise constant solution for $\hat{m}$ can be found in linear time by applying the Pair Adjacent Violators (PAV) algorithm (Ayer et al. 1955).

Table 1 Some key properties of the TIMIT and the Hungarian Numbers speech database

	No. of speakers	No. of utterances	Total length (m:s)	No. of frames
TIMIT				
Training set	440	3520	179:01	1,074,130
Development set	22	176	8:27	50,693
Test set	24	192	9:39	57,919
Total	486	3888	187:28	1,124,823
Hungarian Numbers				
Training set	10	520	9:42	58,165
Development set	5	100	4:19	25,867
Test set	5	100	4:10	25,006
Total	20	720	18:10	109,038

6 Experimental set-up

6.1 Databases

We performed our experiments on two databases. (For the key properties of the two corpora, see Table 1.) The first one was the TIMIT dataset (Lamel et al. 1986), which contains the utterances of North American speakers. All the 61 phonemes were used in a tri-state set-up, resulting in 183 classes overall. The second dataset used in our tests contained recordings of Hungarian Numbers; since in this dataset there were 33 phonemes defined, we had 99 classes overall.

6.2 Training parameters

Training of AdaBoost was carried out using the standard MFCC $+ \Delta + \Delta\Delta$ feature set (Rabiner and Juang 1993), which contains 12 Mel-spectral cepstral coefficients along with energy, and the first- and second-order derivatives (39 attributes overall). To improve the performance, we included the feature vectors of the 8 preceding and 8 following frames, while keeping the original class label. These features were extracted by the HTK toolkit (Young et al. 2006). To evaluate a posterior matrix, we performed a search using a HMM with constant state transition probabilities and calculated the accuracy metric described in Eq. (7) for the resulting phoneme sequence.

We tested simple decision stumps and decision trees with 8 leaves as base learners (being the standard size, e.g. Busa-Fekete and Kégl 2009), using the implementation `multiboost` (Benbouzid et al. 2012). We performed training for 100,000 iterations, evaluating the models after every 10,000 iterations for both methods and data subsets for the TIMIT corpus, while for the Hungarian Numbers corpus, since we had much less training data there, we trained our models for 50,000 iterations. For further details of the training process, see Gosztolya (2014).

6.3 Baseline methods

AdaBoost.MH constructs the final classifier by calculating a weighted sum of h_i base classifiers, each one returning an output score for all the classes in the range $[-1, 1]$. This means that we get values in the range $[0, 1]$ by dividing the resulting output scores by twice the sum of the α_i base classifier weights and adding 0.5. This quite trivial posterior calibration method served as our baseline with the constraint that the resulting values had to be transformed to sum up to one.

As a reference, we also tested a search method that uses no posterior scores at all. For this, for each a_i frame we chose the phoneme state (i.e. class) which was considered the most probable by AdaBoost.MH; then, our search algorithm just relied on these frame-level class labels. This dynamic search method sought to minimize the number of substituted frame labels, with the restrictions that the first frame of the utterance had to be a first state of some phoneme, the last frame had to be a final state of some phoneme and the three states of any phoneme had to be present in increasing order.

6.4 Normalization

For a given frame, a classifier method is expected to return $P(o_j|a_i)$ values which sum up to one. However, after posterior calibration, where we fit a function to the raw $\mathbf{f}(\mathbf{x})$ output values of AdaBoost.MH, this requirement is unlikely to be satisfied. To ensure that this property holds for every case, we inserted another step into our workflow; namely, we normalized the values got by applying the methods presented in Sect. 5 to sum up to one. Of course, calculating the values for any error function was carried out only after performing this normalization step.

6.5 Parameter optimization

From the methods applied, the two baseline techniques and isotonic regression had no parameter at all. The remaining

Table 2 Phonetic accuracy scores obtained by using the different calibration methods and parameter optimization techniques on the TIMIT dataset

Optimization	Calibration method	Stumps		Trees	
		Dev. (%)	Test (%)	Dev. (%)	Test (%)
Frame-level	Linear scaling	68.85	66.17	75.49	72.89
	Logistic correction	68.90	66.11	75.55	73.01
	Platt scaling	68.87	66.41	75.59	73.05
	Isotonic regression	68.88	66.53	75.70	73.14
Speech recognition	Linear scaling	68.85	66.17	75.50	73.08
	Logistic/Platt	69.12	66.40	75.98	73.33
No calibration (baseline)		68.82	66.14	75.52	72.82
Without posterior scores		65.91	63.68	74.09	70.76

three methods all had two parameters [see Eqs. (17), (23] and (25)). For such a small number of parameters, the most straightforward way of optimizing the parameters is to apply a *grid search* (Ensor and Glynn 1997).

Platt's method in fact has the same transformation function as logistic regression does; the difference is the error function to be optimized and that Platt proposed to utilize gradient descent to get the best parameter setting. Unfortunately, as we had to perform normalization after calculating the sigmoid function in Eq. (25), we cannot calculate the derivatives; so we applied grid search here as well. We treated the two methods as distinct, though, as they had different error functions.

The standard approach for parameter optimization in posterior calibration methods is to do it in an example-based fashion, e.g. via Eq. (28) for Platt scaling. Now, however, we are not interested in some frame-level accuracy score, as we seek to improve *utterance-level* phonetic accuracy instead, which involves combining the frame-level calibrated likelihoods. Therefore, we experimented with a further optimization approach: we performed posterior calibration (involving normalization) for all frames, and then, we performed the HMM search. We chose the calibration meta-parameter vector for which the accuracy of the phoneme sequences returned for all utterances was the highest. We will refer to this procedure as "optimization by speech recognition" or simply by "ASR" later on.

As logistic regression and Platt's calibration differed only in the example-level error function, they were practically identical in this case, leaving us with just two methods (i.e. linear and logistic/Platt's).

7 Results

7.1 TIMIT corpus

Table 2 shows the phonetic accuracy scores we got by using the different posterior calibration methods for both base learners tested. It can be seen that there is a big gap between utilizing the information stored in the posterior scores and just using the class label of each frame ("Without posterior scores" case): it leads to an improvement of 2.5–3% for decision stumps and 2–2.5% for decision trees. The baseline scores ("No calibration") are surprisingly good, especially considering the simplistic nature of this process.

However, except for a few cases, the posterior calibration methods were able to outperform the baseline scores, sometimes quite significantly. Among the approaches, perhaps linear scaling was the least effective, which can be explained by its simplicity. Transforming the raw classifier outputs via a sigmoid function, however, seems to be a good approach, while isotonic regression performed surprisingly well. In our opinion, this is because this method has no meta-parameters at all; and since we have relatively few data in the development set, the meta-parameters of the calibration techniques can be tuned only in a way which is moderately robust. Isotonic correction, however, had no meta-parameter to set, and therefore, it could lead to a more robust calibration model.

Among the two optimization categories, the direct optimization of speech recognition accuracy resulted in higher scores than optimizing some frame-level (example-based) function. Even for the quite simple linear scaling method, we got a 0.2% improvement in the case of decision trees in the test accuracy, and for the logistic/Platt scaling case the improvement was 0.3% on the test set, which meant a relative error reduction (RER) score of 1%. The reason for it is probably that it makes sense to directly optimize for the accuracy of our whole (speech recognition) system instead of some intermediate step (e.g. frame-level classification accuracy). Overall, we managed to improve the accuracy scores over the baseline by 1–2% relative error-wise, which is a significant achievement by speech recognition standards.

The phonetic accuracy scores on both the development and test sets as a function of the stopping iteration are shown in Figs. 3 and 4 for decision stumps and decision trees as base learners, respectively. When using the decision stumps as base learners, the optimum on the development set is around 50,000–60,000 iterations; after that, the phonetic accuracy

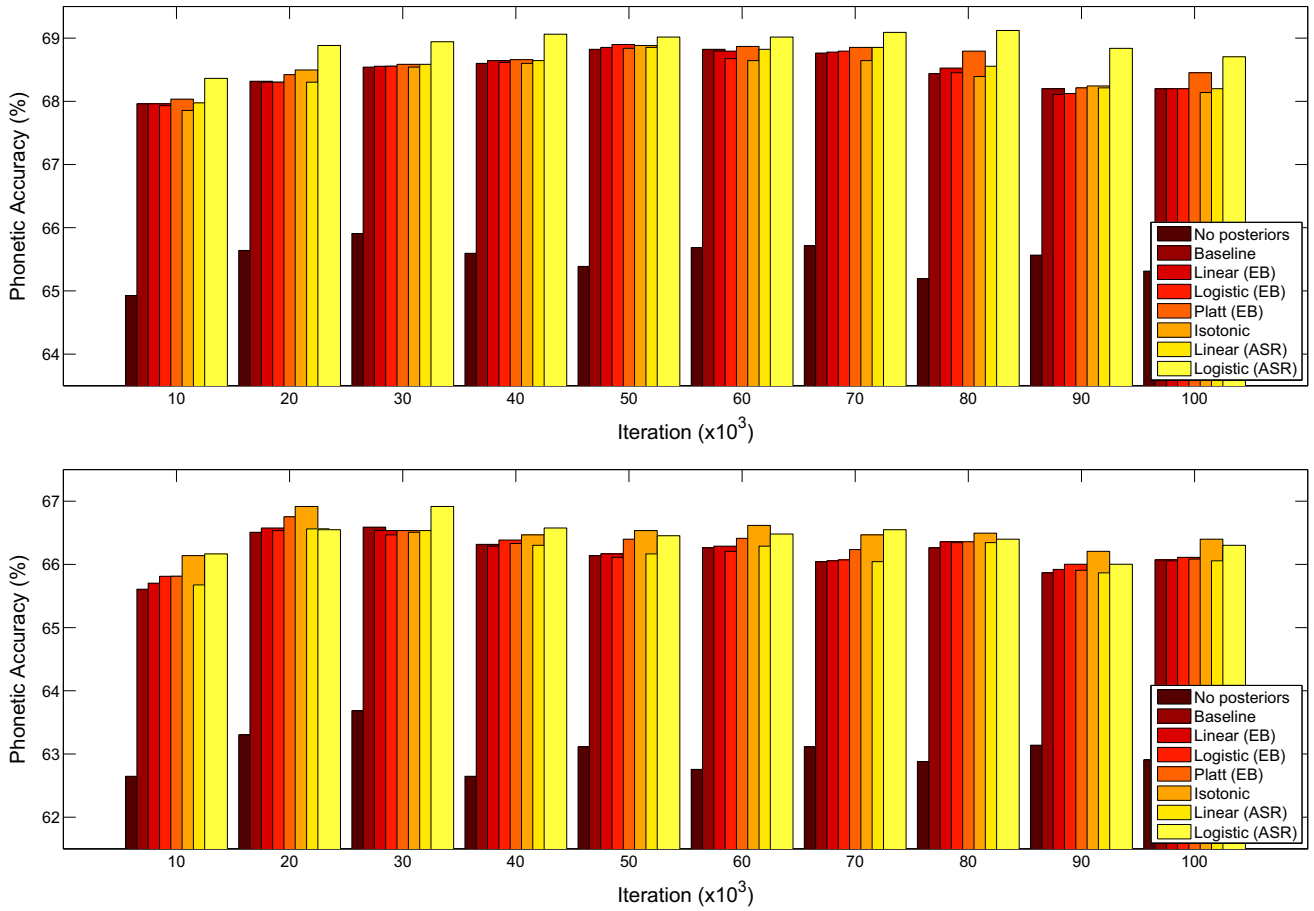

Fig. 3 Phonetic accuracy scores got from applying the different posterior calibration methods as a function of the AdaBoost.MH halting iterations on the development set (up) and on the test set (down) of the TIMIT corpus, when using decision stumps as base learners

starts to worsen. Surprisingly, on the test set the optimal phonetic accuracy scores appear around 20,000–30,000 iterations, but the values in the latter iterations are only slightly lower. When we applied decision trees, the accuracy scores display a general increasing trend, indicating that 100,000 or more iterations are required to achieve optimal phoneme classification scores. (In some cases, the actual optimum could be just before that, though.) In our opinion, this reflects the fact that phoneme classification is such a complex task that is cannot be efficiently handled by simple decision stumps, but it is worth employing the slightly more complicated (small) decision trees instead.

Regarding the posterior calibration methods, it is quite apparent that the logistic/Platt method (optimized via ASR) performs consistently better than the others using both base learners on both sets. Apart from this, after optimizing in the example-based manner, the methods which utilize the posterior scores produce quite similar accuracy scores.

Overall, it is clear that the accuracy scores could be improved significantly by posterior calibration. Although an improvement of 0.5% (2% RER) may not seem high by machine learning standards, it is still a significant one in ASR, reflecting a definite improvement in the recognition performance. This is especially true if we consider that it was achieved by applying a simple transformation of the raw output scores of a classifier, which is computationally extremely cheap.

7.2 Hungarian Numbers corpus

Table 3 shows our results obtained for the Hungarian Numbers corpus. Overall, our key findings are similar as those on the TIMIT dataset: decision trees are more effective as base learners than decision stumps are, and using posterior estimates (even if they are the standard ones) leads to higher accuracy scores than relying only on the optimal frame-level phonetic labels.

Regarding the posterior calibration techniques, they led to improvements in the phonetic accuracy in almost every case, using linear calibration with decision stumps as base learners being the only exception. In general, linear calibration performed worst: even with decision trees, the improvement in

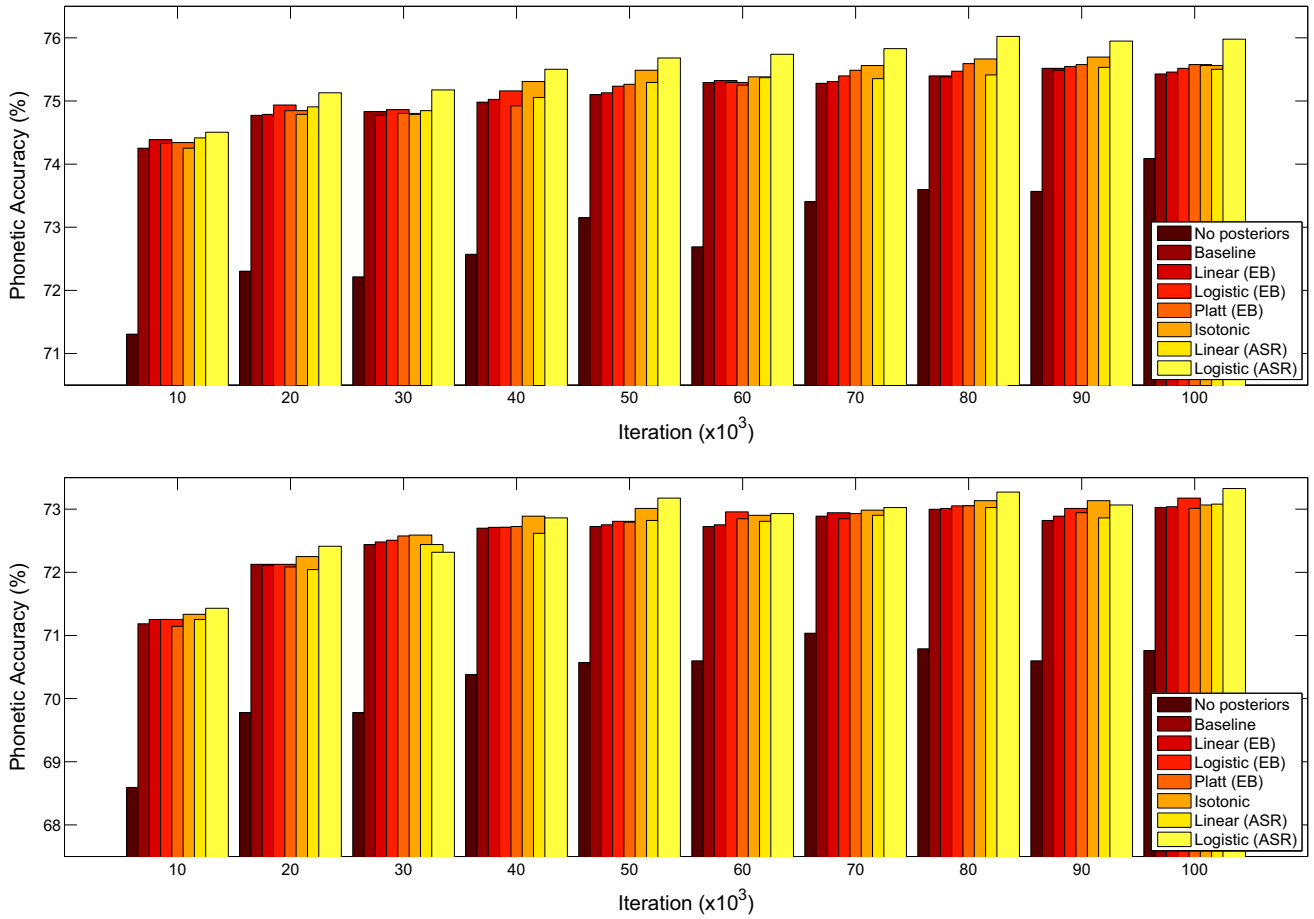

Fig. 4 Phonetic accuracy scores got from applying the different posterior calibration methods as a function of the AdaBoost.MH halting iterations on the development set (up) and on the test set (down) of the TIMIT corpus, when using decision trees as base learners

Table 3 Phonetic accuracy scores obtained by using the different calibration methods and parameter optimization techniques on the Hungarian Numbers dataset

Optimization	Calibration method	Stumps		Trees	
		Dev. (%)	Test (%)	Dev. (%)	Test (%)
Frame-level	Linear scaling	50.18	59.41	56.72	65.30
	Logistic correction	50.55	59.52	57.09	65.63
	Platt scaling	50.37	59.81	57.09	65.52
	Isotonic regression	50.22	59.93	57.20	65.56
Speech recognition	Linear scaling	50.37	59.74	56.87	65.48
	Logistic/Platt	50.78	59.96	57.72	65.81
No calibration (baseline)		50.19	59.41	56.54	65.19
Without posterior scores		49.70	58.11	55.72	63.82

the test set is only 0.11%, which is hardly significant. Logistic correction and Platt's scaling methods, being inherently similar, led to similar accuracy scores as well, but isotonic regression turned out to be the most effective method again.

Furthermore, we found using speech recognition to fine-tune the calibration meta-parameters beneficial for this dataset as well. Although the amount of improvement which can be obtained via linear calibration is limited, fitting a sigmoid function to the raw outputs of AdaBoost.MH proved to be beneficial: the accuracy scores improved by 0.55% and 0.62%, using decision stumps and decision trees as base learners, respectively.

8 Calibration time requirements

For any machine learning technique, besides accuracy, its execution time is also quite important. Therefore, it is worth

Table 4 Average time of finding the optimal calibration parameters for the Hungarian Numbers dataset, using decision trees as base learners

Optimization	Calibration method	Time (s)
Frame-level	Linear scaling	6.33
	Logistic correction	1315.98
	Platt scaling	9.30
	Isotonic regression	27.03
Speech recognition	Linear scaling	25.25
	Logistic/Platt	5550.05

examining the computational cost of the different calibration methods applied. Since our study is an application-oriented one, we were interested in the actual execution times measured instead of theoretical computational complexity. Furthermore, notice that the actual posterior transformation methods (see Sect. 5) do not require that much calculation: in fact, they are practically negligible compared to evaluating a machine learning method such as Deep Neural Networks or AdaBoost.MH, or to the other components of a speech recognition framework. However, the time requirement of finding their optimal meta-parameters (e.g. a and b of Platt's method, or the piecewise linear function used in isotonic regression) might be quite time-consuming. Therefore, in the following, we will present the measured wall-clock times of parameter calibration.

We will present the values measured for the decision trees case for the Hungarian Numbers corpus; of course, the scores appeared to be similar when using decision trees. For the TIMIT corpus, the times were similar, but proportionally larger, since it had a larger development set (see Table 1) used in the posterior calibration process.

Table 4 shows the measured meta-parameter setting times. We can see that the scores vary to a high extent; the main reason for it is the number of parameter variations which had to be tested. Linear scaling was quite quick in this aspect, but it also led to the worst scores of all the techniques tested. For Platt scaling, the use of optimization for meta-parameter setting speeded up the calibration process, while for logistic correction, we had to use grid search. Isotonic regression was also relatively fast.

It is also quite clear that incorporating the speech recognition system into the function to optimize adds quite a large overhead to the whole calibration process, as the measured times are cca. 4 times higher than the corresponding the frame-level ones are. On the other hand, the resulting accuracy values were also higher. Inspecting this table, however, we can see that the high performance of using the "logistic / Platt" approach and setting its a and b parameters via speech recognition is paired with quite high execution times as well. Therefore, relying on isotonic regression seems to be a good compromise, as it also led to high accuracy scores, but its execution time appeared to be quite low.

9 Conclusions

The standard workflow of automatic speech recognition relies heavily on the phoneme classification or phoneme posterior estimation sub-task. There, machine learning methods are used to provide local probability estimates for the phonemes of the given language. Since in a latter step these local estimates are combined to form utterance-level hypotheses of what the speaker said, it is quite important that the machine learning method used should not only find the correct phoneme from the local information, but the posterior estimates it returns should be precise as well. In an earlier study we showed that AdaBoost.MH, an iterative meta-learning algorithm can be effectively used in speech recognition; however, it is well known that the likelihoods this machine learning method supplies are quite imprecise. In this study, we applied several state-of-the-art posterior calibration methods to correct the tendency of the posterior scores returned. In the end, we found that by using these techniques, significant improvements can be achieved in the utterance-level accuracy of speech recognition. In our opinion, these improvements are even more valuable since the calibration methods utilized have quite low computational costs. We also found that instead of setting the parameters of the calibration approaches at the example level as is common in machine learning, it is worth incorporating parameter optimization into the speech recognition process, as the accuracy scores obtained this way were notably higher.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

A Proof of Proposition 1

Proof Let us assume a strong classifier $\mathbf{f}^{(T)}(\mathbf{x})$. Then, we seek to find weak classifier $\mathbf{h}(.)$ such that $J(\mathbf{f}^{(T)} + c\mathbf{h})$ is minimized. By using the linearity of the expectation, one may consider the second-order expansion of $J(\mathbf{f}^{(T)} + c\mathbf{h})$ for fixed $c \in \mathbb{R}_+$ and $\mathbf{h}(x) = (0, \ldots, 0)$ as

$$
\begin{aligned}
&J(\mathbf{f}^{(T)} + c\mathbf{h}) \\
&\quad = \mathbb{E}\left[\sum_{\ell=1}^{K} I(\mathbf{y}, \ell) \exp(-y_\ell(f_\ell^{(T)}(\mathbf{x}) + ch_\ell(\mathbf{x})))\right] \\
&\quad = \sum_{\ell=1}^{K} \mathbb{E}\left[I(\mathbf{y}, \ell) \exp(-y_\ell(f_\ell^{(T)}(\mathbf{x}) + ch_\ell(\mathbf{x})))\right]
\end{aligned}
$$

$$\approx \sum_{\ell=1}^{K} \mathbb{E}\Big[I(\mathbf{y}, \ell) \exp(- y_\ell f_\ell^{(T)}(\mathbf{x})) \\ (1 - c y_\ell h_\ell(\mathbf{x}) + c_\ell^2 y_\ell^2 h_\ell(\mathbf{x})^2/2)\Big] \\ = \sum_{\ell=1}^{K} \mathbb{E}\Big[I(\mathbf{y}, \ell) \exp(- y_\ell f_\ell^{(T)}(\mathbf{x})) \\ (1 - c y_\ell h_\ell(\mathbf{x}) + c_\ell^2/2)\Big]$$

where the last equation holds because $h_\ell(\mathbf{x}) \in \{-1, 1\}$ and $y_\ell \in \{-1, 1\}$. This means that minimizing the multi-class exponential loss with respect to $\mathbf{h}$ is equivalent to maximizing the weighted expectation

$$\mathbb{E}\left[\sum_{\ell=1}^{K} w(\mathbf{x}, \mathbf{y}, \ell) y_\ell h_\ell(\mathbf{x}) \right] \tag{31}$$

where

$$w(\mathbf{x}, \mathbf{y}, \ell) = I(\mathbf{y}, \ell) \exp\left(- y_\ell f_\ell^{(T)}(\mathbf{x}) \right).$$

Note that the weak learner maximizes the edge given in (14) computed on a finite dataset, which is an estimate of the weighted expectation.

The label distribution in the multi-class case for fixed $\mathbf{x}$ is a multinomial distribution with $(p_1, \ldots, p_K) \in \Delta_K$, where Δ_K is the K dimensional probability simplex. Note that $(p_1, \ldots, p_K)$ depends on $\mathbf{x}$, but with a slight abuse of notation we shall not indicate this dependence if $\mathbf{x}$ is fixed and this does not lead to any confusion. We shall write $(p_1(\mathbf{x}), \ldots, p_K(\mathbf{x}))$ to denote the dependence on $\mathbf{x}$.

Recall that the goal is to maximize the weighted expectation given in (31). As a next step we are going to compute the form of the optimal weak classifier. Let us define the vector $\mathbf{e}_\ell$ as

$$e_{\ell,\ell'} = \begin{cases} 1, & \ell = \ell' \\ -1, & \text{otherwise} \end{cases}$$

Furthermore, assume that we are given a weak classifier $\mathbf{h}(.)$ such that $h_{\widehat{\ell}}(\mathbf{x}) = 1$ for some $\widehat{\ell}$ and $h_\ell(\mathbf{x}) = 0$ otherwise. Then, the numerator of (31) can be written for a fixed $\mathbf{x}$ with a label distribution $(p_1, \ldots, p_K)$ as

$$\mathbb{E}\left[\sum_{\ell=1}^{K} w(\mathbf{x}, \mathbf{y}, \ell) y_\ell h_\ell(\mathbf{x}) | \mathbf{x} \right] \\ = \sum_{\ell=1}^{K} p_\ell \left[w(\mathbf{x}, \mathbf{e}_\ell, \ell) h_\ell(\mathbf{x}) - \sum_{\ell' \neq \ell} w(\mathbf{x}, \mathbf{e}_\ell, \ell') h_{\ell'}(\mathbf{x}) \right] \\ = p_{\widehat{\ell}} \left[w(\mathbf{x}, \mathbf{e}_{\widehat{\ell}}, \widehat{\ell}) + \sum_{\ell' \neq \widehat{\ell}} w(\mathbf{x}, \mathbf{e}_{\widehat{\ell}}, \ell') \right] \\ + \sum_{\ell \neq \widehat{\ell}} p_\ell \left[- w(\mathbf{x}, \mathbf{e}_\ell, \ell) - w(\mathbf{x}, \mathbf{e}_\ell, \widehat{\ell}) + \sum_{\ell' \neq \ell, \widehat{\ell}} w(\mathbf{x}, \mathbf{e}_\ell, \ell') \right] \\ = p_{\widehat{\ell}} \sum_{\ell=1}^{K} w(\mathbf{x}, \mathbf{e}_{\widehat{\ell}}, \ell) \\ + \sum_{\ell \neq \widehat{\ell}} p_\ell \left[- 2 w(\mathbf{x}, \mathbf{e}_\ell, \ell) - 2 w(\mathbf{x}, \mathbf{e}_\ell, \widehat{\ell}) + \sum_{\ell'=1}^{K} w(\mathbf{x}, \mathbf{e}_\ell, \ell') \right] \\ = \sum_{\ell=1}^{K} \sum_{\ell'=1}^{K} w(\mathbf{x}, \mathbf{e}_\ell, \ell') - 2 \sum_{\ell \neq \widehat{\ell}} p_\ell \left[w(\mathbf{x}, \mathbf{e}_\ell, \ell) + w(\mathbf{x}, \mathbf{e}_\ell, \widehat{\ell}) \right] \\ = \sum_{\ell=1}^{K} \sum_{\ell'=1}^{K} w(\mathbf{x}, \mathbf{e}_\ell, \ell') - 2 \sum_{\ell=1}^{K} p_\ell w(\mathbf{x}, \mathbf{e}_\ell, \ell) \\ + 2 \left[p_{\widehat{\ell}} w(\mathbf{x}, \mathbf{e}_{\widehat{\ell}}, \widehat{\ell}) - \sum_{\ell \neq \widehat{\ell}} p_\ell w(\mathbf{x}, \mathbf{e}_\ell, \widehat{\ell}) \right] \tag{32}$$

Since only the last term depends on $\widehat{\ell}$, the optimal classifier can be written in the form

$$\widehat{\ell}(\mathbf{x}) = \arg\max_{1 \leq \ell \leq K} \\ \left[p_\ell(\mathbf{x}) w(\mathbf{x}, \mathbf{e}_\ell, \ell) - \sum_{\ell' \neq \ell} p_{\ell'}(\mathbf{x}) w(\mathbf{x}, \mathbf{e}_{\ell'}, \ell) \right] \\ = \arg\max_{1 \leq \ell \leq K} \left[p_\ell(\mathbf{x}) w(\mathbf{x}, \mathbf{e}_\ell, \ell) \right],$$

and hence, the optimal classifier $\mathbf{h}(\mathbf{x})$ can be written in the form of

$$h_\ell(\mathbf{x}) = \begin{cases} 1, & \text{if } \ell = \widehat{\ell}(\mathbf{x}) \\ -1, & \text{otherwise} \end{cases}. \tag{33}$$

Considering (32) over the joint distribution of the labels and features, we get that the classifier which is in the form of (33) minimizes the population-level edge defined as

$$\gamma^* = \mathbb{E}\left[\sum_{\ell=1}^{K} w(\mathbf{x}, \mathbf{y}, \ell) y_\ell h_\ell(\mathbf{x}) \right],$$

where the expectation is taken over the joint distribution.

As a next step, one can determine the optimal coefficient c of $\mathbf{h}(.)$ given in (33) by optimizing $J(\mathbf{f}^{(T)} + c\mathbf{h})$:

$$c = \underset{0<c}{\operatorname{argmin}}\, \mathbb{E}\left[\sum_{\ell=1}^{K} w(\mathbf{x}, \mathbf{y}, \ell) \exp(- c y_\ell h_\ell(\mathbf{x})) \right] \\ \leq \underset{0<c}{\operatorname{argmin}}\, \mathbb{E}\left[\sum_{\ell=1}^{K} w(\mathbf{x}, \mathbf{y}, \ell) \left(\frac{1 + y_\ell h_\ell(\mathbf{x})}{2} \exp(-c) \right. \right. \\ \left. \left. + \frac{1 - y_\ell h_\ell(\mathbf{x})}{2} \exp(c) \right) \right]$$

Then, one can compute analytically that the right-hand side is minimized by

$$c = \frac{1}{2} \log \frac{1+\gamma^*}{1-\gamma^*},$$

where γ^* is the population-level edge. Note that the AdaBoost.MH computes the coefficient of the weak classifier in a similar way by using the edge on a finite training data instead of taking the population-level edge. The rest of the proof regarding the weight update is analogous to that of Theorem 1 in Friedman et al. (2000). □

References

Ayer M, Brunk H, Ewing G, Reid W, Silverman E (1955) An empirical distribution function for sampling with incomplete information. Ann Math Stat 5(26):641–647

Bartlett PL, Traskin M (2007) AdaBoost is consistent. J Mach Learn Res 8:2347–2368

Benbouzid D, Busa-Fekete R, Casagrande N, Collin FD, Kégl B (2012) MultiBoost: a multi-purpose boosting package. J Mach Learn Res 13:549–553

Bishop CM (1995) Neural networks for pattern recognition. Clarendon Press, Oxford

Bodnár P, Nyúl LG (2015) Improved QR code localization using boosted cascade of weak classifiers. Acta Cybern 22(1):21–33

Busa-Fekete R, Kégl B (2009) Accelerating AdaBoost using UCB. In: KDDCup 2009 (JMLR W&CP), vol 7, pp 111–122, Paris, France

Busa-Fekete R, Kégl B, Éltetö T, Szarvas G (2013) Tune and mix: learning to rank using ensembles of calibrated multi-class classifiers. Mach Learn 93(2–3):261–292

Crammer K, Singer Y (2001) On the algorithmic implementation of multiclass kernel-based vector machines. J Mach Learn Res 2:265–292

Drish J (2001) Obtaining calibrated probability estimates from support vector machines. Technical report, University of California, San Diego, CA, USA

Duda RO, Hart PE (1973) Pattern classification and scene analysis. Wiley, New York

Ensor KB, Glynn PW (1997) Stochastic optimization via grid search. In: Lectures in Applied Mathematics, vol 33. American Mathematical Society, pp 89–100

Friedman J, Hastie T, Tibshirani R (2000) Additive logistic regression: a statistical view of boosting. Ann Stat 28:337–374

Gosztolya G (2014) Is AdaBoost competitive for phoneme classification? In: Proceedings of CINTI (IEEE), pp 61–66, Budapest, Hungary

Gosztolya G (2015) On evaluation metrics for social signal detection. In: Proceedings of InterSpeech, pp 2504–2508, Dresden, Germany

Gosztolya G, Busa-Fekete R, Tóth L (2013) Detecting autism, emotions and social signals using AdaBoost. In: Proceedings of InterSpeech, pp 220–224, Lyon, France

Gosztolya G, Beke A, Neuberger T, Tóth L (2016) Laughter classification using deep rectifier neural networks with a minimal feature subset. Arch Acoust 41(4):669–682

Gupta R, Audhkhasi K, Lee S, Narayanan SS (2013) Speech paralinguistic event detection using probabilistic time-series smoothing and masking. In: Proceedings of InterSpeech, pp 173–177

Imseng D, Bourlard H, Magimai-Doss M, Dines J (2011) Language dependent universal phoneme posterior estimation for mixed language speech recognition. In: Proceedings of ICASSP, pp. 5012–5015, Prague, Czech Republic

Jelinek F (1997) Statistical methods for speech recognition. MIT Press, Cambridge

Kaya H, Karpov AA, Salah AA (2015) Fisher Vectors with cascaded normalization for paralinguistic analysis. In: Proceedings of InterSpeech, pp 909–913

Lamel L, Kassel R, Seneff S (1986) Speech database development: design and analysis of the acoustic-phonetic corpus. In: Proceedings of DARPA speech recognition workshop, pp 121–124

Levenshtein VI (1966) Binary codes capable of correcting deletions, insertions, and reversals. Sov Phys Dokl 10(8):707–710

Manning C, Raghavan P, Schütze H (2008) Introduction to information retrieval. Cambridge University Press, Cambridge

Mease D, Wyner A, Buja A (2007) Boosted classification trees and class probability/quantile estimation. J Mach Learn Res 8:409–439

Morgan N, Bourland H (1995) An introduction to hybrid HMM/connectionist continuous speech recognition. Signal Process Mag 1025–1028, May 1995

Neuberger T, Beke A (2013) Automatic laughter detection in spontaneous speech using GMM–SVM method. In: Proceedings of TSD, pp 113–120

Niculescu-Mizil A, Caruana R (2005) Obtaining calibrated probabilities from boosting. In: Proceedings of 21st conference on uncertainty in artificial intelligence (UAI'05), pp 413–420

Platt J (2000) Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In: Smola A, Bartlett P, Schoelkopf B, Schuurmans D (eds) Advances in large margin classifiers. MIT Press, Cambridge, pp 61–74

Rabiner L, Juang BH (1993) Fundamentals of speech recognition. Prentice Hall, Englewood Cliffs

Robertson T, Wright F, Dykstra R (1988) Order restricted statistical inference. Wiley, New York

Schapire RE, Freund Y (2012) Boosting: foundations and algorithms. MIT Press, Cambridge

Schapire RE, Singer Y (1999) Improved boosting algorithms using confidence-rated predictions. Mach Learn 37(3):297–336

Schölkopf B, Platt JC, Shawe-Taylor JC, Smola AJ, Williamson RC (2001) Estimating the support of a high-dimensional distribution. Neural Comput 13(7):1443–1471

Tóth L, Kocsor A, Csirik J (2005) On naive Bayes in speech recognition. Int J Appl Math Compu Sci 15(2):287–294

Tóth S, Sztahó D, Vicsi K (2012) Speech emotion perception by human and machine. In: Proceedings of COST action, pp 213–224, Patras, Greece

van Leeuwen DA, Martin AF, Przybocki MA, Bouten JS (2006) NIST and NFI-TNO evaluations of automatic speaker recognition. Comput Speech Lang 20(2–3):128–158

Waegeman W, Dembczynski K, Jachnik A, Cheng W, Hüllermeier E (2014) On the Bayes-optimality of f-measure maximizers. J Mach Learn Res 15(1):3333–3388

Wu T, Lin C, Weng R (2004) Probability estimates for multi-class classification by pairwise coupling. J Mach Learn Res 5:975–1005

Young S, Evermann G, Gales MJF, Hain T, Kershaw D, Moore G, Odell J, Ollason D, Povey D, Valtchev V, Woodland P (2006) The HTK book. Cambridge University, Cambridge

Zadrozny B, Elkan C (2001) Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. In: Proceedings of ICML, pp 609–616