Discriminative Writer Adaptation

Martin Szummer, Christopher M. Bishop

Discriminative Writer Adaptation

Martin Szummer *Christopher M. Bishop*
Microsoft Research, Cambridge, UK
szummer@microsoft.com, cmbishop@microsoft.com

Abstract

We propose a general method for adapting a writer-independent classifier to an individual writer. We employ a mixture of experts formulation, where the classifiers are trained on weighted clusters of writers. The clusters are determined by which experts classify individual writing correctly. The method adapts by choosing the appropriate combination of classifiers for a new user. It applies to any probabilistic discriminative classifier, and adapts discriminatively without modeling the input feature distribution.

We apply the method to online character recognition. Specifically, we use a mixture of neural networks as well as a mixture of logistic regressions. We train the mixture via conjugate gradient ascent or via the EM algorithm on 192,000 Latin characters of 98 classes and 216 writers, and show adaptation results for 21 writers.

Keywords: writer-adaptation, mixture of experts, neural networks, allographs, personalization

1. Introduction

The large variability of handwriting across individuals makes handwriting recognition a challenging problem. Writer-independent systems trained from examples require large training sets from many writers to deal with this variability. In contrast, writer-dependent systems can be trained on a specific user's writing to achieve the same recognition accuracy with orders of magnitude fewer examples. However, accuracy can be increased further by leveraging both large multi-writer training sets as well as a writer-specific set. In this paper we explore how to *adapt* a writer-independent recognizer to make it writer-dependent. Our recognizer can work immediately without adaptation for a new user, yet adapt gradually as examples of the user's handwriting are accumulated.

From an abstract viewpoint, the adaptation problem can be seen as a learning problem in which the data distribution changes from a generic one modeling data of many users, to a specific one for a new user. (We typically know which user an input comes from.) The question is how aspects of the generic dataset can be exploited when building the adapted classifier. One possibility is to cluster the generic dataset to identify groups of users sharing similar styles, and then to train a classifier for each group. To adapt to a new user, we apply the most appropriate classifier. Another possibility is to train a new classifier for the new user, but incorporating a prior over its parameters learned from the generic dataset.

We perform user-adaptation in the domain of online character recognition. Our focus is on adaptation purely, without involving any language model as often exploited during word recognition. Also we do not consider other types of adaptation, such as adaptation to novel character classes or symbols, adaptation to new fonts, or adaptation of word lexicons. We consider supervised adaptation where labels are available for the adaptation examples of the new user. The approach could be extended to unsupervised adaptation by labeling the examples using the writer-independent classifier [11].

Our strategy is to build classifiers for groups of users sharing similar styles, and then to choose the best combination of classifiers for a new user. More precisely, we learn a mixture of classifiers, where each classifier is trained on data from some weighted subset of the users. Thus we exploit knowledge of which data came from which user. Each classifier can be thought of as an "expert", and our model is an instance of mixture of experts [6]. For a given writer, we choose the experts that classify his writing well. The choice is robustly based on the aggregate confidence across all of the user's characters, and is not specific to individual input characters. The mixture model defines a clustering of users, by grouping users sharing the same experts. Note that this clustering is *discriminative*—we do not model the users' writing input distribution directly, but rather cluster users by classification confidence. Observe also that the clustering and classification is performed jointly, in a single stage. The clustering is probabilistic and assignments can be soft.

The advantage of the discriminative mixture model is that it is optimized for the classification task we ultimately care about. Thus, the model finds clusters useful for classification purposes, not necessarily clusters of similar visual styles which could be irrelevant for classifier accuracy. By avoiding to model the potentially high-dimensional input distribution, the model requires fewer resources and is more parsimonious. The adaptation can be based on very few examples from the new user, as we only need to determine which classifier is most confidently correct for his writing. The architecture is generic and works with any discriminative probabilistic classifiers, such as logistic regression, neural networks, nearest neighbor with probabilistic outputs, Gaussian process classifiers and so on. Classifiers that can take weighted training examples are especially straightforward, and can be trained using the equations provided here.

1.1. Related Work

In the past, clustering of characters into allographs or styles [1, 12] has been done in a separate stage preceding classification. Moreover, typical clustering criteria assume generative models (e.g. mixtures of Gaussians) that do not match the subsequent classification objective, thus the resulting clusters may not aid classification. Unfortunately, the clustering also typically ignores information about which user wrote which examples, so that commonalities within user styles are not modeled. Nevertheless, several adaptation systems do use generative clustering models [5].

Caruana [4] suggests a neural network architecture that learns several related tasks, in which representations learned for each task can benefit learning of the others. In the context of adaptation, one could treat data from each user as a separate classification task, but these tasks could all share related representations in the hidden layer of the neural network. This is a fully discriminative procedure, but to our knowledge, has not been applied to user-adaptation yet.

The most common adaptation method in the literature is to train a new classifier for the new user, while basing it on the multi-writer training set. The multi-writer training set can a) simply be appended to the new writer training set, or b) serve as a prior for the new classifier's parameters, possibly by constraining some of them, or c) it can be used to learn a feature representation. For example, nearest neighbor classifiers are adapted by starting with the writer-independent examples as prototypes and then adding and removing prototypes based on the new user's examples [10].

Hidden Markov models are adapted using strategy b). One trains HMMs for each character on the multi-writer set, and then reestimates a subset of the HMM parameters (e.g. the observation means) from the new user set. Alternatively, the multi-writer set can serve to estimate priors for the HMM to be adapted. When the new writer's dataset is small, one can group some of the HMM parameters together and do a constrained update of the parameters (e.g. a linear transformation of means as in maximum likelihood linear regression) [9, 3, 7].

Neural network models can be updated using strategy c). Guyon et al. [8] reuse the hidden layer representation gained from the multi-writer training as features for a linear SVM classifier for the new user. Unfortunately, all three strategies lump together data from the multi-writer dataset without modeling which user wrote it.

2. Discriminative Adaptation

Assume we have K classifiers, each good at recognizing some styles of handwriting. Some of these classifiers will be applicable to a large number of writers, others may be employed infrequently. Let π_k be the prior probability of employing classifier k, corresponding to the fraction of users whose writing it is good at recognizing. For an input x, this classifier will assign a probability $P(t \mid x, \boldsymbol{\theta}_k)$ to class label t. Here $\boldsymbol{\theta}_k$ are the parameters of the classifier. Let us group input items by user, and denote all inputs from user n by $\boldsymbol{x}_n = \{x_{ni}\}_i$, with i indexing individual examples. For this data the classifier will assign the set of labels $\boldsymbol{t}_n = \{t_{ni}\}_i$ in which the probability $P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}_k) = \prod_i P(t_{ni} \mid x_{ni}, \boldsymbol{\theta}_k)$.

A priori we do not know which classifier we should apply to a user. We can take a weighted average of classifiers to calculate the marginal probability of labels for a user's input, so that

$$P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}_k).$$

This is a mixture of classifiers, where π_k are the mixture weights. Such a mixture is commonly referred to as a mixture of experts [6]. The classifiers are the experts here. Note that this is a mixture per user, so that the probability π_k of choosing a particular classifier is multiplied only once per user, not per input item. For brevity, we have collected the mixture weights and classifier parameters into $\boldsymbol{\theta} = \{\pi_k, \boldsymbol{\theta}_k\}_k$.

The conditional log likelihood of the label set from N users is then

$$L(\boldsymbol{\theta}) = \log \prod_{n=1}^{N} P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}).$$

To train the system, we will optimize the parameters of the classifiers as to maximize this likelihood. We will first describe how to do the maximization via gradient ascent, and secondly show how to apply the EM algorithm. Both methods are introduced for general discriminative classifiers, and later sections 3-4 detail particular instances.

2.1. Gradient Ascent Training

Given a labelled training set, we can maximize the conditional log likelihood $L(\boldsymbol{\theta})$ by an efficient ascent technique such as conjugate gradient ascent. For this we require the derivatives with respect to the parameters π_k and $\boldsymbol{\theta}_k$. However, the priors π_k are non-negative parameters that sum to one. To satisfy automatically these conditions, we reparameterize $\{\pi_k\}$ in terms of $\{\eta_k\}$ as

$$\pi_k = \frac{e^{\eta_k}}{\sum_{j=1}^{K} e^{\eta_j}},$$

and then take derivatives with respect to η_k instead.

We will assume that we already know how to train the individual classifiers by gradient ascent, in other words, we can compute the gradients $\frac{\partial}{\partial \boldsymbol{\theta}_k} \log P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}_k)$. For example, if the classifier is a neural network, we calculate this gradient by backpropagation. Typically, only classifier k will have any contribution to the gradient with respect to parameters $\boldsymbol{\theta}_k$ (although we will discuss a case of parameter-tying across classifiers in section 4). Then the gradient of the likelihood can be expressed in terms of the gradients of the individual classifiers by

$$\frac{\partial}{\partial \boldsymbol{\theta}_k} L(\boldsymbol{\theta}) = \sum_{n=1}^{N} \gamma_{nk} \frac{\partial}{\partial \boldsymbol{\theta}_k} \log P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}_k),$$

where

$$\gamma_{nk} = \frac{\pi_k P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}_k)}{\sum_m \pi_m P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}_m)}. \tag{1}$$

Furthermore, the gradient of the mixture parameters is

$$\frac{\partial}{\partial \eta_k} L(\boldsymbol{\theta}) = \sum_{n=1}^{N} (\gamma_{nk} - \pi_k).$$

The γ_{nk} have a natural interpretation as posterior probabilities that classifier k is responsible for classifying data from user n. Intuitively, if the classifier prior π_k is smaller than the posterior γ_{nk} we increase likelihood by increasing η_k (hence π_k), effectively trying to match prior with posterior.

We shall note that some care is needed to avoid numerical underflow when calculating γ_{nk}. The probabilities $P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}_k)$ are products over multiple items for a user, and frequently underflow unless computed in the log domain, and unless $\max_j \pi_j P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}_j)$ is factored out of the sum.

Finally, we can regularize the problem by adding a prior on the classifier parameters. A Gaussian prior with diagonal covariance $\sigma^2 I$ on the weights subtracts an extra term $\boldsymbol{\theta}_k/\sigma^2$ from the gradient for each k.

2.2. EM Training

The expectation maximization (EM) algorithm [6] is another method for maximizing likelihood. In our case, it alternates between determining which classifiers are applicable to which users, and then maximizing the classifier parameters with respect to those users. It can be employed whenever we are able to maximize the weighted likelihood of the individual classifiers.

To derive the EM algorithm, we note that the likelihood would be simple to optimize if we knew what classifier applied to each user. Hence, we introduce stochastic latent variables z_{nk}, where $z_{nk} = 1$ indicates that classifier k applies to user n, or otherwise $z_{nk} = 0$. A priori, exactly one classifier applies per user, so that $\sum_k z_{nk} = 1$ for all n. A posteriori, the assignment of classifiers is soft, i.e. a combination of classifiers per user.

The distribution over the latent variables is given by the priors as

$$P(\boldsymbol{z}_n|\boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{z_{nk}}$$

where $\boldsymbol{z}_n = \{z_{nk}\}$, and $\boldsymbol{\pi} = \{\pi_k\}$.

The joint log likelihood of the labels and stochastic variables is

$$\begin{aligned}
&\log \prod_n P(\boldsymbol{t}_n, \boldsymbol{z}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}) = \\
&\log \prod_n P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{z}_n, \boldsymbol{\theta}) P(\boldsymbol{z}_n \mid \boldsymbol{\pi}) = \\
&\log \prod_n \prod_k (\pi_k P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}_k))^{z_{nk}} = \\
&\sum_n \sum_k z_{nk} \log(\pi_k P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}_k)).
\end{aligned}$$

In the EM algorithm, we maximize the observed "incomplete" data likelihood $\log \prod_n P(\boldsymbol{t}_n|\boldsymbol{x}_n, \boldsymbol{\theta})$ by iteratively maximizing the expected complete likelihood $E[\log \prod_n P(\boldsymbol{t}_n, \boldsymbol{z}_n|\boldsymbol{x}_n, \boldsymbol{\theta})]$ where the expectation is with respect to $P(\boldsymbol{z}_n|\boldsymbol{t}_n, \boldsymbol{x}_n, \boldsymbol{\theta}^{\text{old}})$, and $\boldsymbol{\theta}^{\text{old}}$ are parameters from the previous iteration. Denote $\gamma_{nk} = E[z_{nk}]$, often referred to as "responsibilities", which turn out to be identical to the posterior probabilities in eq. (1). Then

$$\begin{aligned}
&E[\log \prod_n P(\boldsymbol{t}_n, \boldsymbol{z}_n|\boldsymbol{x}_n, \boldsymbol{\theta})] \\
&= \sum_n \sum_k \gamma_{nk} \log(\pi_k P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}_k)).
\end{aligned}$$

The algorithm now iterates between an M-step and an E-step as follows:

M-step: maximize over the parameters π_k and $\boldsymbol{\theta}_k$, keeping the responsibilities γ_{nk} fixed. The maximization over each classifier's parameters $\boldsymbol{\theta}_k$ can be done independently and reduces to training a single classifier rather than a mixture, with the difference that the data points are weighted by γ_{nk}, so that for classifier k we find

$$\arg\max_{\boldsymbol{\theta}_k} \sum_n \sum_k \gamma_{nk} \log P(\boldsymbol{t}_n|\boldsymbol{x}_n, \boldsymbol{\theta}_k)$$

The maximization over π_k simplifies to

$$\pi_k = \frac{1}{N} \sum_n \gamma_{nk}$$

E-step: calculate the responsibilities γ_{nk} using the parameters from the M-step. The responsibilities are the posterior probabilities of classifier k and user n, given by eq. (1).

The EM algorithm can be started by initializing the parameters of individual classifiers, setting the responsibilities to be uniform $1/K$, and then proceeding with the E-step.

Both the EM algorithm and the gradient ascent method apply to a variety of classifiers. We have described the general case so far, but in sections 3 and 4 we specifically build a mixture of logistic regressors, and a mixture of neural networks.

2.3. Classification for a New User

In the above training procedures, we exploited information about user identity by seeing how well a classifier predicts the labels per user. This allowed the classifiers to specialize on particular users by training on a weighted set of the data. Effectively, the classifiers clustered the writers, according to classifiers' ability to recognize their handwriting.

When a new user enrols in the system, we can begin by classifying his handwriting by taking an average of classifiers, according to the prior weights π_k. In order to adapt to the writer, we must obtain some labeled data from him, possibly by asking him to write a known sample, or by requesting labels for his writing. Given some labeled data, we calculate the posterior probabilities γ_k (eq. (1)) for each classifier and subsequently predict according to $P(t \mid x, \boldsymbol{\theta}) = \sum_k \gamma_k P(t \mid x, \boldsymbol{\theta}_k)$. Note that we do not need to train the classifiers at this point, hence we only require a few bits of information in order to choose the appropriate weighting of classifiers. Thus, we can adapt quickly with a few labeled examples from a new user. If sufficient labeled writer-specific data becomes available, it may be worth retraining the classifiers, but we do not try that here.

If very little labeled data is available for a user, the posterior probabilities are likely to be soft reflecting the uncertainty of limited information. As more data is obtained, the posteriors typically become deterministic 0 or 1, settling on a particular classifier. For efficiency reasons, we may restrict the system to only run the classifier with the highest posterior for the user.

3. Mixture of Logistic Regressors on Synthetic Data

In order to demonstrate the use of conditional mixture models trained by EM in the context of style personalization, we consider a synthetic two-class problem with an easily visualized two-dimensional feature space. The data are points sampled from 8 Gaussians, simulating features of two letters written by four users. The users group into two styles, such that users within a style produce more similar features than users in the other style.

For illustration purposes, we will represent one letter class by solid shapes, and the other class by outline shapes. The two styles are indicated by 3-sided and 4-sided shapes, and examples from an individual user are shown as a specific shape such as a square or a diamond. The classifiers are given the label and the user identity associated with each example, but must discover the styles.

For simplicity we consider a simple logistic regression classifier [2] in which the posterior probability is given by

$$P(t \mid \boldsymbol{x}, \boldsymbol{\theta}) = 1/(1 + e^{-t\boldsymbol{\theta}^T \boldsymbol{x}}),$$

where $\boldsymbol{x}$ is the input vector (in this case the Cartesian coordinates of the data point in the input space), $t = \{+1, -1\}$ is the class label and $\boldsymbol{\theta}$ is the parameter vector. A single lo-

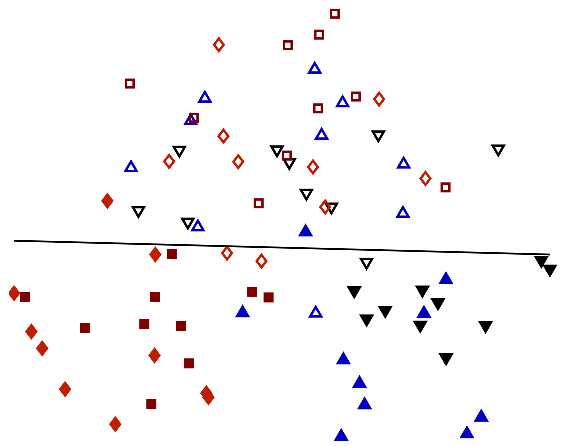
Figure 1. Classification task with two classes "solid" and "outline" (hollow), with examples written by four users (square, diamond, left triangle, right triangle). Two users write in a particular style (shown as 3-sided, i.e. triangles) and the other two share another style (4-sided, i.e. square/diamond). The styles differ for the solid class, so that features for the 3-sided style are to the right of features from the 4-sided. The styles do not differ for the outline class. The decision boundary (line) is from a classifier trained with no adaptation.

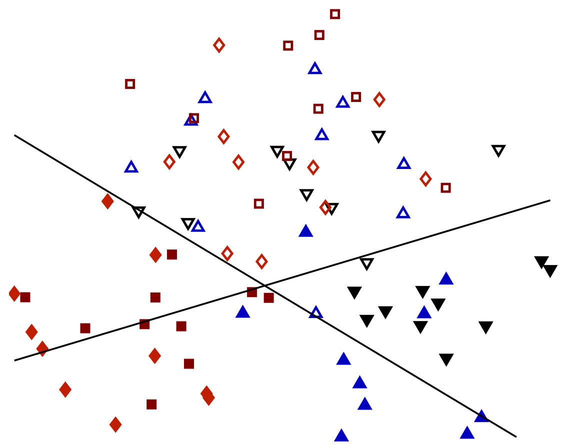
Figure 2. Data as in Figure 1, but with two decision boundaries from a two-classifier mixture trained to automatically discover two styles (3-sided/4-sided style information is not given).

gistic regression model is fitted to the data set, and the decision boundary, corresponding to $P(t = 1 \mid \boldsymbol{x}, \boldsymbol{\theta}) = 0.5$, is also shown in Figure 1.

Now we train a mixture of two logistic regression models on the same data set. The resulting pair of decision boundaries is shown in Figure 2. Note that each component does a better job of separating the two classes for its own style compared to the single model of Figure 1. This is confirmed in Figure 3, where we see data from a new user, and we have shown the decision boundary for

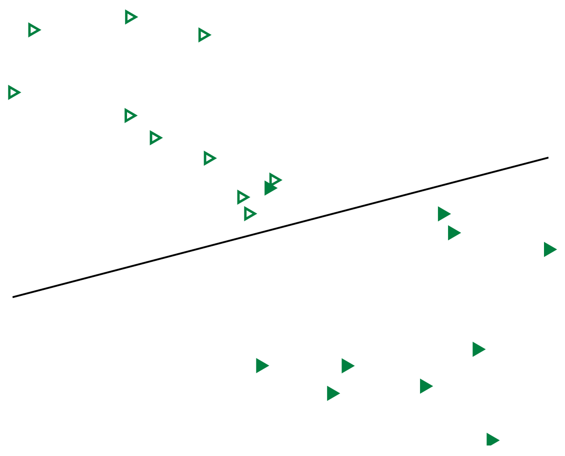

Figure 3. Data from a new user. The classifier for the 3-sided style is most confident at classifying this data and therefore chosen by the system.

their more probable 3-sided (triangle) style.

4. Mixture of Neural Networks on Handwriting Data

Here we apply the adaptation technique to real multiclass data, namely online handwritten characters. We choose two-layer neural networks for the experts. The logistic regression classifiers from the previous section have decision boundaries linear in the features, which may not be sufficiently flexible here.

We train the mixture of neural networks by conjugate gradient ascent. The neural networks employ $\tanh$ nonlinearities at the hidden units, and softmax nonlinearities at the outputs. We will simultaneously optimize all network parameters as well as the mixture probabilities π_k.

The classifier responsibilities typically become hard 0 or 1, so that each network is effectively trained on only a subset of the data. One concern is that the networks may then not have enough training data and overfit as a result. We address the problem by first training a single network, and then initializing the expert networks from its weights plus some random Gaussian noise to break symmetry. Secondly, we stop training the mixture early after a handful of iterations.

As an optional third method of regularization, we can also tie networks together to share the first layer. The first layer can be thought of as a feature extractor, and the same features may be applicable to different classification tasks. This scheme is reminiscent of Rich Caruana's multitask learning [4], where each user can be seen as a different task.

4.1. Experimental Results

We employed a training set of 192,000 single online Latin characters, spanning 98 different classes, including upper-case, lower-case and symbols. These were collected from 216 writers. Network prior parameters were tuned on a validation set of 38,000 characters. Adaptation was performed by estimating the posterior responsibilities on an adaptation training set of 21,000 characters for 21 new users, corresponding to an average of 10 labeled examples per class per user. Finally, the system was tested on an adaptation test set of 59,000 examples for those 21 users.

The dataset had precomputed features, corresponding to upper, lower and side contours, as well as Chebyshev polynomial coefficients, yielding 64 features of roughly zero mean and unit variance. Our networks all employed 150 hidden units and 98 outputs, totalling almost 25,000 weights each.

We put a weak Gaussian prior on the network weights with a standard deviation of 31, and stopped training of the single network after 200 conjugate gradient steps. We then added Gaussian random noise with standard deviation 0.01 to the weights to initialize a mixture of neural networks which we trained for another 5 to 20 iterations on the training set. This step was fairly sensitive to overfitting due to the much larger learning capacity of the mixture. Finally, we estimated the posterior classifier probabilities on the adaptation training set, and tested on the adaptation test set. The training time was about 3 hours for the initial baseline training and 2 hours for the mixture of networks.

Table 1. Classification results. Error(π) denotes the error using the prior classifier probabilities, whereas Error(γ) denotes error using the posterior (adapted) classifier probabilities.

Model	Error (π)	Error (γ)	Bits
Baseline neural net	10.3%	–	–
4 network mixture	10.2%	9.3%	0.12
16 network mixture	10.2%	**9.1**%	0.24

The results in Table 1 show good performance with the baseline neural network of 10.3% error on a 98 class problem. The personalized mixture of 4 networks improves on this result by bringing the error down to 9.3% and the 16-network mixture achieves 9.1%. This reflects a 12% reduction in error from a strong baseline, comparable to results in [5]. Note that the personalization using posterior mixing coefficients is crucial, as by using the prior mixture weights on the network we get the column results given by the Error(π) column, which are very close to the baseline. The final column shows the sharpness of the posterior mixing coefficients, measured as posterior entropy in bits.

Examining the per-user results shows that 16 out of 21 subjects show improved error rate compared to the baseline on the 4-network mixture, and 18 out of 21 subjects show improvement on the 16-network mixture. The most common errors involve confusions among letter 'l', digit '1', and bar '|', as well as uppercase letter 'O', lowercase 'o', digit '0', and degree symbol '°' (Table 2). The baseline and personalized classifiers all have difficulty distinguishing these characters.

Table 2. Most common classification errors for a 16-network mixture, adapted for each of 21 users.

True character	Predicted character	% of total error
| (bar)	l (letter)	7.6%
1 (digit)	l (letter)	4.8%
0 (digit)	O (letter)	4.6%
- (hyphen)	_ (underscore)	2.8%
l (letter)	1 (digit)	2.1%
+	t	1.9%
q	9	1.7%

5. Discussion and Future Work

We have shown how to build a discriminative mixture model whose components are standard classifiers. The mixture can be learned effectively using either gradient ascent or by the EM algorithm. Personalization requires a small amount of labeled data from the new user, and improves performance. The system works "out of the box" without adaptation as well.

In future work, we hope to explore further ways of personalizing the system. For example, the classifier coefficients do not depend on the input the user is writing. It could be the case that the user has a special input-dependent style for some letter that would be better handled by another classifier. We also note that the individual classifiers can be trained further given the labeled data, and improve performance over time given many examples. It would be instructive to examine the clusters of users formed by the algorithm, to see what users assigned to particular classifiers have in common. Finally, an open question is how to optimize the text that the user has to write to personalize the system as to gain an optimal amount of handwriting.

Acknowledgments

I wish to thank Patrick Haluptzok and Michael Revow for the data set, and Tom Minka for discussions on this project.

References

[1] C. Bahlmann and H. Burkhardt, "The writer independent online handwriting recognition system frog on hand and cluster generative statistical dynamic time warping", *IEEE Trans. Pattern Analysis and Mach. Intell. (PAMI)*, 26(3):299–310, Mar 2004.

[2] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford Univ. Press, 1995.

[3] A. Brakensiek, A. Kosmala and G. Rigoll, "Comparing adaptation techniques for on-line handwriting recognition", *Intl. Conf. on Document Analysis and Recognition (ICDAR)*, 2001, pp 486–490.

[4] R. Caruana, "Multitask Learning", *Machine Learning*, 28:41–75, 1997.

[5] S. Connell and A. Jain, "Writer adaptation for online handwriting recognition", *IEEE Trans. Pattern Analysis and Mach. Intell. (PAMI)*, 24(3):329–346, March 2002.

[6] M. I. Jordan and R. A. Jacobs, "Hierarchical Mixtures of Experts and the EM Algorithm", *Neural Computation*, 6:181–214, 1994.

[7] C. Leggetter and P. Woodland, "Maximum likelihood linear regression for speaker adaptation of continuous density HMMs", *Computer Speech and Language*, 9:171–185, 1995.

[8] N. Matić, I. Guyon, J. Denker and V. Vapnik, "Writer-adaptation for On-line Handwritten Character Recognition", *Intl. Conf. on Document Analysis and Recognition (ICDAR)*, 1993, pp 187–191.

[9] J. Subrahmonia, K. Nathan and M. Perrone, "Writer dependent recognition of on-line unconstrained handwriting", *IEEE Intl. Conf. on Acoust. Sp. and Sig. Proc. (ICASSP)*, 1996, volume 6, pp 3478–3481.

[10] V. Vuori, J. Laaksonen, E. Oja and J. Kangas, "On-line adaptation in recognition of handwritten alphanumeric characters", *Intl. Conf. on Document Analysis and Recognition (ICDAR)*, 1999, pp 792–795.

[11] V. Vuori, J. Laaksonen, E. Oja and J. Kangas, "Controlling on-line adaptation of a prototype-based classifier for handwritten characters", *Proc. Intl. Conf. on Pattern Recognition (ICPR)*, 2000, volume 2, pp 331–334.

[12] L. Vuurpijl and L. Schomaker, "Coarse writing-style clustering based on simple stroke-related features", A. Downton and S. Impedovo, editors, *Progress in Handwriting Recognition*, pp 29–34. World Scientific, 1997.