Binary Linear Classification and Feature Selection via Generalized Approximate Message Passing

[illegible]

*Dept. of ECE, The Ohio State University, Columbus, OH 43210. (Email: zinielj@ece.osu.edu, schniter@ece.osu.edu)

†Dept. of Psychology, The Ohio State University, Columbus, OH 43210. (Email: sederberg.1@osu.edu)

[illegible]

I. [illegible]

In this work we consider binary linear classification [illegible] is the signum function and [illegible] is a random perturbation accounting for model inaccuracies. For the purpose of learning $\boldsymbol{w}$, we assume the availability of M labeled training examples generated independently according to this model:

$$[illegible] \quad 1, \ldots, M, \qquad (1)$$

with [illegible] i.i.d. [illegible] It is common to express the relationship between the label [illegible] and the score [illegible] via the conditional pdf [illegible] known as the "activation function," which can be related to the perturbation pdf [illegible] via

$$p_{y_m|z_m}(1|z_m) = \int_{-\infty}^{z_m} p_e(e)\, de = 1 - p_{y_m|z_m}(-1|z_m). \qquad (2)$$

We are particularly interested in classification problems in which the number of potentially discriminatory features N drastically exceeds the number of available training examples M. Such computationally challenging problems are of great interest in a number of modern applications, including text classification [2], multi-voxel pattern analysis [3]–[5], conjoint analysis [6], and microarray gene expression [7]. In the N [illegible] M regime, the model (1) also coincides with that of *noisy one-bit compressed sensing* (CS) [8], [9], where $\boldsymbol{w}$ is the sparse vector of interest and [illegible] is the [illegible] row of the M [illegible] N measurement matrix. [illegible] study to confirm the efficacy, flexibility, and speed afforded by our GAMP-based approaches to binary classification.

II. GAMP [illegible]

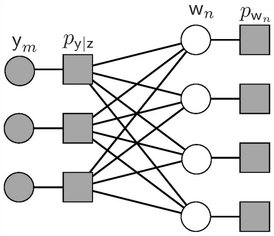
We now formulate the binary linear classifier design problem from both minimum error-rate and maximum a posteriori (MAP) perspectives, and show that each case can be tackled by a variant of the GAMP algorithm.

A. Sum-Product GAMP

Suppose that we are given M labeled training examples [illegible] and [illegible] test feature vectors [illegible] associated with unknown test labels [illegible] all obeying the noisy linear model (1) under some known error pdf [illegible] and thus [illegible] (1),

[illegible]

[illegible]



[illegible]

known [illegible]. We then consider the problem of computing the classification-error-rate minimizing hypotheses [illegible]

$$[illegible] \qquad (3)$$

with [illegible] and [illegible]. Note that we treat the labels [illegible] as random but the features [illegible] as deterministic parameters. The probabilities in (3) can be computed via the marginalization

$$[illegible] \qquad (4)$$

$$[illegible] \qquad (5)$$

with scaling constant [illegible], label vector [illegible] and constraint set [illegible] s.t. [illegible] y and [illegible] $\dots, M\}$ which fixes the tth element of [illegible] at the value [illegible] and the first M elements of [illegible] at the values of the corresponding training labels. The joint pdf in (5) factors as

$$[illegible] \qquad (6)$$

due to the linear model [illegible] and assuming a separable weight-vector prior, i.e., [illegible].

The factorization (6) is illustrated using the *factor graph* in Fig. 1a, which connects the various random variables to the pdf factors in which they appear. Although exact computation of the marginal posterior test-label probabilities via (5) is computationally intractable due to the high-dimensional summation and integration, the factor graph in Fig. 1a suggests the use of loopy belief propagation (LBP), and in particular the *sum-product algorithm* (SPA) [11], as a tractable way to approximate these marginal probabilities. Although the SPA guarantees exact marginal posteriors only under non-loopy (i.e., tree-structured graphs), it has proven successful in many applications with loopy graphs [12].

Because a direct application of the SPA to the factor graph in Fig. 1a is itself computationally infeasible in the high-dimensional case of interest, we turn to a recently developed approximation: the sum-product variant of GAMP [10], as specified in Algorithm [illegible]. There, the expectation and variance in lines 5-6 and 16-17 are taken elementwise w.r.t. the GAMP-approximated marginal posterior pdfs (with superscript k denoting the iteration)

$$[illegible] \qquad (7)$$

$$[illegible] \qquad (8)$$

with appropriate normalizations [illegible] and [illegible] and the vector-vector multiplications and divisions in lines 3, 9, 11, 12, 14, 13, 20 are performed elementwise. Due to space limitations, we refer the interested reader to [10] for an overview and derivation of GAMP, to [13] for rigorous analysis under large i.i.d. sub-Gaussian [illegible] and to [14] for fixed-point analysis under arbitrary [illegible].

[illegible] Generalized Approximate Message Passing

```
[illegible] Matrix [illegible], priors [illegible], activation functions [illegible], and mode [illegible] {SumProduct, MaxSum}
[illegible]
    [illegible]
    [illegible]
    if SumProduct then
        [illegible]
        [illegible] var{[illegible]}
    else if MaxSum then
        [illegible]
        [illegible]
    end if
    [illegible]
    [illegible]
    [illegible]
    [illegible]
    if SumProduct then
        [illegible] E{[illegible]}
        [illegible] var{[illegible]}
    else if MaxSum then
        [illegible]
        [illegible]
    end if
    $k \leftarrow k+1$
until Terminated
```

Applying GAMP to the classification factor graph in Fig. 1a and examining the resulting form of lines 5-6 in Algorithm 1, it becomes evident that the test-label nodes [illegible] do not affect the GAMP weight estimates [illegible] and thus the factor graph can effectively be simplified to the form shown in Fig. 1b, after which the (approximated) posterior test-label pdfs are computed via

$$[illegible] \qquad (9)$$

where $\hat{z}_t^\infty$ and [illegible] denote the [illegible] element of the GAMP vectors

Zk and a 1 respectively, at the final iteration "$k =$ š

@ S Max-Sum GAMP

An alternate approach to linear classifier design is through the minimization of a regularized loss function, e.g.,

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w} \in \mathbb{R}}{\operatorname{argmin}} \sum_{m=1}^{M} f_{z_m}(x \;\; w) + \sum_{n=1}^{LŠ} f_{\,n}(w \quad w \quad n \tag{10}$$

where $f_{z_m}()$ are y_m-dependent convex loss functions (e.g., logistic, probit, or hinge) and where $f_{\,n}$ w () are convex regularization terms (e.g., $f_{\,n}(w \;) =$ AW^2 for £2 regularization and f_w (, w Aiwi for £1 regularization).

The solution to (10) can be recognized as the *maximum a posteriori* (MAP) estimate of random vector $\boldsymbol{w}$ under a separable prior $p_w()$ and likelihood corresponding to (1), i.e.,

$$p_{y|w}(y|w;X) = \prod_{m=1}^{M} p_{y_m|z_m}(y_m|x \;\; w), \tag{11}$$

when $f_z J z) = -\log p_{y_m|z_m}(y_m|z)$ and f_{w} $_n$ (w $-\log p_{w_n}(w$) Importantly, this statistical model is exactly the one yielding the reduced factor graph in Fig. 1b.

Similar to how sum-product LBP can be used to compute (approximate) marginal posteriors in loopy graphs, *max-sum* LBP can be used to compute the MAP estimate [15]. Since max-sum LBP is itself intractable for the high-dimensional problems of interest, we turn to the max-sum variant of GAMP [10], which is also specified in Algorithm There, lines 8-9 are to be interpreted as

$$\hat{z}_m^k = \operatorname{prox}_{\tau_{p_m}^k f_{z_m}}(\hat{p}_m^k), \quad m = 1, \ldots, M, \tag{12}$$

$$\tau_{z_m}^k = \tau_{p_m}^k \operatorname{prox}'_{\tau_{p_m}^k f_{z_m}}(\hat{p}_m^k), \quad m = 1, \ldots, M, \tag{13}$$

with $(\cdot)'$ and $(\cdot)''$ denoting first and second derivatives and

$$\operatorname{prox}_{\tau j}(v) \; f \; \underset{u \in \mathbb{R}}{\operatorname{argmin}} \Big[\; u f + \frac{1}{2\tau} (u - v)^2 \; \Big] \tag{14}$$

$$\operatorname{prox} \; j(v) = (1 + \;\; 1 \, \tau (\operatorname{prox}_{\tau j}(v))) - \backslash \tag{15}$$

and lines 19–20 are to be interpreted similarly. It is known [14] that, for *arbitrary* X, the fixed points of GAMP correspond to the critical points of the optimization objective (10).

III. 1 (; / ; ; ($(= (4 2 q = " q (S q = = " q# ? 4 / > = (4 2 q

One of the hallmarks of GAMP is that its behavior in the large-system limit (i.e., M, N š š with fixed ratio 3 ‡ 1) under i.i.d. sub-Gaussian X is characterized by a scalar state evolution [10], [13]. We now describe how this state evolution can be used to characterize the test-error rate of the linear-classification GAMP algorithms described in Section II.

The GAMP state evolution characterizes average GAMP performance over an ensemble of (infinitely sized) problems, each associated with one realization (y, X, w) of the random triple (yX, $\boldsymbol{w}$). Recall that, for a given problem realization (y, X, w), the GAMP iterations in Algorithm 1 yield the sequence of estimates $\{wk\}$ $_{\mathsf{I}}$ of the true weight vector W. Then, according to the state evolution, $p_{W, Wk}(w, wk)$, $\prod p_{w_n, w}(w_n, w)$ and the first two moments of the joint pdf $p_{w_n, w}$ can be computed using [10, Algorithm 3].

Suppose that the (yX,) above represent training examples associated with a true weight vector $\boldsymbol{w}$, and that $(y, \boldsymbol{x})$ represents a test pair also associated with the same $\boldsymbol{w}$ and with $\boldsymbol{x}$ having i.i.d. elements distributed identically to those of $\boldsymbol{X}$ (with, say, variance). The true and iteration-k-estimated test scores are then $z \doteq \boldsymbol{x}^{\mathsf{T}}\boldsymbol{w}$ and $\hat{z}k \doteq \boldsymbol{x}^{\mathsf{T}}\boldsymbol{w}k$, respectively. The corresponding test-error rate[3] £k f Pr$\{y \neq \operatorname{sgn}(\hat{z}k)\}$ can be computed as follows. Letting & $ denote an indicator function that assumes the value 1 when its Boolean argument is true and the value 0 otherwise, we have

$$£k = \# \{ \mathrm{I} \{ y \# \operatorname{sgn}(\hat{z}k) \} \} \tag{16}$$

$$= \sum_{y \in \{-1,1\}} \cdot \; \mathrm{I} \{ \# \operatorname{sgn}(\hat{z}k) \} \, \Big(p_{y, \hat{z}k, z}(y, zk, z) \, dz \, dzk \tag{17}$$

$$= \sum_{y \in \{-1,1\}} \mathrm{I}\,\mathrm{I}\,\mathrm{I} \{ \# \operatorname{sgn}(\hat{z}k) \} p_{y,z}(y|z) p_{z, \hat{z}k}(z, zk) \, dz \, dzk. \tag{18}$$

Furthermore, from the definitions of ($\hat{z}k$) and the bivariate central limit theorem, we have that

$$+ . \xrightarrow{d} , \mathcal{N} / (\quad 0 = \; , \mathcal{N}(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix} \Sigma = , \Sigma^k) \;_{m}((\;\; \mathrm{BA}) \tag{19}$$,

where $\xrightarrow{d}$ indicates convergence in distribution. In [16], it is shown that the above matrix components are

$$\Sigma_{11}^k = \delta^{-1}\big(\operatorname{var}\{\mathsf{w}_n\} + \mathrm{E}[\mathsf{w}_n]^2\big), \tag{20}$$

$$\Sigma_{12}^k = \Sigma_{21}^k = \delta^{-1}\big(\operatorname{cov}\{\mathsf{w}_n, \widehat{\mathsf{w}}_n^k\} + \mathrm{E}[\mathsf{w}_n]\mathrm{E}[\widehat{\mathsf{w}}_n^k]\big), \tag{21}$$

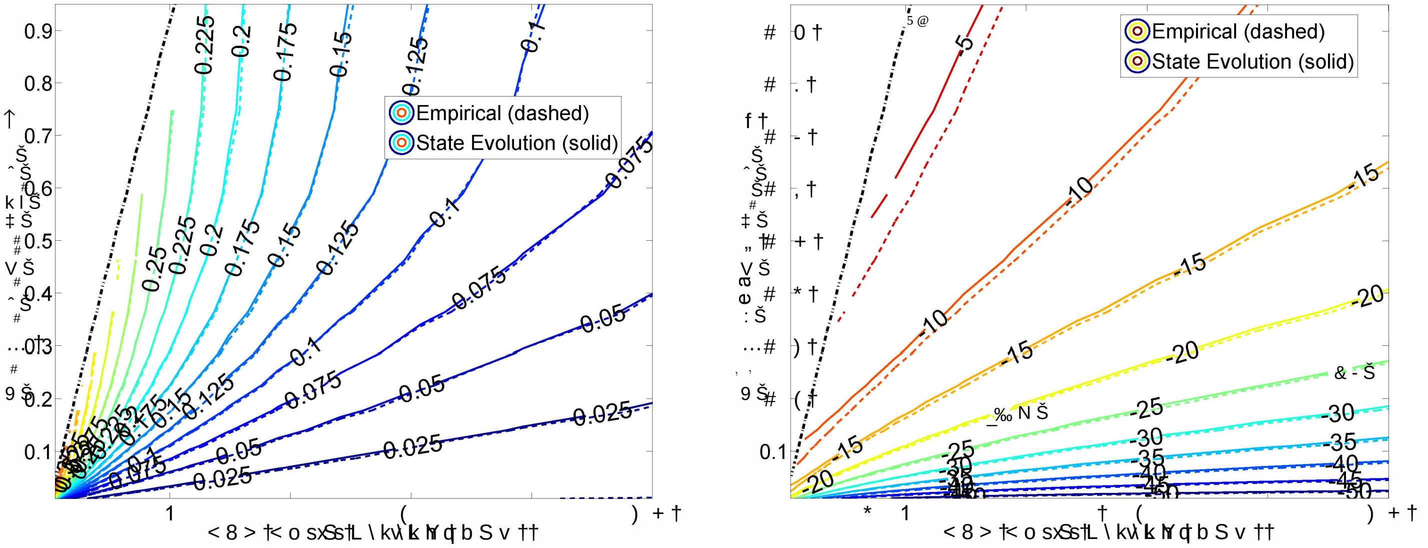
$$\Sigma_{22}^k = \delta^{-1}\big(\operatorname{var}\{\widehat{\mathsf{w}}_n^k\} + \mathrm{E}[\widehat{\mathsf{w}}_n^k]^2\big) \tag{22}$$

for label-to-feature ratio 3 As described earlier, the above moments can be computed using [10, Algorithm 3]. The integral in (18) can then be computed (numerically if needed) for a given activation function $p_{y|z}$, yielding an estimate of GAMP's test-error rate at the k^{th} iteration.

To validate the accuracy of the above asymptotic analysis, we conducted a Monte-Carlo experiment with data synthetically generated in accordance with the assumed model. In particular, for each of 1 0 0 problem realizations, a true weight vector W c š ^L š was drawn i.i.d. zero-mean Bernoulli-Gaussian and a feature matrix X was drawn i.i.d. Gaussian, yielding true scores 8: = Xw, from which the true labels y were randomly drawn using a probit activation function $p_{y|z}$' A GAMP weight-vector estimate $\hat{\boldsymbol{w}}^{\infty}$ was then computed using the training data I:M' X I:M)' from which the test-label estimates $\{y \; f\}$ t HI with $y \; f = \operatorname{sgn}(x \, l \, \hat{\boldsymbol{w}}^{\infty})$ were computed and compared to the true test-labels in order to calculate the test-error rate for that realization. Figure 2a plots the Monte-Carlo averaged empirical test-error rates (dashed) and state-evolution predicted rates (solid) as level curves over different combinations of training ratio 1 and discriminative-feature ratio , where $K = \|W\|_0$ and $N = 1$ 0 4 Similarly, Fig. 2b plots average empirical MSE versus state-evolution predicted MSE, where MSE = kE{‖wo - wi‖D.

---

[3] $ b e f q [' c ^ [L [g k m O q H f f i ' O H N O L [f [b i a ^ q O R g Y O R q e ' } q ` † u X j € ` † H ^ g Y b i X g Y q N e O q L [f [b i a ^ q L f H a J q O H L L b ' ' b N H g [a b q i e H a H ^ m f [f q



[illegible]

In both Fig. 2a and Fig. 2b, the training-to-feature ratio $\frac{M}{N}$ increases from left to right, and the discriminative-feature ratio % increases from bottom to top. The region to the upper-left of the dash-dotted black line contains ill-posed problems (where the number of discriminative features K exceeds the number of training samples M) for which data was not collected. The remainders of Fig. 2a and Fig. 2b show very close agreements between empirical averages and state-evolution predictions.

IV. GAMP [illegible]

We now recall some of the weight-vector priors $V_x(\cdot)$ and activation functions $P_{y|z}(y_m|\cdot)$ (equivalently, regularizations $e_x(\cdot)$ and losses $f_{z_m}(\cdot)$) commonly used for binary linear classification, and we describe how GAMP handles them via the nonlinear steps 5-6, 8-9, 16-17, and 19-20 in Algorithm For sum-product GAMP, we recall that the mean and variance computations in lines 5-6 and 16-17 are computed based on the pdfs in (7) and (8), respectively, and for max-sum GAMP the prox steps in 8-9 are computed using equations (12)-(13) and those in 19-20 are computed similarly.

A. Activation Functions

Arguably the most popular activation function for binary linear classification is the logistic sigmoid [1, §4.3.2]:

$$P_{y|z}(y|z;e) = \frac{1}{1+\exp(-yez)}, \quad Y \in \{-1,1\} \tag{23}$$

where $e > 0$ controls the steepness of the transition. For logistic sum-product GAMP, the mean and variance $(\hat{z}, T_z)$ of the marginal posterior approximation can be computed using a variational approach described in [16]. For logistic max-sum GAMP, $\hat{z}$ from (12) solves the scalar minimization problem (14) with $f(u) = -\log p_{y|z}(y|u;e)$ from (23), which is convex. This $\hat{z}$ can be found via bisection search, by locating the root of $\frac{d}{du}[f(u) + \frac{1}{2}(\cdot)^2]$. The T_z from (13) can then be computed in closed-form using $\hat{z}$ and $f''(\cdot)$ via

Another popular activation function is the probit [1, §4.3.5]:

$$P_{y|z}(1|z;v) = \int N(T;0,V)\,dT = \Phi\left(\frac{z}{\sqrt{v}}\right) \tag{24}$$

where $P_{y|z}(-1|z) = 1 - P_{y|z}(1|z)$ and where $v > 0$ controls the steepness of the sigmoid. In the sum-product case, the density (7) corresponds to the posterior pdf of a random variable z with prior $N(f, T_p)$ from an observation y measured under the model (24), and a derivation in [17, §3.9] provides expressions for the moments used by GAMP in lines 5-6 of Algorithm In the max-sum case, $(\hat{z}, T_z)$ can be computed using bisection search, akin to that described above.

The *hinge loss* $f_z(z) = \max(0, 1 - y_m z)$, commonly used in the *support vector machine* (SVM) approach to maximum-margin classification [1, §7.1], yields the activation function

$$p_{y|z}(y|z) \propto \exp(-\max(0, 1 - yz)), \quad Y \in \{-1, 1\}. \tag{25}$$

For sum-product GAMP, the corresponding $(\hat{z}, T_z)$ from (7) can be computed in closed-form using a procedure described in [16], and for max-sum GAMP, the proximal steps (12)-(13) can be efficiently computed using bisection search.

B. Weight-Vector Priors

For linear classification and feature selection with $N \gg M$, it is customary to choose a prior $V_x(\cdot)$ that leads to sparse (or approximately sparse) weight vectors as discussed below.

In the sum-product case, this is typical accomplished using a Bernoulli-$\tilde{p}$ prior, i.e.,

$$[illegible] \tag{26}$$

where $\delta(\cdot)$ is the Dirac delta function, $\lambda \in [0,1]$ is the prior probability that $w_n \neq 0$, and $\tilde{p}_{w_n}(\cdot)$ is the prior pdf of w_n when non-zero. With GAMP it is common to use Bernoulli-Gaussian [18], Bernoulli-Gaussian-mixture [19], and Bernoulli-Laplacian [16], all of which yield (8) with closed-form first and second moments, $(\hat{w}, \tau_w)$.

In the max-sum case, the GAMP nonlinear outputs $(\hat{w}, \tau_w)$ are computed just as $(\hat{z}, \tau_z)$ are in (12)-(15), but with $-\log p_{w_n}(\cdot)$ used in place of $-\log p_{y|z}(y_m|\cdot)$. Common examples include $f_{w_n}(w) = \lambda|w|_1$ for ℓ_1 regularization [20], $f_{w_n}(w) = \lambda_2 w^2$ for ℓ_2 regularization [10], and $f_{w_n}(w) = \lambda_1|w| + \lambda_2 w^2$ for the "elastic net" [21], all of which yield closed-form implementations of $(\hat{w}, \tau_w)$. (See [16] for details.)

V. AUTOMATIC PARAMETER TUNING

The activation functions and weight-vector priors described in Section IV depend on modeling parameters that, in practice, must be tuned. For example, the logistic sigmoid (23) depends on α, the probit depends on v, ℓ_1 regularization depends on λ, and the Bernoulli-Gaussian-mixture prior depends on λ and $\{\omega_l, \mu_l, \phi_l\}_{l=1}^{L}$, where ω_l parameterizes the weight, μ_l the mean, and ϕ_l the variance of the lth mixture component. Although cross-validation (CV) is the customary approach to tuning parameters such as these, it suffers from two major drawbacks: First, it can be very computationally costly, since each parameter must be tested over a grid of hypothesized values and over multiple data folds. For example, K-fold cross-validation tuning of P parameters using G hypothesized values of each requires the design and evaluation of KG^P classifiers. Second, leaving out a portion of the training data for CV can degrade classification performance, especially in the example-starved regime where $M \ll N$ (see, e.g., [22]).

As an alternative to CV, we consider *online learning* of the unknown model parameters $\boldsymbol{\theta}$ using the method proposed in [19], [23]. Here, the goal is to compute the maximum-likelihood estimate $\hat{\boldsymbol{\theta}}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{y}; \boldsymbol{\theta})$, where our data model implies a likelihood function of the form

$$p(\boldsymbol{y}; \boldsymbol{\theta}) = \int_{\boldsymbol{w}} \prod_m p_{y|z}(y_m | \boldsymbol{x}_m^T \boldsymbol{w}; \boldsymbol{\theta}) \prod_n p_{w_n}(w_n; \boldsymbol{\theta}) \, d\boldsymbol{w}. \quad (27)$$

Because it is computationally infeasible to evaluate and/or maximize (27) directly, approximate-ML estimation is performed using expectation-maximization (EM) [24], treating $\boldsymbol{w}$ as the "hidden" data, which yields the iteration-j estimate

$$\hat{\boldsymbol{\theta}}^{j+1} = \arg\max_{\boldsymbol{\theta}} \mathrm{E}\{\log p(\boldsymbol{y}, \boldsymbol{w}; \boldsymbol{\theta}) \,|\, \boldsymbol{y}; \hat{\boldsymbol{\theta}}^{j}\} \quad (28)$$

$$= \arg\max_{\boldsymbol{\theta}} \sum_m \mathrm{E}\{\log p_{y|z}(y_m | z_m; \boldsymbol{\theta}) \,|\, \boldsymbol{y}; \hat{\boldsymbol{\theta}}^{j}\} + \sum_m \mathrm{E}\{\log p_{w_n}(w_n; \boldsymbol{\theta}) \,|\, \boldsymbol{y}; \hat{\boldsymbol{\theta}}^{j}\}. \quad (29)$$

Furthermore, to evaluate the conditional expectations in (29), GAMP's posterior approximations from (7)-(8) are used. It was shown in [25] that, in the large-system limit, the estimates generated by this procedure are asymptotically consistent (as $j \to \infty$ and under certain identifiability conditions). Moreover, it was shown in [19], [23] that, for various priors and likelihoods of interest in compressive sensing (e.g., AWGN likelihood, Bernoulli-Gaussian-Mixture priors, ℓ_1 regularization), the quantities needed from the expectation in (29) are implicitly computed by GAMP, making this approach computationally attractive. Still, this EM procedure runs GAMP several times, once for each EM iteration (although not necessarily to convergence), thus potentially increasing the runtime over GAMP without EM.

In this work, we export the EM-GAMP methods from [19], [23] to the classification setting, which requires deriving the EM updates for the activation-function parameters, such as α in the logistic model (23) and v in the probit model (24). Due to lack of space, we leave the algorithmic details to [16], but we demonstrate the performance of this approach in Section VI.

VI. NUMERICAL STUDY

In this section we apply GAMP[4] to a text classification problem. This experiment was conducted on a workstation running Red Hat Enterprise Linux (r2.4), with an Intel Core i7-2600 CPU (3.4 GHz, 8MB cache) and 8GB DDR3 RAM. Other experiments can be found in [16], but they are not shown here for reasons of space.

We first consider a binary text classification problem based on the Reuter's Corpus Volume I (RCV1) dataset [26]. As in [27], [28], newswire article topic codes CCAT and ECAT were combined to form the positive class while GCAT and MCAT were combined to form constitute the negative class.[5] Although the original dataset consisted of 20,242 balanced training examples of $N = 47{,}236$ features, with 677,399 examples reserved for testing, we followed the approach in [27], [28] and swapped training and testing sets in order to test computational efficiency on a large training dataset (and thus $M = 677{,}399$). As in [27], we constructed feature vectors as cosine-normalized logarithmic transformations of the TF-IDF (term frequency–inverse document frequency) data vectors. We note that the resulting features are very sparse; only 0.16% of the entries in $\boldsymbol{X}$ are non-zero. Finally, we trained linear classifiers (i.e., weight vectors) using four GAMP-based methods and three existing state-of-the-art methods: TRON [29], CDN [27], and TFOCS [30] in L1-LR mode. For EM learning we used [illegible] EM iterations, and for cross-validation we used 2 folds and a logarithmically spaced grid size of 10.

Table I summarizes the performance achieved by the resulting classifiers, including the test-set classification accuracy, weight-vector density (i.e., the fraction of non-zero weights),[6] and two runtimes: the *total* runtime needed to train the classifier, which includes EM- or cross-validation-based parameter tuning, and the *post-tuning* runtime. Although it is customary to report only the latter, we feel that the former better captures the true computational cost of classifier design.

Table I shows all 7 classifiers achieving nearly identical test-set classification accuracy. However, if feature selection is

---

[4] [illegible]

[5] [illegible]

[6] [illegible]

Classifier	Tuning	Accuracy	Runtime (s)	Density
spGAMP: BG-PR	EM	97.6%	[illegible]	11.1%
spGAMP: BG-HL	EM	[illegible]	468 / 93	[illegible]
msGAMP: L1-LR	EM	97.6%	684 / 123	9.8%
msGAMP: L1-LR	xval	97.6%	3068 / 278	19.6%
CDN	xval	[illegible]	1298 / 112	10.9%
TRON	xval	[illegible]	1682 / 133	10.8%
TFOCS: L1-LR	xval	97.6%	1086 / 94	19.2%

[illegible]

the goal, then weight-vector density is also important, and for this we see significant differences among algorithms. Notably, the two most discriminative (i.e., sparse) classifiers are the result of EM-GAMP methods. We attribute this sparsity in large part to EM-tuning, since Table I shows that when cross-validation is used in place of EM-tuning with L1-LR GAMP, the weight vector is twice as dense.

Table I also shows a wide range of runtimes. Among the post-tuning runtimes, the two fastest are EM-GAMP based. Moreover, among the total runtimes, the three fastest are EM-GAMP based, with the best (at [illegible] seconds) beating the fastest non-GAMP algorithm (at [illegible] seconds) by more than a factor of [illegible]. That said, some caution must be used when comparing runtimes. For example, while all algorithms were given a "stopping tolerance" of 10^{-3}, the algorithms apply this tolerance in different ways. Also, CDN and TRON are implemented in C++, while GAMP is implemented in object-oriented MATLAB (and therefore is far from optimized).

Finally, we note that, although GAMP was derived under the assumption that the elements of X are realizations of an i.i.d. sub-Gaussian distribution, it worked well even with the X of this experiment, which was far from i.i.d. sub-Gaussian. We attribute the robust performance of GAMP to the adaptive damping mechanism included in the GAMPmatlab implementation (described in [31]).

[illegible]

[illegible] *Pattern Recognition and Machine Learning.* [illegible]

[illegible] *J. Mach. Learn. Res.*, [illegible]

[illegible] *Science*, [illegible]

[illegible] *NeuroImage*, [illegible]

[illegible] *NeuroImage*, [illegible]

[illegible] *Conjoint Measurement: Methods and Applications.* [illegible]

[illegible] *Int'l Wkshp. Mach. Learn.*, [illegible]

[illegible] *Proc. Conf. Inform. Science & Sys.*, [illegible]

[illegible] *arXiv:1202.1212*, [illegible]

[illegible] *Proc. IEEE Int'l Symp. Inform. Theory*, [illegible] *arXiv:1010.5141*).

[illegible] *IEEE Trans. Inform. Theory*, [illegible]

[illegible] *Adv. Neural Info. Process. Sys.*, [illegible]

[illegible] *arXiv:1211.5164*, [illegible]

[illegible] *Proc. IEEE Int'l Symp. Inform. Theory*, [illegible] *arXiv:1301.6295*).

[illegible] *IEEE Signal Process. Mag.*, [illegible]

[illegible] *Message Passing Approaches to Compressive Inference Under Structured Signal Priors.* [illegible]

[illegible] *Gaussian Processes for Machine Learning.* [illegible]

[illegible] *Conf. on Information Sciences and Systems (CISS)*, [illegible]

[illegible] *IEEE Trans. Signal Process.*, [illegible]

[illegible] *Proceedings of the National Academy of Sciences*, [illegible]

[illegible] *Roy. Statist. Soc.*, [illegible]

[illegible] *Machine Learning*, [illegible]

[illegible] *Proc. IEEE Workshop Comp. Adv. Multi-Sensor Adaptive Process.*, [illegible] *arXiv:1310.2806*).

[illegible] *Roy. Statist. Soc.*, [illegible]

[illegible] *Proc. Neural Inform. Process. Syst. Conf.*, [illegible] *arXiv:1207.3859*).

[illegible] *J. Mach. Learn. Res.*, [illegible]

[illegible] *J. Mach. Learn. Res.*, [illegible]

[illegible] *Proc. 24th Int'l Conf. Mach. Learn.*, [illegible]

[illegible] *SIAM J. Optim.*, [illegible]

[illegible] *Math. Prog. Comp.*, [illegible]

[illegible] *Allerton Conf. Commun., Control, Comput.*, [illegible]