Binary Classification and Feature Selection via Generalized Approximate Message Passing

Phil Schniter

THE OHIO STATE UNIVERSITY

Collaborators: Justin Ziniel (OSU-ECE) and Per Sederberg (OSU-Psych)

Supported in part by NSF grant CCF-1018368, NSF grant CCF-1218754 and DARPA/ONR grant N66001-10-1-4090.

CISS (Princeton) — March’14
Binary Linear Classification

- Observe \( m \) training examples \( \{(y_i, a_i)\}_{i=1}^m \), each comprised of a binary label \( y_i \in \{-1, 1\} \) and a feature vector \( a_i \in \mathbb{R}^n \).

- Assume that data follows a generalized linear model:
  \[
  \Pr\{y_i = 1 \mid a_i; x_{\text{true}}\} = p_{Y|Z}(1 \mid a_i^T x_{\text{true}}) \]
  \[
  \triangleq z_{i,\text{true}}
  \]
  for some “true” weight vector \( x_{\text{true}} \in \mathbb{R}^n \) and some activation function \( p_{Y|Z}(1 \mid \cdot) : \mathbb{R} \to [0, 1] \).

- **Goal 1:** estimate \( \hat{x}_{\text{train}} \approx x_{\text{true}} \) from training data, so to be able to predict the unknown label \( y_{\text{test}} \) associated with a test vector \( a_{\text{test}} \):
  \[
  \text{compute } \Pr\{y_{\text{test}} = 1 \mid a_{\text{test}}; \hat{x}_{\text{train}}\} = p_{Y|Z}(1 \mid a_{\text{test}}^T \hat{x}_{\text{train}})
  \]
Binary Feature Selection

- Operating regimes:
  - \( m \gg n \): Plenty of training examples: feasible to learn \( \hat{x}_{\text{train}} \approx x_{\text{true}} \).
  - \( m \ll n \): Training-starved: feasible if \( x_{\text{true}} \) is sufficiently sparse!

- The training-starved case motivates...

  **Goal 2:** Identify salient features (i.e., recover support of \( x_{\text{true}} \)).

  - Example: From fMRI, learn which parts of the brain are responsible for discriminating two classes of object (e.g., cats vs. houses):
    \[
    n = 31398 \quad \leftrightarrow \quad \text{fMRI voxels} \\
    m = 216 \quad \leftrightarrow \quad 2 \text{ classes } \times 9 \text{ examples } \times 12 \text{ subjects}
    \]

  - Can interpret as support recovery in noisy one-bit compressed sensing:
    \[
    y = \text{sgn}(Ax_{\text{true}} + w) \text{ with i.i.d noise } w.
    \]
Bring out the GAMP

Zed: Bring out the Gimp.

Maynard: Gimp’s sleeping.

Zed: Well, I guess you’re gonna have to go wake him up now, won’t you?


We propose a new approach to binary linear classification and feature selection based on generalized approximate message passing (GAMP).

Advantages of GAMP include

- flexibility in choosing activation $p_{Y|Z}$ & weight prior $p_X$
- excellent accuracy & runtime
- state-evolution governing behavior in large-system limit
- can tune without cross-validation (via EM extension [Vila & S. ’11])
- can learn & exploit structured sparsity (via turbo extension [S. ’10])
Approximate Message Passing

- AMP is derived from a simplification of message passing (sum-product or max-sum) that holds in the large-system limit.

- AMP manifests as a sophisticated form of iterative thresholding, requiring only two applications of $A$ per iteration and few iterations.

- The evolution of AMP:
  - The original AMP [Donoho, Maleki, Montanari '09] solved the LASSO problem $\arg \min_x \|y - Ax\|_2^2 + \lambda \|x\|_1$ assuming i.i.d sub-Gaussian $A$.
  - The Bayesian AMP [Donoho, Maleki, Montanari '10] extended to MMSE inference in AWGN for any factorizable signal prior $\prod_j p_X(x_j)$.
  - The generalized AMP [Rangan '10] framework extends to MAP or MMSE inference under any factorizable signal prior & likelihood.
GAMP Theory

- In the **large-system limit with i.i.d sub-Gaussian** $A$, GAMP follows a state-evolution trajectory whose fixed points are **MAP/MMSE optimal solutions** when unique [Rangan '10], [Javanmard, Montanari '12]

- **With arbitrary finite-dimensional** $A$,
  - the fixed-points of **max-product GAMP** coincide with the critical points of the MAP optimization objective

  $$\arg \max_x \left\{ \sum_{i=1}^{m} \log p_{Y_i|Z_i}(y_i|[Ax]_i) + \sum_{j=1}^{n} \log p_{X_j}(x_j) \right\}$$

  - the fixed-points of **sum-product GAMP** coincide with the critical points of a certain free-energy optimization objective [Rangan, S., et al'13] and **damping** can be used to ensure that GAMP converges to its fixed points. [Rangan,S.,Fletcher’14]
So, how do we use GAMP to design the weight vector \( \hat{x} \)?

1. Choose GAMP’s linear transform \( A \):
   - **Linear classification**: the rows of \( A \) are the feature vectors \( \{a_i^T\}_{i=1}^n \).
   - **Kernel-based classification**: \([A]_{i,j} = \mathcal{K}(a_i, a_j)\) with appropriate \( \mathcal{K}(\cdot, \cdot) \).

2. Choose inference mode:
   - **max-sum**: finds \( \hat{x} \) that minimizes regularized loss, i.e.,
     \[
     \hat{x} = \arg\min_x \left\{ \sum_{i=1}^m f([Ax]_i; y_i) + \sum_{j=1}^n g(x_j) \right\}
     \]
     for chosen \( f \) and \( g \)
   - **sum-product**: computes the marginal weight posteriors \( p_{X|Y}(\cdot|y) \) under the assumed statistical model:
     \[
     \Pr\{y|A, x\} = \prod_{i=1}^m p_{Y|Z}(y_i|a_i^T x) \quad \text{and} \quad p(x) = \prod_{j=1}^n p_X(x_j).
     \]

3. Choose activation fxn \( p_{Y|Z}(y_i|\cdot) \propto e^{-f(\cdot; y_i)} \) and prior \( p_X(\cdot) \propto e^{-g(\cdot)} \)
GAMPmatlab: Implemented Activations and Priors

For given $p_{Y|Z}$ and $p_X$, GAMP needs to compute the mean and variance (sum-product), or the max and sensitivity (max-sum), of the scalar pdfs:

\[
p_{Z|Y}(z|y; \mu_Q, v_Q) \propto p_{Y|Z}(y|z)\mathcal{N}(z; \mu_Q, v_Q)
\]
\[
p_{X|Q}(x|q; \mu_R, v_R) \propto p_X(x)\mathcal{N}(x; \mu_R, v_R)
\]

Our [http://sourceforge.net/projects/gampmatlab/](http://sourceforge.net/projects/gampmatlab/) implementation handles these computations for various common choices of $p_{Y|Z}$ and $p_X$:

| activation | $p_{Y|Z}$ | sum-prod | max-prod |
|------------|-----------|----------|----------|
| logit      | VI        |          | RF       |
| probit     | CF        |          | RF       |
| hinge      | CF        |          | RF       |
| robust-*   | CF        |          | RF       |

<table>
<thead>
<tr>
<th>prior</th>
<th>$p_X$</th>
<th>sum-prod</th>
<th>max-prod</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gaussian</td>
<td>CF</td>
<td>CF</td>
</tr>
<tr>
<td></td>
<td>Laplace</td>
<td>CF</td>
<td>CF</td>
</tr>
<tr>
<td></td>
<td>Elastic Net</td>
<td>CF</td>
<td>CF</td>
</tr>
<tr>
<td></td>
<td>Bernoulli-*</td>
<td>CF</td>
<td>–</td>
</tr>
</tbody>
</table>

CF=closed-form, NI=numerical integration, VI=variational inference, RF=root-finding.
Beyond GAMP: The EM & turbo Extensions

The basic GAMP algorithm requires

1. separable priors \( p(y|z) = \prod_i p_{Y_i|Z_i}(y_i|z_i) \) and \( p(x) = \prod_j p_{X_j}(x_j) \)
2. that are perfectly known.

The EM-turbo-GAMP framework circumvents these limitations by learning possibly non-separable priors:

![Diagram showing the EM-turbo-GAMP framework with local and global parameter sets.]

- EM
- turbo
- GAMP
- local \( \{p_{Y_i|Z_i}(y_i|z_i)\}_{i} \)
- global \( p(y|z; \theta_{Y|Z}) \)
- parameters \( \theta_{Y|Z} \)
- linear transform \( A \)
- local \( \{p_{X_j}(x_j)\}_{j} \)
- global \( p(x; \theta_{X}) \)
- parameters \( \theta_{X} \)
Recall that GAMP obeys a state evolution that characterizes the quality of $\hat{x}$ at each iteration $t$ (with i.i.d sub-Gaussian $A$ in the large-system limit).

We can use this to predict the classification test-error rate.

For example, with $A \sim \text{i.i.d } \mathcal{N}(0, 1)$, $p_X \text{ Bernoulli-Gaussian}$, $p_{Y|Z} \text{ probit}$, we get...

Notice close agreement between SE (solid) and empirical (dashed).
Robust Classification

- Some training sets contain corrupted labels (e.g., randomly flipped).
- In response, one can “robustify” any given activation fn $p_{Y|Z}$ via
  $$
  \tilde{p}_{Y|Z}(y|z) = (1 - \varepsilon)p_{Y|Z}(y|z) + \varepsilon p_{Y|Z}(1 - y|z),
  $$
  where $\varepsilon \in [0, 1]$ models the flip probability.

- The example shows test-error rate for standard (upper) and robust (lower) activation fns with genie-tuned and EM-tuned $\varepsilon$:

- Details: $x_j \sim \text{iid } \mathcal{N}(0, 1)$,
  $a_i^T|\{y_i = \pm 1\} \sim \text{iid } \mathcal{N}(\pm \mu, I)$,
  $\varepsilon$-flipped logistic $p_{Y|Z}$,
  $m = 8192$, $n = 512$.  

(Bayes error rate = 0.05)
Text Classification Example

- Reuter’s Corpus Volume 1 (RCV1) dataset
- \( n = 47,236 \) features, \( m = 677,399 \) training examples
- 0.0016-sparse features (far from i.i.d sub-Gaussian \( \mathbf{A}! \))

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Tuning</th>
<th>Accuracy</th>
<th>Runtime (s)</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>spGAMP: BG-PR EM</td>
<td></td>
<td>97.6%</td>
<td>317 / 57</td>
<td>11.1%</td>
</tr>
<tr>
<td>spGAMP: BG-HL EM</td>
<td></td>
<td>97.7%</td>
<td>468 / 93</td>
<td>8.0%</td>
</tr>
<tr>
<td>msGAMP: L1-LR EM</td>
<td></td>
<td>97.6%</td>
<td>684 / 123</td>
<td>9.8%</td>
</tr>
<tr>
<td>CDN xval</td>
<td></td>
<td>97.7%</td>
<td>1298 / 112</td>
<td>10.9%</td>
</tr>
<tr>
<td>TRON xval</td>
<td></td>
<td>97.7%</td>
<td>1682 / 133</td>
<td>10.8%</td>
</tr>
<tr>
<td>TFOCS: L1-LR xval</td>
<td></td>
<td>97.6%</td>
<td>1086 / 94</td>
<td>19.2%</td>
</tr>
</tbody>
</table>

⇒ EM-GAMP yields fast, accurate, and sparse classifiers.
Haxby Example

- We now return to the problem of learning, from fMRI measurements, which parts of the brain are responsible for discriminating two classes of object.

- Note that the main problem here is feature selection, not classification. The observed classification error rate is used only to judge the validity of the support estimate.

- For this we use the famous Haxby data, with

  \[ n = 31398 \leftrightarrow \text{fMRI voxels} \]
  \[ m = 216 \leftrightarrow 2 \text{classes} \times 9 \text{examples} \times 12 \text{subjects} \]

### Haxby: Cats vs. Houses

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Setup</th>
<th>Error Rate</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>EM-GAMP</td>
<td>sum-prod logit/B-Laplace</td>
<td>0.9%</td>
<td>9 sec</td>
</tr>
<tr>
<td>EM-GAMP</td>
<td>sum-prod probit/B-Laplace</td>
<td>1.9%</td>
<td>13 sec</td>
</tr>
<tr>
<td>EM-turbo-GAMP</td>
<td>sum-prod probit/B-Laplace 3D-MRF</td>
<td>2.8%</td>
<td>14 sec</td>
</tr>
</tbody>
</table>

#### Without 3D MRF

![Haxby Classification: Houses vs. Cats | GAMP: i.i.d. Bernoulli-Laplacian + Probit](image1.png)

#### With 3D MRF

![Haxby Classification: Houses vs. Cats | GAMP: 3D MRF + Bernoulli-Laplacian + Probit](image2.png)
Conclusions

- We presented a novel application of GAMP to binary linear classification & feature selection.

- Some nice properties of classification-GAMP include:
  - flexibility in choice of activation function and weight prior
  - runtime (e.g., 3-4× faster than recent methods)
  - state-evolution can be used to predict test error-rate
  - can handle corrupted labels (via robust prior)
  - can tune without cross-validation (via EM extension)
  - can exploit and learn structured sparsity (via turbo extension)

- All of the above also applies to one-bit compressive sensing.
All these methods are integrated into GAMPmatlab:
http://sourceforge.net/projects/gampmatlab/

Thanks!
GAMP Heuristics (Sum-Product)

1. Message from $y_i$ node to $x_j$ node:

$$p_{i \rightarrow j}(x_j) \propto \int_{\{x_r\}_{r \neq j}} p_{Y|Z}(y_i; \sum_r a_{ir} x_r) \prod_{r \neq j} p_{i \leftarrow r}(x_r)$$

$$\approx \int_{z_i} p_{Y|Z}(y_i; z_i) N(z_i; \hat{z}_i(x_j), \nu_i^z(x_j)) \approx N$$

To compute $\hat{z}_i(x_j), \nu_i^z(x_j)$, the means and variances of $\{p_{i \leftarrow r}\}_{r \neq j}$ suffice, thus Gaussian message passing!

Remaining problem: we have $2mn$ messages to compute (too many!).

2. Exploiting similarity among the messages $\{p_{i \leftarrow j}\}_{i=1}^m$, GAMP employs a Taylor-series approximation of their difference, whose error vanishes as $m \rightarrow \infty$ for dense $A$ (and similar for $\{p_{i \rightarrow j}\}_{j=1}^n$ as $n \rightarrow \infty$). Finally, need to compute only $O(m+n)$ messages!
The GAMP Algorithm

**Require:** Matrix $A$, sum-prod $\in \{\text{true}, \text{false}\}$, initializations $\hat{x}^0, \nu_x^0$

$t = 0, \hat{s}^{-1} = 0, \forall ij: S_{ij} = |A_{ij}|^2$

repeat

\[
\nu_p^t = S\nu_x^t, \quad \hat{p}^t = A\hat{x}^t - \hat{s}^{t-1}\nu_p^t \quad \text{(gradient step)}
\]

if sum-prod then

\[
\forall i : \nu_{z_i}^t = \text{var}(Z|P; \hat{p}_i^t, \nu_{p_i}^t), \quad \hat{z}_i^t = \text{E}(Z|P; \hat{p}_i^t, \nu_{p_i}^t),
\]

else

\[
\forall i : \nu_{z_i}^t = \nu_{p_i}^t \text{ prox}'_{\nu_{p_i}^t} \log p_Y|Z(y_i,.)(\hat{p}_i^t) \quad \hat{z}_i^t = \text{prox}_{-\nu_{p_i}^t} \log p_Y|Z(y_i,.)(\hat{p}_i^t),
\]

end if

\[
\nu_s^t = \frac{(1 - \nu_{z_i}^t./\nu_p^t)./\nu_p^t, \quad \hat{s}^t = (\hat{z}^t - \hat{p}^t)./\nu_p^t \quad \text{(dual update)}
\]

\[
\nu_r^t = 1.\/(S^T\nu_s^t), \quad \hat{r}^t = \hat{x}^t + \nu_r^t.A^T\hat{s}^t \quad \text{(gradient step)}
\]

if sum-prod then

\[
\forall j : \nu_{x_j}^{t+1} = \text{var}(X|R; \hat{r}_j^t, \nu_{r_j}^t), \quad \hat{x}_j^{t+1} = \text{E}(X|R; \hat{r}_j^t, \nu_{r_j}^t),
\]

else

\[
\forall j : \nu_{x_j}^{t+1} = \nu_{r_j}^t \text{ prox}'_{\nu_{r_j}^t} \log p_X(.)(\hat{r}_j^t) \quad \hat{x}_j^{t+1} = \text{prox}_{-\nu_{r_j}^t} \log p_X(.)(\hat{r}_j^t),
\]

end if

\[t \leftarrow t + 1\]

until Terminated

Note connections to Arrow-Hurwicz, primal-dual, ADMM, proximal FB splitting, ...