Large-Scale Sparse Bayesian Linear Regression

Ralf Herbrich

May 2, 2014

Abstract

In this short note, we derive an update algorithm for the sparse Bayesian linear regression using factor graph inference. We will also show how this algorithm computes the best factorising approximation using Expectation Propagation and MapReduce. This algorithm is particularly effective for highly sparse and high-dimensional linear models because the runtime and memory complexity is $O(ND)$ and $O(K)$ instead of $O\left(K^3\right)$ and $O\left(K^2\right)$, respectively, for the standard non-sparse Bayesian Linear Regression inference algorithms, where N is the number of training examples, D is the average sparsity of each training examples and K is the dimensionality of the parameter/weight vector. Since we are using a Bayesian framework, we can reliably deal with $K = O(N)$. Also, the proposed algorithm is very applicable when the dataset is split over multiple files on a Map-Reduce system.

1 Introduction

Notation In this rest of the note, we use the following notation: Lower-case bold symbols denote column vectors, e.g. $\mathbf{x}$, while upper-case bold symbols denote matrices, e.g. $\mathbf{A}$. The transpose of a vector is denoted by $^{\mathrm{T}}$, for example, $\mathbf{x}^{\mathrm{T}}$. We will use two notations for the Gaussian density: $\mathcal{N}\left(x;\mu,\sigma^2\right)$ and $\mathcal{G}\left(x;\tau,\pi\right)$ which are related to each other via

$$\tau = \mu\sigma^{-2} \qquad \text{and} \qquad \pi = \sigma^{-2}\,, \tag{1}$$

$$\mu = \tau\pi^{-1} \qquad \text{and} \qquad \sigma^2 = \pi^{-1}\,. \tag{2}$$

The (more standard) notation $\mathcal{N}$ is useful when studying the location and spread parameter of the Gaussian density while the canonical parameterisation $\mathcal{G}$ is more useful for product and division because

$$\mathcal{G}\left(x;\tau_1,\pi_1\right)\cdot\mathcal{G}\left(x;\tau_2,\pi_2\right) \propto \mathcal{G}\left(x;\tau_1+\tau_2,\pi_1+\pi_2\right)\,. \tag{3}$$

2 Problem Setting

In the following, we will study the following setting: All our data is given as a (distributed) set of examples $\{(\mathbf{x}_i,y_i)\}_{i=1}^N$ where $(\mathbf{x}_i,y_i)\in\mathbb{R}^K\times\mathbb{R}$. We will also assume that $N > 10^9$ and $K > 10^6$ and the entire dataset is stored on a distributed set of files. Moreover, we shall also assume that any linear model we learn cannot be stored in a single RAM or file and will be partitioned over distributed storage systems (e.g., files). Finally, we make the *crucial assumption that $\|\mathbf{x}_i\|_0 < D \ll K$, that is, all feature vectors $\mathbf{x}_i$ have almost all components set to zero (in practise, we can assume that $D \approx 10^2$).*

Our data model is that of a linear model with zero-mean Gaussian noise of input-dependent variance, that is,

$$p\left(y|\mathbf{x},\mathbf{w}\right) = \mathcal{N}\left(y;\mathbf{x}^{\mathrm{T}}\mathbf{w},\beta_{\mathbf{x}}^2\right)\,.$$

Note that the sparsity assumption means that the inner product $\mathbf{x}^{\mathrm{T}}\mathbf{w}$ can be carried out with D lookup and product/sum operations. We shall denote by V_i the set of non-zero indices of $\mathbf{x}_i$

and note that $|V_i| \leq D$. Finally, we assume that we will constantly maintain a fully factorising Gaussian belief distribution $p(\mathbf{w})$, that is,

$$p(\mathbf{w}) = \prod_{j=1}^{K} \mathcal{N}\left(w_j; \mu_j, \sigma_j^2\right) .$$

2.1 Predictive Distribution

Given these assumptions, it is easy to work out the predictive distribution at a new example $\mathbf{x}$ which is simply given by

$$\begin{aligned} p(y|\mathbf{x}) &= \int p(y|\mathbf{x}, \mathbf{w}) \cdot p(\mathbf{w}) \; d\mathbf{w} \\ &= \int \mathcal{N}\left(y; \mathbf{x}^{\mathrm{T}}\mathbf{w}, \beta_{\mathbf{x}}^2\right) \cdot \prod_{j=1}^{K} \mathcal{N}\left(w_j; \mu_j, \sigma_j^2\right) \; d\mathbf{w} \\ &= \mathcal{N}\left(y; \mathbf{x}^{\mathrm{T}}\boldsymbol{\mu}, \sum_j x_j^2\sigma_j^2 + \beta_{\mathbf{x}}^2\right) . \end{aligned}$$

Note again that due to the assumed sparsity assumption, the inner products are reduced to D lookups for the non-zero dimensions of $\mathbf{x}$!

2.2 Single Point Inference

In order to derive the single-point inference algorithm, we will be using a factor graph representation that experesses the computation in terms of operations over messages from all factors in the graph corresponding to

$$p(y, s, \mathbf{w}|\mathbf{x}) = \underbrace{p(y|s)}_{h(s)} \cdot \underbrace{p(s|\mathbf{x}, \mathbf{w})}_{g(s, w_1, \ldots, w_K)} \cdot \underbrace{p(\mathbf{w})}_{\prod_{j=1}^{K} f_j(w_j)} ,$$

where the three factors are defined by

- Factor f_j: Gaussian prior $\mathcal{N}\left(w_j; \mu_j, \sigma_j^2\right)$
- Factor g: Inner product between $\mathbf{x}$ and $\mathbf{w}$, that is, $\delta\left(s = \mathbf{x}^{\mathrm{T}}\mathbf{w}\right)$
- Factor h: The data likelihood $\mathcal{N}\left(y; s, \beta_{\mathbf{x}}^2\right)$

Looking at the factor graph, we see that the posterior over the weights can be written as follows:

$$\begin{aligned} p(w_j|\mathbf{x}, y) &= m_{g \to w_j}(w_j) \cdot m_{f_j \to w_j}(w_j) \\ &= m_{g \to w_j}(w_j) \cdot \mathcal{N}\left(w_j; \mu_j, \sigma_j^2\right) \\ &= \mathcal{N}\left(w_j; \tilde{\mu}_j, \tilde{\sigma}_j^2\right) . \end{aligned}$$

In order to compute $m_{g \to w_j}$ we note that $m_{s \to g}(s) = m_{h \to s}(s) = \mathcal{N}\left(s; y, \beta_{\mathbf{x}}^2\right)$ and $m_{w_j \to g}(w_j) = m_{f_j \to w_j}(w_j) = \mathcal{N}\left(w_j; \mu_j, \sigma_j^2\right)$. Using the message update equation for the summation factor from the TrueSkill paper, we get

$$m_{g \to w_j}(w_j) = \mathcal{N}\left(w_j; -\frac{1}{x_j}\left(\sum_{k=1}^{K} x_k\mu_k - x_j\mu_j - y\right), \frac{1}{x_j^2}\left(\sum_{k=1}^{K} x_k^2\sigma_k^2 - x_j^2\sigma_j^2 + \beta_{\mathbf{x}}^2\right)\right) .$$

Note again that due to the sparsity, the inner products reduce to D lookups for the non-zero dimensions of $\mathbf{x}$. Also, in practise we can transform the Gaussian into the $\mathcal{G}$ parameterisation using (1) in two-divisions and then carry out the product above using (3).

Also note that we can express the mean and variance update equations in a closed-form by carrying out these steps in terms of $\mathbf{x}, y$ and the prior paramters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$. More formally, we have

$$\begin{aligned}
\tilde{\sigma}_j^2 &= \left(\pi_{g\to w_j} + \pi_{f_j\to w_j}\right)^{-1} \\
&= \left(\frac{x_j^2}{\sum_{k=1}^K x_k^2\sigma_k^2 - x_j^2\sigma_j^2 + \beta_{\mathbf{x}}^2} + \frac{1}{\sigma_j^2}\right)^{-1} \\
&= \left(\frac{x_j^2\sigma_j^2 + \sum_{k=1}^K x_k^2\sigma_k^2 - x_j^2\sigma_j^2 + \beta_{\mathbf{x}}^2}{\sigma_j^2\left[\sum_{k=1}^K x_k^2\sigma_k^2 - x_j^2\sigma_j^2 + \beta_{\mathbf{x}}^2\right]}\right)^{-1} \\
&= \sigma_j^2 \cdot \left(\frac{\sum_{k=1}^K x_k^2\sigma_k^2 - x_j^2\sigma_j^2 + \beta_{\mathbf{x}}^2}{\sum_{k=1}^K x_k^2\sigma_k^2 + \beta_{\mathbf{x}}^2}\right).
\end{aligned}$$

$$\boxed{\tilde{\sigma}_j^2 = \sigma_j^2 \cdot \left(1 - \frac{x_j^2\sigma_j^2}{\sum_{k=1}^K x_k^2\sigma_k^2 + \beta_{\mathbf{x}}^2}\right)} \qquad (4)$$

and, for the mean $\tilde{\mu}_j$

$$\begin{aligned}
\tilde{\mu}_j &= \frac{\tau_{g\to w_j} + \tau_{f_j\to w_j}}{\pi_{g\to w_j} + \pi_{f_j\to w_j}} \\
&= \left(-x_j \cdot \frac{\mathbf{x}^{\mathrm{T}}\boldsymbol{\mu} - x_j\mu_j - y}{\sum_{k=1}^K x_k^2\sigma_k^2 - x_j^2\sigma_j^2 + \beta_{\mathbf{x}}^2} + \frac{\mu_j}{\sigma_j^2}\right) \cdot \frac{\sigma_j^2\left(\sum_{k=1}^K x_k^2\sigma_k^2 - x_j^2\sigma_j^2 + \beta_{\mathbf{x}}^2\right)}{\sum_{k=1}^K x_k^2\sigma_k^2 + \beta_{\mathbf{x}}^2} \\
&= \frac{-x_j\sigma_j^2 \cdot \left[\mathbf{x}^{\mathrm{T}}\boldsymbol{\mu} - x_j\mu_j - y\right] + \mu_j \cdot \left[\sum_{k=1}^K x_k^2\sigma_k^2 - x_j^2\sigma_j^2 + \beta_{\mathbf{x}}^2\right]}{\sum_{k=1}^K x_k^2\sigma_k^2 + \beta_{\mathbf{x}}^2} \\
&= \frac{\mu_j \cdot \left[\sum_{k=1}^K x_k^2\sigma_k^2 + \beta_{\mathbf{x}}^2\right] - x_j\sigma_j^2 \cdot \left[\mathbf{x}^{\mathrm{T}}\boldsymbol{\mu} - y\right]}{\sum_{k=1}^K x_k^2\sigma_k^2 + \beta_{\mathbf{x}}^2}.
\end{aligned}$$

$$\boxed{\tilde{\mu}_j = \mu_j + \frac{x_j\sigma_j^2}{\sum_{k=1}^K x_k^2\sigma_k^2 + \beta_{\mathbf{x}}^2} \cdot \left(y - \mathbf{x}^{\mathrm{T}}\boldsymbol{\mu}\right)} \qquad (5)$$

2.3 Expectation Propagation

The above algorithm in terms of update equations (5) and (4) provides the best factorizing approximation to the posterior $p\left(\mathbf{w}|\mathbf{x}, y\right)$ on a single example but does not give the optimal approximation to a set of observations $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$. In order to get a good approximation for this case, we make the following approximation:

$$p\left(\mathbf{w}|\left\{\mathbf{x}_i, y_i\right\}\right) \approx q\left(\mathbf{w}\right) = \prod_{i=1}^N \underbrace{\prod_{j\in V_i} \mathcal{N}\left(w_j; \mu_{ij}, \sigma_{ij}^2\right)}_{\approx p(y_i|\mathbf{x}_i,\mathbf{w})} \cdot \underbrace{\prod_{j=1}^K \mathcal{N}\left(w_j; \mu_j, \sigma_j^2\right)}_{p(\mathbf{w})}.$$

The approximating Gaussians $\mathcal{N}\left(w_j; \mu_{ij}, \sigma_{ij}^2\right)$ are called *messages* from factor $p\left(y_i|\mathbf{x}_i, \mathbf{w}\right)$ to variable w_j. Note that due to the sparsity, we only have $D \times N$ many messages rather than $O\left(K^2\right)$ many elements that we would need to represent the full covariance matrix of all weights spanned by the N examples. The Expectation Propagation (EP) algorithm proceeded by continually updating the approximation for $p\left(y_i|\mathbf{x}_i, \mathbf{w}\right)$ as follows:

1. **Initialize** $m_{ij} = \mathcal{G}(w_j; 0, 0)$ for all $N \times D$ messages.
2. **Initialize** the current marginals $q(w_j) = \mathcal{N}\left(w_j; \mu_j, \sigma_j^2\right)$.
3. **Do**
   (a) Set $\delta = 0$.
   (b) **For** all i in $1, \ldots, N$
      i. For all $j \in V_i$ compute $p(w_j) = q(w_j) / m_{ij}(w_j)$ using (1), (2), and (3).
      ii. Compute the new $q_{\text{new}}(w_j)$ using (4) and (5) using the μ_j and σ_j^2 from step i.
      iii. Update $m_{ij}(w_j) \leftarrow q_{\text{new}}(w_j) / p(w_j)$ using (1), (2), and (3).
      iv. Update $\delta \leftarrow \max\left(\delta, \text{KL}\left(q_{\text{new}}(w_j), q(w_j)\right)\right)$ for all $j \in V_i$.
      v. Update the marginals by $q \leftarrow q_{\text{new}}$.
4. **Until** $\delta < \varepsilon$
5. **Return** q

This algorithm computes the best possible factorising approximation of $p(\mathbf{w})$ by incrementally improving the approximation of each data likelihood term $p(y_i|\mathbf{x}_i, \mathbf{w})$. Since there are no other approximations than the factorization of the posterior, the mean $\boldsymbol{\mu}$ of the posterior is identical to the mean of the non-sparse posterior

$$\left(\boldsymbol{\Sigma}_0^{-1} + \text{diag}\left(\boldsymbol{\beta}^{-2}\right)\mathbf{X}^{\text{T}}\mathbf{X}\right)^{-1}\left(\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 + \text{diag}\left(\boldsymbol{\beta}^{-2}\right)\mathbf{X}^{\text{T}}\mathbf{y}\right).$$

Note that this posterior would require $O\left(K^3\right)$ computations and $O\left(K^2\right)$ storage which is much larger than the above algorithm for sparse data!

2.4 Map-Reduce

The above algorithm is very efficient if all the parameters can be fit into a single machine's memory and all the data can be processed on a single CPU. However, for very large datasets, all the training data will live on a set of distributed files. In order to deal with this situation, we now devise a Map-Reduce approach to scale and parallelize the Expectation Propagation algorithm in blocks of data. Formally, let's assume that our observations are given in M blocks T_m where $T_m := \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_m}$. Then, the posterior can also be written as

$$\begin{aligned} p(\mathbf{w}|T_1, \ldots, T_m) \approx q(\mathbf{w}) &= \prod_{m=1}^{M} \prod_{(\mathbf{x}_i, y_i) \in T_m} \underbrace{\prod_{j \in V_i} \mathcal{N}\left(w_j; \mu_{ij}, \sigma_{ij}^2\right)}_{\approx p(y_i|\mathbf{x}_i, \mathbf{w})} \cdot \underbrace{\prod_{j=1}^{K} \mathcal{N}\left(w_j; \mu_j, \sigma_j^2\right)}_{p(\mathbf{w})} \\ &= \prod_{m=1}^{M} \underbrace{\prod_{j \in \cup_{i \in T_m} V_i} \mathcal{N}\left(w_j; \mu_{mj}, \sigma_{mj}^2\right)}_{\approx p(T_m|\mathbf{w})} \cdot \underbrace{\prod_{j=1}^{K} \mathcal{N}\left(w_j; \mu_j, \sigma_j^2\right)}_{p(\mathbf{w})}. \end{aligned}$$

Here, with slight abuse of notation, we use $i \in T_m$ to denote the set of training set indicies that are in the m-th block of data. Again, the approximating Gaussian $\mathcal{N}\left(w_j; \mu_{mj}, \sigma_{mj}^2\right)$ are called *messages* from factor $p(T_m|\mathbf{w})$ to variable w_j. Again, due to the sparsity of the individual examples, we have at most $D \times N_m$ many messages rather than $O\left(K^2\right)$ that we would need for the full covariance. The Map-Reduce algorithm can now be written as follows:

1. **Initialize** Initialize $m_{lj} = \mathcal{G}(w_j; 0, 0)$ for all $N \times D$ messages for $l \in \{1, \ldots, M\}$.
2. **Initialize** the current marginals $q(w_j) = \mathcal{N}\left(w_j; \mu_j, \sigma_j^2\right)$.

3. **Do**

   (a) Set $\delta = 0$.

   (b) **Map-Phase** on mapper l

      i. For all $j \in \cup_{i \in T_l} V_i$ compute $p(w_j) = q(w_j) / m_{lj}(w_j)$ using (1), (2), and (3).

      ii. Compute the new $q_{l,\text{tmp}}(w_j)$ with the EP algorithm from Subsection 2.3 using the training set T_l and p computed in step i.

      iii. Update $m_{lj}(w_j) \leftarrow q_{l,\text{tmp}}(w_j) / p(w_j)$ using (1), (2), and (3) and forward to the j-reducer.

   (c) **Reduce-Phase** on index j

      i. Compute $q_{\text{new}}(w_j) = \mathcal{N}\left(w_j; \mu_j, \sigma_j^2\right) \cdot \prod_{l=1}^{M} m_{lj}(w_j)$ using (1), (2), and (3).

      ii. Update $\delta \leftarrow \max\left(\delta, \text{KL}\left(q_{\text{new}}(w_j), q(w_j)\right)\right)$

      iii. Update the marginal to $q(w_j) \leftarrow q_{\text{new}}(w_j)$.

4. **Until** $\delta < \varepsilon$

5. **Return** q

While this algorithm uses the same factorization approximation than the single-machine EP algorithm above, the parameter processing is slightly different due to the lack of a centralized, consistent memory in the *map*-phase: Instead of constantly keeping a best possible marginal approximation q, each mapper just computes the best possible approximation of $p(T_l|\mathbf{w})$ as a function of $\mathbf{w}$ but disregards the (local) posterior $q_{l,\text{tmp}}$ (which would have only been computed on N_l many training examples). Instead, the only time that all approximations for $p(T_l|\mathbf{w})$ are multiplied together with the original prior $p(\mathbf{w})$ is in the *reduce*-phase. This de-coupling of memories gives the MapReduce scheme the ability to scale to arbitrarily large training sets.

In contrast, the EP algorithm in Subsection 2.3 is constantly maintaining the best possible approximation $q(\mathbf{w})$ in memory - this is possible because most modern computers have a consistent and fast memory model. However, the consistency of the memory makes the EP algorithm non-scalable.