Learning Complex Classification Models from Large Data Sets

Julian L. Center, Jr.

Creative Research Corp., 385 High Plain Road, Andover, MA 01810, jcenter@ieee.org

Abstract To design a Bayesian classification algorithm, we typically start with a general model form with adjustable parameters and learn a posterior probability distribution for the model parameters based on a set of training data. In many applications, the model form has a large number of parameters, and a large number of samples is needed to narrow the range of probable models. Because the model is complex and there are many samples, computing the likelihood of a particular model takes significant computer time. Therefore, exploring the large model-parameter space in detail becomes an intractable problem.

We outline a computationally feasible solution to this problem based on breaking the large data set into several smaller data subsets and processing the subsets in stages. Each stage combines its subset of the training data with an intermediate distribution that summarizes previous stages. We combine search methods with nested sampling to focus our exploration of the model space on high probability areas. We then use variational methods to approximate the resulting distribution on the parameter space. This approximation summarizes the results and becomes the next intermediate distribution to feed the next processing stage.

THE PROBLEM

A Bayesian pattern classification algorithm computes the probability that a class label k should be associated with an attribute vector $\mathbf{x}$ observed in a specific operating environment. This probability is conditioned on a collection of training data $\mathcal{T}$ that is representative of the operating environment. In other words, the algorithm is designed to compute $p\left(k|\mathbf{x},\mathcal{T}\right)$. (As usual, conditioning on other initial information $\mathcal{I}$ is implied throughout.)

Developing a practical classification algorithm usually begins with the choice of a family of models, with a particular model within the family specified by a vector of model parameters $\theta \in \Theta$. Choosing a model within the family completely determines a classifier, and we assume that $p\left(k|\mathbf{x},\theta,\mathcal{T}\right) = p\left(k|\mathbf{x},\theta\right)$. Neural networks, mixture models, and hidden Markov models are examples of such model families.

Theoretically, the formulas of probability theory tell us that we can determine the probability we want by

$$p\left(k|\mathbf{x},\mathcal{T}\right) = \int_{\Theta} p\left(k|\mathbf{x},\theta\right) p\left(\theta|\mathcal{T}\right) d\theta$$

where we use Bayes' rule to compute

$$p\left(\theta|\mathcal{T}\right) = \frac{1}{Z} p\left(\mathcal{T}|\theta\right) p\left(\theta\right), \text{ where } Z \triangleq p\left(\mathcal{T}\right) = \int_{\Theta} p\left(\mathcal{T}|\theta\right) p\left(\theta\right) d\theta$$

We assume that, for any choice of model θ, we can compute values of the prior $p\left(\theta\right)$ and the likelihood $p\left(\mathcal{T}|\theta\right)$. However, in most cases, the model space Θ is infinite and integrals above cannot be evaluated in closed form. This means we cannot directly compute Z and the integrals must be approximated by summations.

If the parameter space is small (low-dimensional), quadrature methods can be used to approximate the integrals [2]. But for many applications, the models become quite complex, and the associated parameter space becomes quite large. For example, an image recognition algorithm may be based on a mixture model with many components, each with many parameters. In these cases, quadrature methods are impractical.

Several methods have been suggested for approximating $p(k|\mathbf{x},\mathcal{T})$. These include Markov Chain Monte Carlo (MCMC), slice sampling, importance sampling, and nested sampling [2][3][6]. Each of the methods can be viewed as approximating the posterior density $p(\theta|\mathcal{T})$ with a discrete distribution on model space

$$p(\theta|\mathcal{T}) \simeq \sum_{i=0}^{n-1} w_i \delta(\theta, \theta_i)$$

Here, the weights w_i are non-negative and sum to one, and $\delta(\theta,\theta_i)$ is a delta function that peaks when θ and θ_i match. The θ_i's are samples drawn from some probability distribution over the model space, $s(\theta)$. This type of approximation leads to

$$p(k|\mathbf{x},\mathcal{T}) \simeq \sum_{i=0}^{n-1} w_i p(k|\mathbf{x},\theta_i)$$

a form that is sometimes referred to as a "mixture of experts".

Both MCMC and slice sampling strive to sample from the posterior probability distribution $p(\theta|\mathcal{T})$. Therefore, the weights are chosen to be equal, $w_i = \frac{1}{n}$. For importance sampling, the samples are drawn from a known distribution $s(\theta)$. The weights are chosen to be

$$w_i = \frac{r(\theta_i)}{\sum_{j=0}^{n-1} r(\theta_j)}, \text{ where } r(\theta_i) = \frac{p(\mathcal{T}|\theta_i)\, p(\theta_i)}{s(\theta_i)}$$

Nested sampling [6] can be viewed as a form of adaptive importance sampling. First, m samples are drawn from $s(\theta)$ and ranked by the corresponding values of $r(\theta)$. The sample with the smallest value is removed and saved as θ_0.

The next sample is drawn from $s(\theta)$ with the restriction that samples with $r(\theta) < r(\theta_0)$ are rejected. Again the samples are ranked by $r(\theta)$ and the lowest ranked value is removed and saved as θ_1. This process is repeated until a total of n samples has been drawn.

At step i, the probability of drawing a particular sample (and not rejecting it) is

$$s_i(\theta) = \begin{cases} s(\theta)\left[\int_{\{\theta: r(\theta)\geq r(\theta_{i-1})\}} s(\theta)\, d\theta\right]^{-1} & \text{when } r(\theta) \geq r(\theta_{i-1}) \\ 0 & \text{otherwise} \end{cases}$$

Therefore, the weight for sample i is chosen to be

$$w_i = \frac{v_i r(\theta_i)}{\sum_j v_j r(\theta_j)}$$

where

$$v_i \approx \int_{\{\theta: r(\theta)\geq r(\theta_{i-1})\}} s(\theta)\, d\theta$$

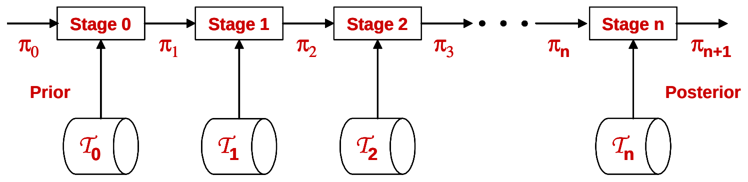

Figure 1: Staged Processing

is an approximation to the volume under $s(\theta)$ restricted to the set where $r(\theta) \geq r(\theta_{i-1})$. Following logic similar to [6], a reasonable approximation is

$$v_i = \begin{cases} \left(\frac{m}{m+1}\right)^i & \text{for } i < n - m \\ \left(\frac{m}{m+1}\right)^{n-m} & \text{for } i >= n - m \end{cases}$$

In effect, these methods approximate the posterior distribution on model space with a mixture of delta functions. In theory, any of these methods can achieve any desired accuracy by including enough samples in the approximation. Unfortunately, the computations needed to determine these approximations often become overwhelming. If we are considering models with a large number of parameters, we need a large amount of training data to narrow the range of probable models. Because the model family is complex and there is a lot of training data, computing the likelihood of a particular model takes significant computer time. Therefore, exploring the large model space in detail becomes an intractable problem.

Of course, if the data set is large enough, the information gain will narrow the range of probable models to a very small subset of the parameter space. If we can find this subspace quickly, we can employ our computational power to adequately explore this region. However, searching for this small region can prove difficult because it is so small and because evaluating each point in the search involves evaluating the complete likelihood function.

A SOLUTION

One approach to solving this problem is to break the large data set into several smaller subsets and design an algorithm for processing the subsets in stages, as shown in Figure 1. That is, we break the training data into disjoint sets

$$\mathcal{T} = \cup_{s=0}^{n-1} \mathcal{T}_s, \text{ with } \mathcal{T}_s \cap \mathcal{T}_t = \varnothing, \forall s \neq t$$

At stage s, the processing algorithm starts with an intermediate probability distribution that summarizes the results of the previous stages

$$\pi_s(\theta) \approx p(\theta | \mathcal{T}_{s-1}, \mathcal{T}_{s-2,} \cdots, \mathcal{T}_0)$$

and works only with its subset of the training data $\mathcal{T}_s$ to produce its own intermediate summary

$$\pi_{s+1}(\theta) \approx p(\theta | \mathcal{T}_s, \mathcal{T}_{s-1,} \cdots, \mathcal{T}_0)$$

Of course, stage 0 starts with the prior distribution

$$\pi_0(\theta) = p(\theta)$$

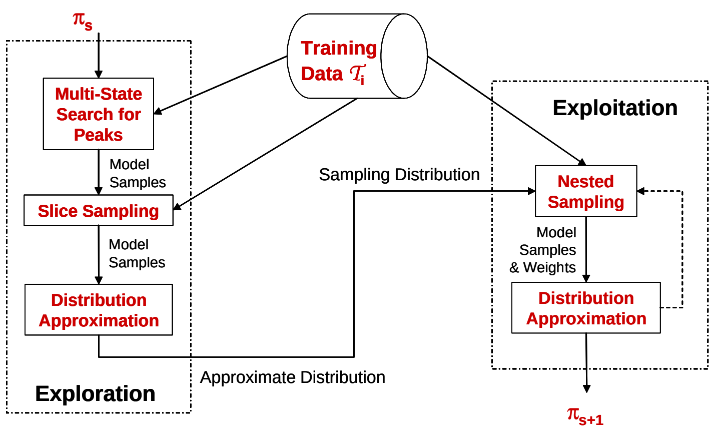

Figure 2: Processing Stage Internal Structure

and final stage produces the desired result

$$\pi_n(\theta) \approx p(\theta|\mathcal{T})$$

This approach offers four advantages: (1) The likelihood function for each subset of the data is far less peaked than the combined likelihood function. This makes the search for the most probable models easier. (2) The evaluation of the likelihood function for each data subset is proportionately easier than for the full likelihood function. Therefore, computations are not wasted on thoroughly evaluating the likelihood function at unlikely points. (3) If additional training data is obtained at a different time, it can be utilized without having to reprocess all of the original training data. (4) Processing can be distributed among many processors, each working with only a subset of the data.

Figure 2 shows a flowchart for one way of implementing each processing stage. Each stage starts with an intermediate distribution summarizing the results of the previous processing stage. We will call this the *input distribution*. The objective is to produce an *output distribution* that summarizes the combination of previous results with information from the local training data set $\mathcal{T}_i$.

Each processing stage is broken into two phases, which we call *exploration* and *exploitation*. In the exploration phase the objective is to try to find all of the significant peaks of the output distribution and then expand outward to arrive at a first approximation to the output distribution. In the exploitation phase, the objective is to refine this first approximation to produce the output distribution for the stage.

In the specific implementation depicted in Figure 2, the exploration phase uses multi-state search methods like the ones discussed by Center [1] and Skilling [5] to find peaks of the output distribution. Then slice sampling methods are used to flesh out the distribution by expanding from these peaks. The result is a set of samples from model space. As noted above, this can be viewed as a discrete distribution on model space. The last step in the exploration phase is to approximate this discrete distribution with a continuous distribution that will act as a sampling distribution during the exploitation phase.

During the exploitation phase, we use slice sampling to draw candidate parameter values from the sampling distribution, and we employ nested sampling techniques to focus on high probability areas of the parameter space. As noted above, nested sampling acts as a form of importance sampling to give a collection of samples and associated weights that form a discrete approximation to the posterior distribution. Finally, we approximate this discrete distribution with a continuous distribution to form the output of the stage. The dashed arrow in Figure 2 indicates that we can iterate the nested sampling and approximation steps to further refine the approximation.

The key to building a practical staged-processing algorithm lies in the approximation form used for the intermediate probability distributions. It must be feasible to compute, and it must be compatible with the search and sampling methods. A discrete distribution is not adequate for this approximation because it limits the models that can be examined in later stages. A specific recommendation for this approximation is presented in the next section.

APPROXIMATE DISTRIBUTIONS

Model Representation

To describe the form of the approximate distribution on model space, we must first examine some different ways of representing models. We will focus on mixture models as a concrete example, but these methods apply to other complex models as well.

We assume that the model space Θ is broken into subspaces Θ_m within which the models are all of the same order. For mixture models, the order corresponds to the number of clusters in the model. For a neural network, the order corresponds to the number of nodes in the network. Of course we have

$$\Theta = \cup_{m=1}^{\infty} \Theta_m \text{ and } \Theta_{m_1} \cap \Theta_{m_2} = \varnothing \text{ if } m_1 \neq m_2$$

Within a subspace Θ_m, we will use three distinct ways of representing a particular choice of model. First, there is the parameter space natural to the model form. For example, we may choose a Gaussian mixture model of the form [1]

$$p(\mathbf{x}, k|\theta) = \sum_{c=0}^{m-1} \gamma_{kc} \beta_c \phi_c(\mathbf{x}; \mu_c, \sigma_c)$$

Here the model parameter set is composed of the cluster weight vector β, the cluster class distribution vectors γ_c, the cluster mean vectors μ_c, and the cluster standard deviation vectors σ_c.

In some cases, there are constraints on the model parameters. For example, in the mixture model above, the β and γ_c vectors must correspond to discrete probability distributions, and the elements of the standard deviation vectors must all be positive. To facilitate some of the operations used in exploring the model probability distributions, we need to eliminate constraints and transform to a model representation that is vector of real numbers that ranges freely over $\mathbf{R}_m = \mathbb{R}^{d_0} \times \mathbb{R}^{d_1} \times \cdots \times \mathbb{R}^{d_n}$. We call this representation the *state space*. For example, in the mixture model, we accomplish this by transforming the m-dimensional cluster weight vector β to an $(m-1)$-dimensional vector $\mathbf{b}$ by the inverse softmax transformation $b_c = \ln \beta_{c+1} - \ln \beta_0$. We can transform back by

$$\begin{aligned}
\beta_0 &= \frac{1}{1 + \sum_{a=0}^{m-2} \exp(b_a)} \\
\beta_c &= \frac{\exp(b_{c+1})}{1 + \sum_{a=0}^{m-2} \exp(b_a)} \text{ for } c = 1 \text{ to } m-1
\end{aligned}$$

For other exploration operations, we need a different representation. To arrive at this new representation, we first transform a point x in a state space to a point y in the Cartesian product of unit cubes $\mathbf{U}_C = (0,1)^{d_0} \times (0,1)^{d_1} \times \cdots \times (0,1)^{d_n}$ by applying the logistic transformation $y_i = [1 + \exp(-x_i)]^{-1}$ to each component. The inverse of the logistic transformation is $x_i = \ln(y_i) - \ln(1 - y_i)$

Finally, we transform to a n-dimensional torus by computing the distance along the Hilbert curve embedded in each of the unit cubes [4]. We represent each of these distances by an extended-precision twos-complement fraction representing a number in the range $[-0.5, 0.5)$. Since addition and subtraction in twos-complement arithmetic wraps around from 0.5 to -0.5, we consider each of these mappings as a mapping to a circle and the combined mapping as onto a n-dimensional torus. We designate this space by $\mathbf{H}_m = H_{d_o} \times H_{d_1} \times \cdots \times H_{d_n}$ and call $\mathbf{H} = \cup_m \mathbf{H}$ the *code space.* We represent the code for model $\theta \in \Theta_m$ by a vector of extended-precision fractions $\mathbf{h}(\theta) \in \mathbf{H}_m$

Approximation Form

To make this approach feasible, we must approximate the intermediate distributions using a manageable form. We choose the following distribution on code space

$$\pi_s(\theta) \sim \sum_{j=0}^{p-1} a_j \varepsilon\left(\theta; \widehat{\theta}_j, \sigma_j\right) + a_p \psi(\theta; \lambda) + a_{p+1} \zeta(\theta)$$

Here the a's are positive weights that sum to one. Each $\varepsilon_j\left(\theta; \widehat{\theta}_j, \sigma_j\right)$ is a probability distribution in model space that measures the similarity of θ to a reference model $\widehat{\theta}_i$. Specifically, for each $\varepsilon\left(\theta; \widehat{\theta}_j, \sigma_j\right)$, we choose a truncated Gaussian distribution on the appropriate subspace of code space, which takes the form

$$\varepsilon\left(\theta; \widehat{\theta}_j, \sigma_j\right) = \begin{cases} \frac{1}{z_j} \exp\left[-\frac{1}{2} \sum_k \left(\frac{h_k(\theta) - h_k\left(\widehat{\theta}_j\right)}{\sigma_{jk}}\right)^2\right] & \text{if } \theta \text{ and } \widehat{\theta}_j \text{ are the same order} \\ 0 & \text{otherwise} \end{cases}$$

Here, $h_k(\theta)$ represents the k^{th} component of the Hilbert code for θ (viewed as an extended-precision fraction), σ_{jk} is a scale parameter (represented as an extended-range double-precision number), and z_j is the normalization necessary to make $\varepsilon\left(\theta; \widehat{\theta}_j, \sigma_j\right)$ a probability distribution. Since the extended-precision fractions only range over $[-0.5, 0.5)$, z_j must take that into account. We will call these terms in the approximation *clumps.*

The last two terms in the equation for $\pi_s(\theta)$ depend only on the model order $m(\theta)$. The function $\psi(\theta)$ represents a Poisson distribution on the model order $m(\theta)$

$$\psi(\theta; \lambda) = \frac{\lambda^{m(\theta)} \exp(-\lambda)}{m(\theta)!}$$

The parameter λ determines how fast the probability decays with order. The function $\zeta(\theta)$ represents a second-order zeta distribution

$$\zeta(\theta) = \frac{6}{\pi^2} \frac{1}{[m(\theta) + 1]^2}$$

which decays relatively slowly with model order.

Fitting the Approximation

To implement the approach in Figure 2, we must fit a continuous distribution $\pi(\theta)$ of the form above to a discrete distribution

$$q(\theta) = \sum_{i=0}^{n-1} w_i \delta(\theta, \theta_i)$$

that comes from either slice sampling or nested sampling. We start by approximating $q(\theta)$ by simply replacing the delta function by truncated Gaussians at the same centers, $\varepsilon\left(\theta; \widehat{\theta}_i, \sigma_i\right)$ with $\widehat{\theta}_i = \theta_i$. The spreads σ_j control the smoothness of the approximation and are initially chosen so that, if $\widehat{\theta}_j$ and $\widehat{\theta}_i$ are of the same order, then $\sigma_j = \sigma_i$. For the Poisson term, we start with the decay rate λ equal to the weighted average of the orders of the models in the discrete distribution. The weights of the Poisson and zeta distribution terms are set to predetermined starting values and the other weights are reduced so that all the weights sum to one. This gives a starting approximation that has as many clumps as there are sample models in the discrete distribution $q(\theta)$.

Next, we use a graph contraction method to reduce the number of clumps by consolidating clumps that are close to each other. To accomplish this, each clump is represented by a node in the graph. The closeness of two nodes corresponding to $\widehat{\theta}_i$ and $\widehat{\theta}_j$ is measured by their affinity, which we choose to be $\varepsilon\left(\widehat{\theta}_i; \widehat{\theta}_j, \sigma_j\right)$ as defined above. Two nodes can be combined if their affinities to all other nodes are similar enough. We combine two nodes (clumps) by simply setting the weight of one clump to zero and adding its original weight to the weight of the other clump.

Now we fit this reduced approximation π to the discrete distribution $q(\theta)$ by minimizing the Kullback-Leibler divergence

$$D_{KL}(q||\pi) = E_q\left[\ln \frac{q(\theta)}{\pi(\theta)}\right] = -\sum_{i=0}^{n-1} w_i \ln \pi(\theta_i) + \sum_{i=0}^{n-1} w_i \ln w_i$$

The minimum can be found by an iterative algorithm very similar in form to the standard EM algorithm for mixture models. We avoid degeneracies by imposing a lower limit on the components of the σ's and by eliminating clumps with weights below a minimum level.

The iterative process is given by the equations below. Here the index i ranges over the sample models in the discrete distribution and j ranges over the components of the approximation. We start a step in the iteration with approximation parameters $a_j^-, \widehat{\theta}_j^-$, and σ_j^-. First we compute the similarities of the model components to the sample models

$$s_{ji} = \left\{ \begin{array}{ll} a_j^- \varepsilon\left(\theta_i; \widehat{\theta}_j^-, \sigma_j^-\right) & \text{for } j = 0 \text{ to } p-1 \\ a_p^- \psi(\theta_i; \lambda^-) & \text{for } j = p \\ a_{p+1}^- \zeta(\theta_i) & \text{for } j = p+1 \end{array} \right\} \text{ for } i = 0 \text{ to } n-1$$

Then we compute the responsibilities

$$\rho_{ji} = \frac{s_{ji}}{\sum_k s_{ki}}$$

and update the approximation parameters by

$$
\begin{aligned}
a_j^+ &= \sum_i w_i \rho_{ji} \\
h_k\left(\widehat{\theta}_j^+\right) &= h_k\left(\widehat{\theta}_j^-\right) + \frac{1}{a_j^+} \sum_i w_i \rho_{ji} \left[h_k(\theta_i) - h_k\left(\widehat{\theta}_j^-\right)\right] \\
\sigma_{jk}^+ &= \frac{1}{a_j^+} \sum_i w_i \rho_{ji} \left[h_k(\theta_i) - h_k\left(\widehat{\theta}_j^-\right)\right]^2 \\
\lambda^+ &= \frac{1}{a_p^+} \sum_i w_i \rho_{pi} m(\theta_i)
\end{aligned}
$$

To retain adequate precision, these equations were implemented in C++ using abstract data types. A vector of extended-precision fractions was used to implement $h(\theta)$, the Hilbert code of model θ. The difference $h_k(\theta_i) - h_k\left(\widehat{\theta}_j^-\right)$ was implemented as an extended-range double-precision number, and suitable arithmetic operations for combining these extended types were implemented.

CONCLUSION

A staged-processing approach makes Bayesian learning of complex models feasible for large data sets. The key to this approach is a method of approximating intermediate probability distributions that is both computationally feasible and compatible with established sampling methods. One such approximation is based on a mixture of truncated Gaussian distributions on a coded model representation based on the Hilbert curve mapping.

References

[1] J. Center, "Approximating Posterior Distributions for Mixture-Model Parameters" in R. Fischer, R. Preuss, U. von Toussaint, *Bayesian Inference and Maximum Entropy Methods in Science and Engineering 24*, American Institute of Physics, Melville, NY, 2004, pp. 437-444.

[2] M. Evans and T. Swartz, *Approximating Integrals via Monte Carlo and Deterministic Methods,* Oxford University Press, 2000.

[3] D. MacKay, *Information Theory Inference, and Learning Algorithms,* Cambridge University Press, 2003, pp. 387-397.

[4] J. Skilling, "Programming the Hilbert curve," in G. Erickson and Y. Zhai, *Bayesian Inference and Maximum Entropy Methods in Science and Engineering 23*, American Institute of Physics, Melville, NY, 2004, pp. 381-387.

[5] J. Skilling, "Using the Hilbert curve," in G. Erickson and Y. Zhai, *Bayesian Inference and Maximum Entropy Methods in Science and Engineering 23*, American Institute of Physics, Melville, NY, 2004, pp. 388-405.

[6] J. Skilling, "Nested Sampling, " in R. Fischer, R. Preuss, U. von Toussaint, *Bayesian Inference and Maximum Entropy Methods in Science and Engineering 24*, American Institute of Physics, Melville, NY, 2004, pp. 395-405.