Chapter 2. Sparse Bayesian Learning

Supervised machine learning is a generic term for methods which take some set of labeled data – that is, items of data about which we have some special knowledge, such as whether or not a DNA sequence comes from a gene's promoter region – and build some kind of model which can be used to predict the label values for additional, unseen data. This is quite different to unsupervised machine learning methods, which are used to detect patterns in sets of unlabeled data.

In principle, simply storing away (memorizing) the training data would count as a valid method of machine learning, but this is only likely to be of use if all the unseen data is very similar to pieces of data which have already been seen. Much more interesting are methods which can generalize – build a model which contains less information than the complete training set, but which still match the labeling on the supplied data. Learning methods with good generalization properties are intuitively more likely to correctly interpret unseen data which is significantly different from any of the pieces of training data, and practical experience throughout the history of machine learning has backed up this intuition. Depending on the exact techniques used, the internal model state of a machine learning system may be amenable to inspection or visualization by human users, who can recognize the generalizations made by the method, and perhaps gain some additional understanding of the problem in hand. Returning briefly to the question of classifying sequences (covered in much more detail in chapter 3), a memorization-type solution might simply encode the complete set of training data, but would probably only make trustworthy predictions for sequences which showed strong similarity along their full length to a sequence in the training set. A more general solution would be to identify some common short motifs or compositional biases which occur in promoter regions. This could yield a much more widely applicable model, and also one which would be of direct interest to

researchers wishing to understand transcription initiation.

It should be noted that generalizing learning machines are not automatically "transparent" in terms of being able to extract the rules which have been learned in a form which can be presented to humans. This criticism has been made in particular about machine learning systems based on neural network approaches. While "black box" prediction systems, when suitably validated, can be useful tools, it is always preferable to understand the basis for the predictions that are made. It is therefore worth specifically considering the issue of transparency when developing or choosing new machine learning technologies.

This chapter introduces one particular approach to supervised learning, called Sparse Bayesian Learning, which based on recent theoretical developments by MacKay and Tipping. Later sections discuss a specific "user oriented" implementation of the Sparse Bayesian Learning approach (as opposed to development implementations in simple test harnesses), which is available as a Java library routine, and can be applied to a wide range of real-world machine learning applications.

2.1. Generalized Linear Models (GLMs)

Generalized Linear Models are common mathematical devices, by which the value of a real function is represented as the weighted sum of a number of basis functions [McCullagh and Nelder 1983]. The general formulation is:

$$\eta(\mathbf{x}) = \sum_i \beta_i \phi_i(\mathbf{x}) + K \qquad (2.1.1)$$

Where β represents a vector of weights, ϕ is the set of basis functions, and K is a constant. The basis functions can be *any* real-valued function of $\mathbf{x}$. In the mathematical literature, $\mathbf{x}$ is normally a vector, and a common choice of basis function is a hyperspherical Gaussian around some point in the appropriately-dimensioned space (commonly called the radial basis function). However,

since the GLM only refers to $\phi(\mathbf{x})$, $\mathbf{x}$ could actually be any type of data – including strings or even records of structured data – so long as a suitable family of basis functions can be defined. Analogous to the radial basis function, these functions will typically be simple functions of a distance metric from some point in a hypothetical "data space".

It is common to refer to the high-dimensional space implied by a list of basis functions as feature space. The common theme of generalized linear modeling is to pick a projection from the natural data space, where the distribution of data may be non-linear (and potentially extremely complex), into a feature space where a linear model is a good fit to the data. Obviously, this means that the choice of feature space is important, and requires either problem-specific knowledge or an appropriate automatic method – this is discussed further in the next section.

This basic form of a GLM is normally applied as a regression system – to estimate the value of a continuous function. However, for many machine learning tasks, it is more interesting to investigate class membership. As an example relevant to this thesis, a DNA sequence $\mathbf{x}$ might belong to the class P (promoter) or not-P. GLMs are applied to classification problems using a link function. For binary classification tasks, the logistic function (figure 2.1) is a common choice:

$$\pi = \sigma(\eta) = \frac{1}{1 + e^{-\eta}} \tag{2.1.2}$$

This function has the desirable property that $0 \leq \pi \leq 1$. By fitting appropriate values for β, π is an estimate of $p(\mathbf{x} \in P)$, our posterior belief that $\mathbf{x}$ is a promoter given the learned model.

It is also possible to use multiple GLMs to separate data into more than two classes. In this case, each class is modeled by its own set of weights, and the probability of a data point being a member of class c is given by the multinomial link function:

$$\pi_c = \frac{e^{\eta_c}}{\sum_{c'=1}^{C} e^{\eta_{c'}}} \tag{2.1.3}$$

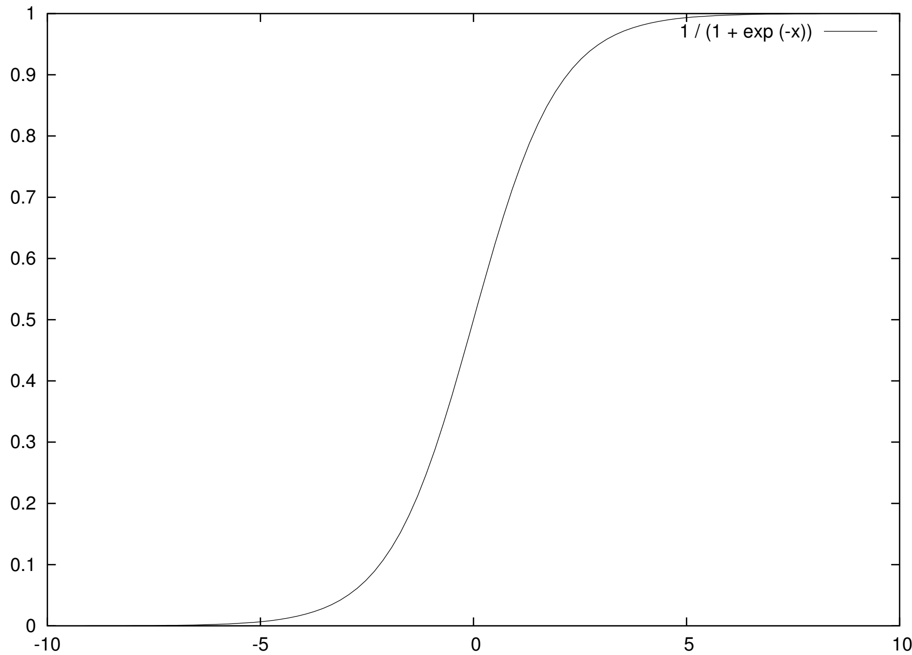

Figure 2.1. Plot of the logistic function (equation 2.1.2).

This multi-GLM formulation could, for instance, be used to build a classifier capable of distinguishing between promoter sequences which are active in several different biological environments.

One point to note about GLM classifiers is that while, given suitable values of β, they can serve as probabilistic models, they are not "end-to-end" probabilistic in the sense of some types of hidden Markov model, where every parameter in the model can be interpreted a probability. The values of the individual basis functions need not be probabilities themselves, and it is the responsibility of the training algorithm to fit a model where the output value has a probabilistic interpretation.

In its simplest form, training a GLM simply means picking a set of values for β and K. For regression questions, the best-known algorithm for this is the least-squares method [Lawson and Hanson 1995], an analytical approach which simply minimizes the square of the error at each point of training data. This method is not directly applicable to classification problems, but comparable methods do exist: for example, iterated reweighted least squares

[Nabney 1999]. However, all these methods are only useful if the data has already been transformed into a suitable feature space.

2.2. Feature selection using pruning priors

The utility of machine learning algorithms is severely limited if users have to apply large amounts of domain-specific knowledge to project the data into a small, meaningful, feature space. Therefore, it is interesting to consider methods which can work in very high-dimensional feature spaces and automatically select smaller, informative subspaces. Methods which combine selection of features with learning an optimal set of parameters are called sparse learning algorithms. The classic example of a sparse learning algorithm, and to date the most widely used, is the Support Vector Machine (SVM) [Schölkopf *et al.* (eds.) 1999], which can learn one particular class of generalized linear model in a sparse manner, often reducing a candidate feature space with thousands of basis functions down to just a handful. However, the restrictions on problems which can be solved with support vector machines are rather stringent: instead of supplying an arbitrary collection of basis functions, the user must supply a single kernel function $K(\mathbf{x}, \mathbf{y})$, whose value is the inner (dot) product of $\mathbf{x}$ and $\mathbf{y}$ when projected into the desired feature space. The basis functions of the learned GLMs are always of the form $\phi_i(\mathbf{x}) = K(\mathbf{x}, \mathbf{x}_i)$, where $\mathbf{x}_i$ is one of the examples from the training data set. An alternative way of looking at this is that the algorithm works on the Gram matrix: a square matrix $\mathbf{A}$ where:

$$\mathbf{A}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) \tag{2.2.1}$$

In addition, Mercer's condition, which forms part of the derivation of the support vector machine algorithm, states the kernel function – and thus the Gram matrix – must be positive definite, i.e.:

$$\mathbf{z}^{\top}\mathbf{A}\mathbf{z} \geq 0 \tag{2.2.2}$$

for any vector, **z**, with a length matching the edge size of the square matrix. Proving that an arbitrary function satisfies this condition can be a demanding requirement, and presents a barrier to simply plugging new types of kernel into the SVM framework.

While SVMs have proved to be effective tools for some biological problems (see, for example, [Furey *et al.* 2000]), the constraints on the forms of models which can be learned can be problematic. Most well-known kernel functions are applicable only to numerical data. Some effort has been made to develop kernel functions which can be applied to alternative data types, such as strings [Jaakkola and Haussler 1999] and graphs [Kondor and Lafferty, 2002], but the requirement to prove that the function used is positive definite makes this problematic, and prevents people without strong mathematical backgrounds from developing new SVM applications. Perhaps more seriously, the kernel view of GLMs does not allow models to be learned with basis functions that pick truly arbitrary sub-spaces of the training data space (for example, just considering dimensions 5, 8, and 10 of vector data, or only one region of a long string), since only the set of basis functions implied by the training data are available. Very recently, a "kernel-like" learning system has been developed which adjusts an additional set of scaling variables to select informative dimensions [Krishnapuram *et al.* 2003], but unlike the approach described below this is only applicable to vector data.

For these reasons, it is interesting to consider alternative forms of learning which can be used to train arbitrary GLMs while achieving sparsity comparable or better than that of SVMs. One extremely promising approach which fulfills these requirements is the Relevance Vector Machine (RVM) [Tipping 2000]. This was originally presented as a direct (and competitive, on the basis of performance on standard machine learning test problems) alternative to the SVM, with basis functions implied from a single kernel function and a set of training data. However, unlike the SVM there is no technical reason why this formulation is necessary, and it is entirely

feasible to implement an RVM-like learning method which uses arbitrary basis functions.

Briefly, the RVM is a Bayesian probabilistic view of training a GLM as defined by equation 2.1.1. In other words, the question is, given a list of training data, X, and a corresponding list of labels or expected outcomes, t, to find probable values of the weights vector, β which make the model outputs match the supplied labeling.

The basis of all Bayesian statistics is Bayes' theorem:

$$P(a|b) = \frac{P(a)P(b|a)}{P(b)} \tag{2.2.3}$$

This says that, given some prior knowledge of the probability of a ($P(a)$), and the conditional probability of b given a (the likelihood, $P(b|a)$), it is possible to calculate the probability of a given b. In cases where there is no prior knowledge whatsoever, a flat (non-informative) prior distribution can, of course, be specified. The evidence term, $P(b)$, is generally treated simply as a normalizing constant. Bayes' theorem supports a modeling view of statistics and learning: if a is the parameters of the model, b is the training data, and the likelihood function $P(b|a)$ encapsulates the logic of the model, Bayes' theorem offers us a probability distribution over possible values for the model parameters.

For a two-way classification model, with training data labeled as either positive ($t_n = 1$) or negative ($t_n = 0$), and treating each element of the training set independently, a likelihood function – the probability of that set of labeled data given a particular set of model parameters – can be written as:

$$P(t \mid X, \beta) = \prod_{n=1}^{N} \sigma(\eta_n)^{t_n} \left(1 - \sigma(\eta_n)\right)^{1-t_n} \tag{2.2.4}$$

where η_n is the linear model output for the nth example in the training set. It is possible to write comparable probabilistic formulations for regression problems by specifying some probability

distribution for errors relative to the model output, then proceed in an analogous fashion, but regression problems fall outside the scope of this thesis.

The second part of the Bayesian inference process is the prior distribution over the parameters being inferred. In general, the preferred choice is a non-informative prior, implying that before the inference operation we have no knowledge of what the parameters are likely to be. However, in this case there *is* a prior preference: if possible, we want to learn sparse models. This is encoded in the RVM by using a more sophisticated prior. The basic prior is an independent Gaussian distribution, $\mathcal{G}$, over the weight of each basis function:

$$P(\beta) = \prod_i \mathcal{G}(\beta_i \mid 0, \alpha_i^{-1}) \tag{2.2.5}$$

The "RVM trick" is to define the inverse variances of these Gaussian distributions, α, as variables, and to infer their values as well. It is therefore, of course, necessary to provide an additional hyperprior over values of these priors. For the hyperprior, a conventional choice of non-informative prior is used: a very broad gamma distribution. Since the parameters of this are chosen such that the distribution is essentially flat over a wide range of "reasonable" values of α, the exact choice of function is in fact irrelevant except for issues of computational convenience. This form of prior is known as an automatic relevance determination (ARD) prior, and was first proposed (in a somewhat difference application, relating to the training of neural networks) in [Mackay 1994].

The inclusion of an ARD prior has been described as an Occam term, since it rewards simplicity: when the α parameter tends to infinity, the probability of β values close to zero becomes extremely high, as can be seen by extrapolating figure 2.2. So for a basis function which cannot directly contribute to the likelihood of the data, the joint likelihood of the data and the β parameters is maximized by setting α to a large value and β to zero. As we shall

see, inferring likely parameters for this model requires an iterative process. After a number of cycles, some of the α values become large. That means that the corresponding weight parameter, β_i, is well-defined with a value extremely close to zero. When α exceeds some large threshold, we can assume that the corresponding dimension of feature space is irrelevant, and not making a substantial contribution to the calculations. At this point, it is reasonable for practical implementations of the algorithm to simply drop that dimension from further calculations. In this way, sparse models are obtained. In addition, the computational cost of each iteration falls with the number of dimensions under consideration.

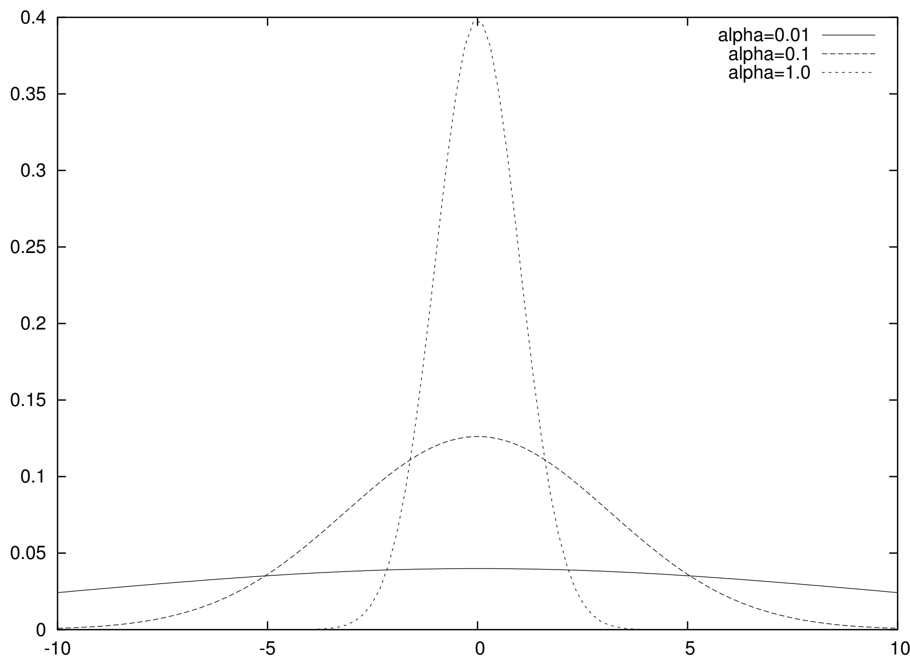

Figure 2.2. Plots of the Gaussian distribution, $\mathcal{G}(\beta_i \mid 0, \alpha_i^{-1})$, for various values of α.

Given this probabilistic view of the problem, Bayes' theorem provides us with the distribution of probable values for the weights:

$$P(\beta \mid X, t) = \frac{\int_{\alpha} P(\beta|\alpha)P(\alpha)P(X, t|\beta)}{P(X, t)} \tag{2.2.6}$$

Note that the α parameters are hidden variables of the model, and not of direct interest. Therefore, this formula marginalizes α by integrating over all possible values. This is the

standard Bayesian approach to handling all parameters whose values are not specifically required in the inference.

Unfortunately, like most Bayesian problems, formula 2.2.6 is not directly tractable, due to problems evaluating the evidence term, $P(X,t)$. Therefore, it is necessary to take one of a variety of approximate methods in order to obtain likely values for β and α. The simplest approaches to solve such problems are Monte-Carlo methods, which make it possible to draw samples from the posterior distribution $P(\beta \mid X,t)$. The basic Monte-Carlo method is the Metropolis-Hastings algorithm [MacKay 2003]. The principle here is that if we wish to sample from the probability distribution $P(x)$, we perform a random walk within this distribution, then take some of the states which are visited as samples from the distribution. So long as a sufficiently large number of steps are made between samples, they will be independent samples from $P(x)$. To perform the random walk, it is necessary to offer a proposal distribution, $Q(x' \mid x)$ which suggests states to try next conditioned on the current state, x, and to be able to easily sample from this distribution. For each step, a new state, x' is proposed by drawing a sample from $Q(x' \mid x)$, and the following quantity is calculated:

$$\alpha = \frac{P(x')}{P(x)} \frac{Q(x \mid x')}{Q(x' \mid x)} \tag{2.2.7}$$

If $\alpha \geq 1$, the new state is always accepted. Otherwise, the new state is accepted with probability α. There are two important points here: firstly, the exact distribution chosen for Q is not important: the second term of equation 2.2.7 counteracts any bias in the set of states which are proposed. However, a bad proposal distribution may mean that very few proposals are accepted and the random walk proceeds very slowly. Secondly, since it is only necessary to calculate the ratio $P(x')/P(x)$, it is not necessary to calculate any constant terms in P, so the evidence term of distributions such as 2.2.6 can be neglected.

I wrote a naive implementation of the Relevance Vector Machine algorithm which used the basic Metropolis-Hastings algorithm. This was able to solve simple problems, such as that

shown in figure 2.3, but required several minutes of processor time. Figure 2.3 also shows off an attractive side effect of using probabilistic classification methods: each prediction comes with a confidence level. While points close to the training data are shown in deep red or blue, indicating close to 100% confidence in the prediction, points that are some distance from training data of either class appear in paler shades, indicating a much lower confidence. Non-probabilistic methods such as support vector machines cannot provide this information.

Some methods exist for optimizing Monte-Carlo simulations: for example, Skilling's leapfrog [MacKay 2003]. For this project, I did not follow this course further, since the alternative method described below gave good performance and worked well on the problems I considered. However, further investigation of Monte Carlo RVM implementations might be interesting in the future, for cases where the variational solution cannot be applied.

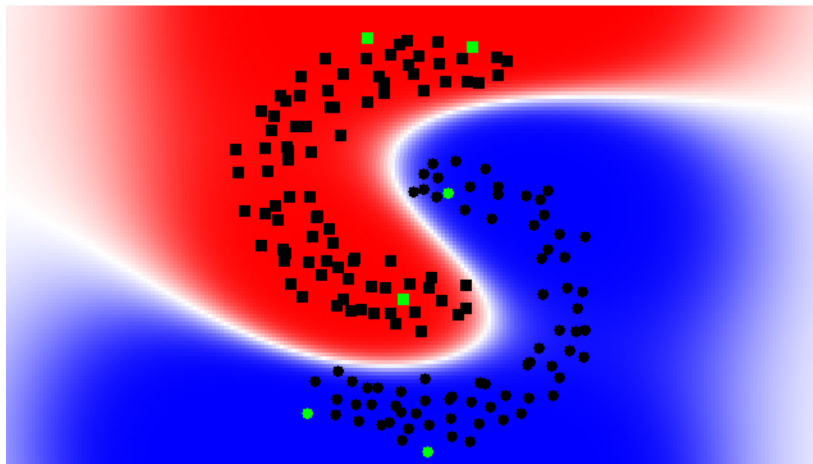

Figure 2.3. Example of sparse Bayesian learning in Cartesian 2-space. The data points chosen as centers for the final set of basis functions are highlighted in green.

An alternative approach is to use free-form variational inference [MacKay 1995]. This is a recent approach to Bayesian inference whereby an intractable posterior distribution is approximated by alternative, simple, probability distributions. Analytical formulae can be obtained which minimize the difference between the true and approximating posterior distribution. A variational approximation of the RVM posterior distributions is given in [Bishop and Tipping 2000]. Like many variational inference solutions, this uses a factorisable

posterior distribution (i.e. independent distributions for all parameters). Although the variational approximation yields formulae to analytically determine the moments of one approximating distribution given values for the other, it is still necessary to iterate the process a number of times in order to fit all the parameters to the data. The variational RVM approach can be applied effectively to both regression and binary classification problems. However, for classification problems it relies on transforming the logistic function to a convex form, which allows a linear approximation to be made using Jensen's inequality. This appears not to be practical in the case of the multinomial function, therefore multi-class problems must still be solved using an alternative method such as Metropolis-Hastings.

Implementations of the RVM methodology which iteratively apply the variational estimators from [Bishop and Tipping 2000] are able to solve problems of the scale of figure 2.3 in a timescale of around 10 seconds on a typical desktop computer. Unlike Monte-Carlo models, it is possible to specify a clearly objective stopping criterion, halting iterations once the training process has "converged" (i.e. the moments of the approximating distributions no longer change significantly from one cycle to the next).

2.3. A pragmatic approach to handling large spaces

In principle, we would like to provide as little prior knowledge as possible about which feature spaces to consider when solving a problem, instead allowing a sparse trainer such as the RVM to pick freely from a wide range of possible basis functions. Unfortunately, even when using the variational approximations to avoid the inefficiencies of a sampling strategy, computational costs become significant. The progressive simplification of the problem means that it is difficult to quantify the full computational complexity since the size of the matrices varies from cycle to cycle. However, scaling is quite substantially worse than linear, as can be seen in figure 2.5.

Here, a pragmatic approach to handling large sets of candidate basis functions is suggested: a working set is first initialized with a subset of basis functions, picked at random from the pool of candidates. The trainer runs as previously described, and as before some α values increase to a level such that it is possible to remove the associated basis functions from further consideration. Once the size of the working set drops below a designated low water mark, additional basis functions are added from the pool. At this point, all α and β values are reinitialized and training continues. In this way, the working set fluctuates between high and low water marks until the pool is exhausted, at which point the trainer continues to run until the weights and priors no longer change significantly between cycles (*i.e.* convergence) to give a complete model.

To evaluate the performance benefits of this method, I considered a simple classification problem in Cartesian 2-space. This involved two equal-sized classes of points sampled from two Gaussians, with a substantial space between them. Basis functions were generated in an "SVM-like" manner, with a radial basis function centred on each point in the training data. Thus, the size of the basis-value matrix of the sparse Bayesian trainer increases with the square of dataset size. As shown in figure 2.4, an optimal model requires only two basis functions, and gives 100% classification accuracy on the training data.

A range of problem sizes were tested, from 50 points in each class (i.e. 100 basis functions) to 300 points. I compared the basic full-set training method with the incremental method, picking a high water mark of 20 basis functions and a low water mark of 15. Timings are shown in figure 2.5. For small problems, the incremental training approach is in fact significantly slower, due to the increased number of cycles and the need to periodically restart the training process as more basis functions are added. However, the scalability of the method to large problems is substantially better.

It is interesting to note that, while in all cases all the training data was correctly classified, the full-set training runs with 300 points did not learn the simple 2-basis function model seen with smaller training sets, but instead converged before all unnecessary basis functions had

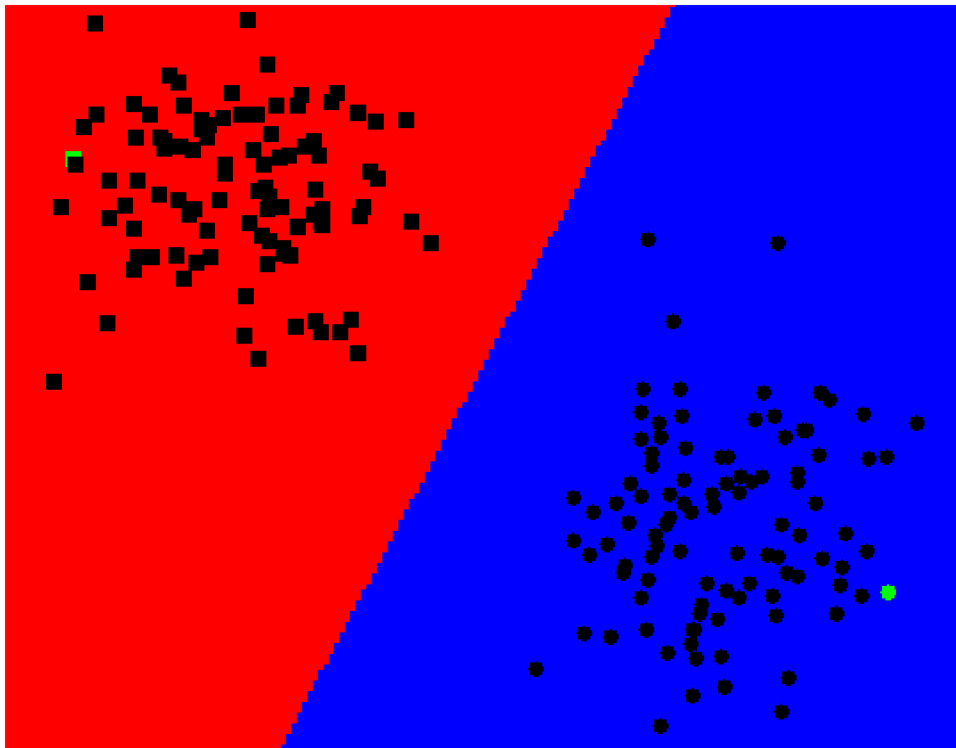

Figure 2.4. Example of a data set used for testing of training speed.

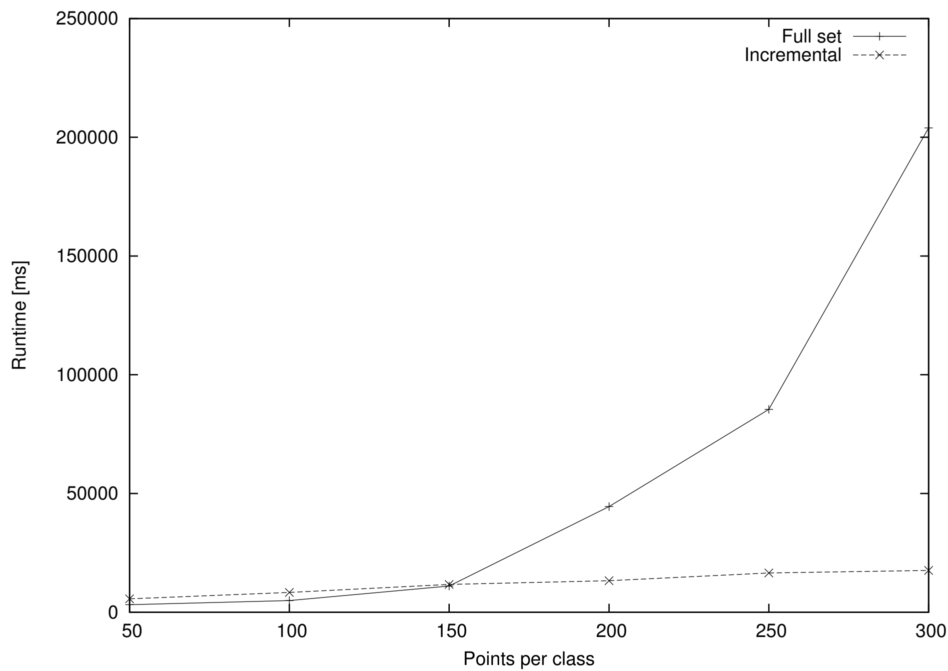

Figure 2.5. Training time vs. problem size for full-set and incremental sparse Bayesian learning.

been removed. However, the incremental trainer consistently learned a sparse model for the

same dataset with no complications. I believe that this indicates the limits of either the linear algebra routines used by the trainer, or the numerical precision of the hardware on which it was running. Errors caused by lack of precision (often described as numerical instability) are a common problem in numerical computation, and large linear algebra computations are a frequent source of trouble. By keeping the size of individual computations down, using the small-working-set variant of the RVM neatly sidesteps these problems and makes it possible to tackle large problems.

Incremental training does have one potential drawback: if the chosen working set sizes are insufficient to cover the actual complexity of the problem in hand, it is possible for the training process to become "stuck" in a state where adding extra basis functions could increase classification accuracy, but it it not possible to remove any functions from the current set. The implementation described here applies some simple checks to detect the stuck state and can, when appropriately configured, automatically increase the user-specified high and low watermarks, but this mechanism has not been intensively tested, so it is suggested that some care is taken in picking appropriate watermark values.

Having made the step toward the approach of gradually sifting through a large set of basis functions, it is only a small additional step to consider an implementation which generates new basis functions on the fly during the training process. In principle, this can allow exploration of infinite sets of basis functions. This is particularly feasible in cases where there are clear correlations between basis functions. For instance, a radial basis function with centre (10.0, 10.2) and variance 10 will have outputs that are highly correlated to a second radial basis function with centre (10.0, 10.0) and equal variance. This means that if the first function is found to be informative for modeling a particular problem, it is likely that the second will also be effective. Therefore, after some period of training, it is likely to be worthwhile proposing more candidate basis functions which are correlated to those currently in the working set.

2.4. A general-purpose Sparse Bayesian trainer

During this project, and especially the work described in chapter 3, the most useful form of sparse learning was the variational implementation of binary classification. I implemented this, and several other algorithms, for the Java 2 platform, as described in chapter 1. I used a number of APIs from the COLT library [Hoschek *et al.* 2000], which offer high-performance implementations of common linear algebra operations for Java programmers. Java code using COLT is often little more verbose than dedicated mathematical programming environments such as Matlab. Source code for the RVM library is available under the terms of the GNU lesser GPL on request from Thomas Down.

It is possible for programmers to make use of the trainer without detailed knowledge of the inference process and how it is implemented internally. Basis functions are provided by writing one or more implementations of the *BasisFunction* interface

```
package stats.glm;

public interface BasisFunction {
    /**
  * Return a feature value for a particular object.
  *
  * @throws ClassCastException if this BasisFunction cannot
  * be evaluated for an object of
  * this type.
  */

    public double evaluate(Object o);
}
```

These *BasisFunction* objects can in principle work with *any* object in the Java virtual machine. Obviously, it is the user's responsibility to ensure that the basis functions match the supplied training data.

Training data is supplied using the *SVMTarget* class, part of a support vector machine toolkit which was previously developed by the author, and is included in the BioJava library.

Reusing code in this way saved duplicated development effort, and makes the Sparse Bayesian trainer more accessible for people who already use BioJava.

The final part of the system is the *BasisSource* interface, which is, as its name suggests, a source of *BasisFunction* objects. For conventional use, the supplied *ListBasisSource* simply provides basis functions from a pre-defined list, until it is exhausted. For training systems which use sampling approaches, developers will have to write their own implementation of *BasisSource*, providing the desired sampling moves.

All these elements are presented to the training code, which returns a model object including the chosen set of basis functions and their weights:

```
// Initialize

SVMTarget target = loadTrainingData();
BasisSource basisSource = new ListBasisSource(basisFunctions);
VRVMTrainer trainer = new VRVMTrainer();

// Train model

GLMClassificationModel model;
model = trainer.trainClassification(
        target,
        basisSource,
        new SimpleTrainingListener()
);

// Print results for test data

for (Iterator i = testData.iterator(); i.hasNext(); ) {
    Object testDataPoint = i.next();
    System.out.println(
        testDataPoint.toString() +
        " -> " +
        model.positiveProbability(testDataPoint)
    );
}
```

The final parameter in the call to the trainer is an implementation of a simple callback interface which receives status notifications during the training process. This is especially useful if the

trainer is to be embedded in a user-oriented application, since it makes it easy to hook up some graphical feedback of training progress.

2.5. Sparse Bayesian Learning Discussion

Sparse Bayesian Learning using the automatic relevance determination prior offers a convenient and principled approach towards developing machine learning systems which actively penalize unnecessary complexity in the model, and consequently build the simplest model which can still give a good fit the training data. While the connection is not absolutely straightforward, both intuition and experience tell us that sparsity usually translates to making generalizations about the supplied data (rather than overfitting or memorizing it) and consequently making good predictions from unseen pieces of data – a valuable property in any learning system. Moreover, since the model output is a probability value, it is possible to distinguish between cases where enough information is available to make a confident prediction and cases where a piece of data is only marginally more likely to fall in one class than the other. It is principled to apply an arbitrary threshold to these scores and, for example, consider only those predictions with 99% confidence.

Throughout this chapter I have concentrated on one specific form of Sparse Bayesian Learning – binary classification GLMs. This was the form which I found directly useful in the course of this project. It is possible to apply the Automatic Relevance Determination principle of sparse learning to other problems – for example multi-way classifiers, but in some cases it is no longer possible to use the elegant analytical approximations obtained by variational inference. Developing these forms of learning into practical methods suitable for widespread application will mean investigating alternative approaches to the inference problem. Optimizations which allow independent samples to be drawn from a Monte Carlo simulation with less computation than is needed for the basic Metropolis-Hastings approach would seem to be a profitable direction to explore.

I have implemented and tested a practical implementation of Sparse Bayesian classification. This is freely available as a Java library. Using the simple interfaces described in this chapter, it is possible to write new types of basis function, and thus apply the trainer to new types of data, without requiring substantial knowledge of variational inference or the internal implementation details of the method. This library is used without further description as the "learning engine" driving the sequence analysis methods described in chapters 3 and 4. The results in these chapter additionally provide validation of the method, and show that it is extremely effective when applied to non-vector data (genomic contexts and genomic sequence fragments respectively). They also show that, subject to the choice of basis functions, RVM-based learning methods can be quite transparent, with models which can be viewed and related back to biological processes. Other researchers have applied exactly the same library to quite different problems. For example in [Pocock 2001], the RVM trainer was used to classify microarray gene expression data from samples taken before and after treatment of a tumour with doxyrubicin. The basis functions chosen indicated a small set of representative genes whose expression levels changed dramatically and consistently when cells were treated with this drug. This application does, however, also show off one limitation of sparse learning systems: to discover the complete set of genes implicated in the process, rather than a minimal informative subset, it is necessary to post-process the results using a clustering algorithm.