An Ensemble Approach to Building Mercer Kernels with Prior Information

Ashok N. Srivastava
Intelligent Systems Division
NASA Ames Research Center
Moffett Field, CA 94043
Email: ashok@email.arc.nasa.gov

Johann Schumann and Bernd Fischer
Research Institute for Advanced Computer Science
NASA Ames Research Center
Moffett Field, CA 94043
Email: schumann@riacs.edu, fisch@riacs.edu

***Abstract*— This paper presents a new methodology for automatic knowledge driven data mining based on the theory of Mercer Kernels, which are highly nonlinear symmetric positive definite mappings from the original image space to a very high, possibly infinite dimensional feature space. We describe a new method called Mixture Density Mercer Kernels (MDMK) to learn kernel function directly from data, rather than using pre-defined kernels. These data adaptive kernels can encode prior knowledge in the kernel using a Bayesian formulation, thus allowing for physical information to be encoded in the model. Specifically, we demonstrate the use of the algorithm in situations with extremely small samples of data. We compare the results with existing algorithms on data from the Sloan Digital Sky Survey (SDSS) and demonstrate the method's superior performance against standard methods. The results show that the Mixture Density Mercer Kernel described here outperforms tree-based classification in distinguishing high-redshift galaxies from low-redshift galaxies by approximately 16% on test data, bagged trees by approximately 7%, and bagged trees built on a *much larger* sample of data by approximately 2%. The code for these experiments has been generated with the AutoBayes tool, which automatically generates efficient and documented C/C++ code from abstract statistical model specifications. The core of the system is a schema library which contains templates for learning and knowledge discovery algorithms like different versions of EM, or numeric optimization methods like conjugate gradient methods. The template instantiation is supported by symbolic-algebraic computations, which allows AutoBayes to find closed-form solutions and, where possible, to integrate them into the code.**

I. Introduction

There is a growing interest in the machine learning and data mining communities in the field of *Mercer Kernels* due to their mathematical properties as well as their use in Support Vector Classifiers and Regressors. The theory of Mercer Kernels allows data which may be embedded in a vector space, such as spectral lines, physical measurements, stock market indices, or may not arise from a vector space, such as sequences, graphs, and trees to be treated using similar mathematics. Work by Haussler [11] shows how to map sequences of symbols into a feature space using kernel similarity measures. In the same paper, Haussler introduced the idea of a Probabilistic Kernel function, or P-Kernel that obeys Mercer's conditions (i.e., the kernel function must be symmetric and positive definite) and is defined as follows:

$$K(\mathbf{x}_i, \mathbf{x}_j) = P(\mathbf{x}_i|\Theta)P(\mathbf{x}_j|\Theta) \qquad (1)$$

This kernel function says that two points are similar if they are both more likely given the model Θ. Thus, data points lying in $\mathcal{R}^n$ which may be far away from each other in the Euclidean sense may turn out to be 'similar' as measured by this kernel. We generalize this notion of similarity using Mixture Density Mercer Kernels.

In a recent paper [16], the notion of Mixture Density Mercer Kernels was introduced. The idea is to express the distribution function $P(\mathbf{x}_i)$ in terms of a full Bayesian formulation of a density function. The kernel function is created by taking bootstrap aggregate samples models based on the distribution function. Thus, for one bootstrap sample, we have:

$$P(\mathbf{x}_i|\Theta) = \sum_{c=1}^{C} P(c)P(\mathbf{x}_i|\theta_c) \qquad (2)$$

Due to the Bayesian formulation, prior distributions can be placed on the model parameters for each bootstrap sample. This allows us to encode domain knowledge into each model. The kernel function is then composed of the sum of the outer products of the class membership matrices. Thus, we have:

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j) \\ &= \frac{1}{M}\sum_{m=1}^{M}\sum_{c_m=1}^{C_m} P_m(c_m|\mathbf{x}_i)P_m(c_m|\mathbf{x}_j) \qquad (3) \end{aligned}$$

where K represents the a sum of M bootstrap samples. $\Phi(\mathbf{x}_i)$ is a composite class membership function, where each member of the composite is the posterior class distribution for a model. Thus, for M models, we have:

$$\begin{aligned} \Phi(\mathbf{x}_i) \quad \propto \quad & [P_1(c=1|\mathbf{x}_i), P_1(c=2|\mathbf{x}_i), \ldots, \\ & P_1(c=C|\mathbf{x}_i), P_2(c=1|\mathbf{x}_i), \ldots, P_M(c=C|\mathbf{x}_i)] \end{aligned}$$

In the hard clustering case, where the posterior class distribution for a given model is a zero-one vector, the (i,j) element of the Mixture Density Mercer Kernel describes how many times, on average, the M models agreed that data points $\mathbf{x}_i$ and $\mathbf{x}_j$ arose from the same mode in the density function. The kernel matrix generated by this procedure is positive definite by construction.

As is the case with ensemble methods, the greater the variability in these models, the better the performance of the overall model. We demonstrate the degree of variability in the models due to different initial conditions in terms of variations in the converged likelihood value. This paper elucidates the idea and demonstrates its feasibility in working with a large astronomical data set known as the Sloan Digital Sky Survey.

The information required to construct the kernel function (i.e., the class membership matrix) can be computed by an application of the EM-algorithm [7] on a suitable training set. A number of EM-implementations is available (e.g., Autoclass [5], [6], EMMIX [13], MCLUST [9]) and any of them could be used. However, in order to insert domain knowledge into the kernel matrix, the EM-code has to be modified accordingly; this is time-consuming and error-prone. Moreover, since the choice of a particular prior has consequences for the quality of the kernel matrix, a certain amount of experimentation is necessary. However, the exact form of the prior can also have substantial consequences on the details of the implementation (e.g., the form of the M-step or the internal data structures) which magnifies the implementation problem. Fortunately, the overall structure of the algorithm remains the same and the details can be derived mechanically. Here, we have applied AUTOBAYES to produce the different variants of the EM-algorithm. AUTOBAYES is a fully automatic program synthesis system that generates efficient and documented C/C++ code from abstract statistical model specifications. It is not yet publicly available but described in more detail in [4], [8], [10].

II. NOTATION

- D is the dimension of the data, N is the number of data points $\mathbf{x}_i$ drawn from a D dimensional space
- M is the number of probabilistic models used in generating the kernel function.
- C is the number of mixture components in each probabilistic model. In principle one can use a different number of mixture components in each model. However, here we choose a fixed number for simplicity.
- $\mathbf{x}_i$ is a $p \times 1$ dimensional real column vector that represents the data sampled from a data set $\mathcal{X}$.
- $\Phi(\mathbf{x}) : \mathcal{R}^p \mapsto \mathcal{F}$ is generally a nonlinear mapping to a high, possibly infinite dimensional, feature space $\mathcal{F}$. This mapping operator may be explicitly defined or may be implicitly defined via a kernel function.
- $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)\Phi^T(\mathbf{x}_j) \in \mathcal{R}$ is the kernel function that measures the similarity between data points $\mathbf{x}_i$ and $\mathbf{x}_j$. If K is a Mercer kernel, it can be written as the outer product of the map Φ. As i and j sweep through the N data points, it generates an $N \times N$ kernel matrix.
- ρ_c is the mixture weight for the c th mixture, and $q(i,c)$ is the posterior probability of class membership, i.e., $q(i,c) = P(c|\mathbf{x}_i)$
- $\Theta = (\rho, \mu, \sigma)$ is the entire set of parameters that specify a mixture model.

III. KERNELS BUILT FROM ENSEMBLES

This section overviews two methods that we have created to build Mercer Kernels directly from data. The first method, which we call Mixture Density Mercer Kernels (MDMK) uses an ensemble of probabilistic models to create a Mercer Kernel. The second method, which we call the Bagged Tree Kernel (BTK), uses an ensemble of decision trees to create a kernel matrix. We begin with a brief discussion of the MDMK followed by a synopsis of the BTK.

A. Mixture Density Mercer Kernels

The Mixture Density Mercer Kernel function given in (3) is similar to the Cluster-based Similarity Partitioning Algorithm (CSPA) discussed in [1]. While their implementation uses hard clustering, it can be extended to the soft clustering (expectation-maximization) approach described here. An important difference between this work and previous work, however, is that we intend to use our kernel function in support vector machines for classification and regression. The modularity of SVMs allow different kernels to be implemented that model the underlying data generating process in different ways.

The Mixture Density Mercer Kernel is built using an ensemble of Bayesian mixture density models. The Bayesian formulation allows for prior information to be encoded in the model. Then, rather than computing a maximum-likelihood estimator, we compute a maximum a posteriori estimator which includes the likelihood function and the prior. The greater the heterogeneity of the models used in generating the kernel, the more effective the procedure. In the AUTOBAYES implementation of the procedure, the training data is sampled M times with replacement. These overlapping data sets, combined with random initial conditions for the EM-algorithm, aid in generating a heterogenous ensemble.

In this work, we assume a Gaussian distribution as the model for each class, with priors expressed in terms of a conjugate prior for the Gaussian distribution. A conjugate prior is defined as a family F of probability density functions such that for every $f \in F$, the posterior $f(\Theta|\mathbf{x})$ also belongs to F.

For a mixture of Gaussians model, priors can be set as follows [3]. For priors on the means, either a uniform distribution or a Gaussian distribution can be used. For priors on the covariance matrices, the Wishart density can be used: $P(\mathbf{\Sigma}_i|\alpha,\beta,\mathbf{J}) \propto |\mathbf{\Sigma}_i^{-1}|^{\frac{\beta}{2}} \exp(-\alpha tr(\mathbf{\Sigma}_i^{-1}\mathbf{J})/2)$. For priors on the mixture weights, a Dirichlet distribution can be used: $P(p_i|\gamma) \propto \prod_{c=1}^{C} p_i^{\gamma_i - 1}$, where $p_i \equiv P(c = i)$. Maximum a posteriori estimation is performed by taking the log of the posterior likelihood of each data point $\mathbf{x}_i$ given the model Θ. The following function is thus optimized using the EM-algorithm [7] for a Gaussian mixture model with priors on the means only.

For example, the log posterior probability

$$\log(P(\mu, x \mid c, \sigma^2) \times P(c \mid \rho))$$

for a model with conjugate priors on the means is thus computed as follows. The first step is to marginalize (i.e., sum out) the latent variable c via the expectation q. However, to keep this step tractable, it is important to delay the actual summation as long as possible. We thus introduce the "delayed summation" operator $\sum_{\text{dom}\ c_i \sim q_i}^{i=1 \dots N}$ which gives us

$$\sum_{\text{dom}\ c_i \sim q_i}^{i=1 \dots N} \log P(\mu, x \mid c, \sigma^2) + \sum_{\text{dom}\ c_i \sim q_i}^{i=1 \dots N} \log P(c \mid \rho)$$

We can then apply the product rule to decompose the probabilities and then replace them by the density functions. This gives us the formidably looking log-likelihood function

$$\sum_{\text{dom}\ c_i \sim q_i}^{i=1 \dots N} \log \left(\prod_{j=1}^{D} \prod_{k=1}^{C} \frac{\exp\left(\frac{-\frac{1}{2}(\mu_{j,k} - \alpha_{j,k})^2}{(\sqrt{\sigma_{j,k}^2}\beta_{j,k})^2} \right)}{\sqrt{2\pi}\sqrt{\sigma_{j,k}^2}\beta_{j,k}} \right.$$
$$\left. \times \prod_{j=1}^{D} \prod_{k=1}^{N} \frac{\exp\left(\frac{-\frac{1}{2}(x_{j,k} - \mu_{j,c_k})^2}{\sqrt{\sigma_{j,c_k}^2}^2} \right)}{\sqrt{2\pi}\sqrt{\sigma_{j,c_k}^2}} \right)$$
$$+ \sum_{\text{dom}\ c_i \sim q_i}^{i=1 \dots N} \log \prod_{j=1}^{N} \rho_{c_j}$$

Now we can actually execute the delayed summations; the crucial step is to replace all occurrences of the latent variable inside the body of the $\sum_{\text{dom}\ c_i \sim q_i}^{i=1 \dots N}$ by appropriately re-indexed and weighted occurrences. For example, this step simplifies $\sum_{\text{dom}\ c_i \sim q_i}^{i=1 \dots N} \log \prod_{j=1}^{N} \rho_{c_j}$ into $\sum_{i=1}^{N} \log \prod_{j=1}^{N} q_{j,i} \rho_{c_i}$. After further simplifications, we then get the more tractable form:

$$-(\frac{1}{2} CD \log 2\pi + DN \log 2\pi + \sum_{i=1}^{D} \sum_{j=1}^{C} \log \beta_{i,j}$$
$$+ \sum_{i=1}^{D} \sum_{j=1}^{C} \log \sigma_{i,j}^2 + \sum_{i=1}^{D} \sum_{j=1}^{C} \frac{(\mu_{i,j} - \alpha_{i,j})^2}{\beta_{i,j}^2 \sigma_{i,j}^2}$$
$$+ \sum_{i=1}^{N} \sum_{j=1}^{C} q_{i,j} \sum_{k=1}^{D} \frac{(x_{k,i} - \mu_{k,j})^2}{\sigma_{k,j}^2} + \sum_{i=1}^{C} \sum_{j=1}^{D} \log \sigma_{j,i}^2 \sum_{j=1}^{N} q_{j,i})$$
$$+ \sum_{i=1}^{C} \log \rho_i \sum_{j=1}^{N} q_{j,i}$$

After dropping the terms in the first line (which are independent of the goal variables), and introducing the Lagrange-multiplier λ we get the following final form of the log-likelihood function.

$$-(\sum_{i=1}^{D} \sum_{j=1}^{C} \log \sigma_{i,j}^2 + \sum_{i=1}^{D} \sum_{j=1}^{C} \frac{(\mu_{i,j} - \alpha_{i,j})^2}{\beta_{i,j}^2 \sigma_{i,j}^2}$$
$$+ \sum_{i=1}^{N} \sum_{j=1}^{C} q_{i,j} \sum_{k=1}^{D} \frac{(x_{k,i} - \mu_{k,j})^2}{\sigma_{k,j}^2} + \sum_{i=1}^{C} \sum_{j=1}^{D} \log \sigma_{j,i}^2 \sum_{j=1}^{N} q_{j,i})$$
$$+ \lambda \sum_{i=1}^{C} \log \rho_i + \sum_{i=1}^{C} \log \rho_i \sum_{j=1}^{N} q_{j,i}$$

This rather onerous derivation of the likelihood function (including the LaTeX-code for the displayed formulae!) was generated fully automatically by AutoBayes. It is important to note that although this likelihood function is much more complicated than the likelihood function for a model with no priors, it does not change the underlying parametric statistical description of the data. Thus, the slightly increased computational burden is only seen at the model building stage, but not at the model evaluation stage.

The art of choosing priors is one of much study in Bayesian data analysis. As will be seen later in this work, we choose priors based on human knowledge about the domain problem as well as from various visualizations of the data that indicate where modes should be placed. In the problem of classifying low and high redshift galaxies, we only include priors on the means of the Gaussians, rather than on the mixture weights or the covariance matrices.

B. *Bagged Tree Kernels*

The Bagged Tree Kernel uses an ensemble of bagged trees to 'vote' on the most likely class distribution for a particular pair of data points $\mathbf{x}_i, \mathbf{x}_j$. Thus, in Equation 3 the class distribution $P_m(c_m|\mathbf{x}_i)$ is given by a tree function. The remaining computations are the same as for the MDMK. We introduce this kernel function as a natural extension of an ensemble of bagged trees into the support vector machines.

IV. AutoBayes

AutoBayes is a fully automatic program synthesis system for the data analysis domain. It is implemented in Prolog and comprises about 80,000 lines of documented code. From the outside, it looks similar to a compiler: it takes an abstract problem specification in the form of a statistical model and translates it into executable code. On the inside, however, it works quite different: AutoBayes first derives a customized *algorithm* implementing the model and then transforms it into optimized C/C++ *code* implementing the algorithm. The algorithm derivation or *synthesis* process—which distinguishes AutoBayes from traditional compilers—relies on the intricate interplay of three key techniques. (*i*) AutoBayes uses *Bayesian networks* (BNs) [3], [15] as a compact internal representation of the statistical models. BNs provide an efficient encoding of the joint probability distribution over all variables and thus enable replacing expensive probabilistic reasoning by faster graphical reasoning. In particular, they speed up the decomposition of a problem into statistically independent simpler

```
model mog as 'Multivar. Mix of Gaussians';

const int D := 5 as 'number of bands'
const int N as 'number of data points'
      with 1 < N;
const int C  as 'number of classes'
      with 1 < C; with C << N;

double rho(1..C) as 'class probabilities'
      with 1 = sum(_i := 1..C, rho(_i));
double mu(1..D, 1..C);
double sigma(1..D, 1..C);

output int c(1..N) as 'latent variable';
c(_) ~ discrete(rho);

data double x(1..D, 1..N);
x(_i,_j) ~ gauss(mu(_i,c(_j)),sigma(_i,c(_j)));

max pr(x | {rho, mu, sigma})
    wrt {rho, mu, sigma};
```

Fig. 1. AUTOBAYES-specification for Gaussian mixture model. Keywords are underlined.

subproblems. (*ii*) AUTOBAYES uses *program schemas* as the basic building blocks for the algorithm derivation. Schemas consist of a parameterized code fragment or template and a set of constraints which are formulated as conditions on BNs. The templates can encapsulate advanced algorithms and data structures, which lifts the abstraction level of the algorithm derivation. The constraints allow the network structure to guide the application of the schemas, which prevents a combinatorial explosion of the search space. (*iii*) AUTOBAYES contains a specialized *symbolic subsystem* which can find closed-form solutions for many problems and emerging subproblems. The combination of these techniques results in a fast synthesis process which compares in speed to the compilation of the synthesized code.

Specification Language. A *statistical model* describes the properties of the data in a fully declarative fashion: for each problem variable of interest (i.e., observation or parameter), properties and dependencies are specified via probability distributions and constraints. Figure 1 shows how the standard Gaussian mixture model with diagonal covariance matrices can be represented in AUTOBAYES's specification language. The model assumes that the data consists of N points in D dimensions such that each point belongs to one of C classes; the first few lines of the specification just declare these symbolic constants and specify the constraints on them. However, instead of drawing each point x(_i,_j) (_i, _j are index variables) from a multivariate Gaussian c(_j) with a full D×D-dimensional covariance matrix, each band _i is drawn independently from a univariate Gaussian with mean mu(_i,c(_j)) and standard deviation sigma(_i,c(_j)). The unknown distribution parameters can be different for each class and each band; hence, we declare them as matrices. The unknown assignment of the points to the distributions (i.e., classes) is represented by the latent variable c; since we are interested in the classification results as well (and not only the distribution parameters), c is declared as output. c is distributed as a discrete distribution with the relative class frequencies given by the also unknown vector rho. Since each point must belong to a class, the sum of the probabilities must be equal to one. Finally, we specify the goal inference task, maximizing the conditional probability pr(x | {rho, mu, sigma}) with respect to the parameters of interest, rho, mu, and sigma. Maximum aposteriori estimates (MAP) can be specified by adding priors to the model. Note that the model is completely declarative and does not require the user to prescibe any algorithmic aspects of the estimation program. AUTOBAYES is thus free to select any clustering algorithm that is applicable; however, users can force the derivation of specific solutions (e.g. k-means instead of EM) via command line parameters.

Bayesian Networks. A *Bayesian network* is a directed, acyclic graph whose nodes represent random variables and whose edges define probabilistic dependencies between the random variables. AUTOBAYES uses hybrid BNs with plates [3] to represent the statistical models internally. Hence, nodes can represent discrete as well as continuous random variables. Plates generalize the concept of independent and identically distributed (*i.i.d.*) random variables and "collapse" collections of independent, co-indexed random variables into graph nodes representing the non-repeated core structure; this keeps the graphs compact and the graphical reasoning routines (e.g., computing the parents, children, or Markov blanket [15] of a node) fast. Distribution and dimension information for the random variables is attached to the respective nodes and plates.

Program Schemas. A *schema* consists of a parameterized code fragment (i.e., template) and a set of constraints. The parameters are instantiated by AUTOBAYES, either directly or by calling itself recursively with a modified problem. The constraints determine whether a schema is applicable and how the parameters can be instantiated. Constraints are formulated as conditions on the Bayesian network or directly on the specified model; they include the maximization goal as special case. This allows the network structure to guide the application of the schemas and thus to constrain combinatorial explosion of the search space, even if a large number of schemas is available. Schemas are implemented as Prolog-clauses and search control is thus simply relegated to the Prolog-interpreter: schemas are tried in their textual order. This simple approach has not caused problems so far, mainly because the domain admits a natural layering which can be used to organize the schema library. The top layer comprises network decomposition schemas which try to break down the network into independent subnets, based on independence theorems for Bayesian networks. These are domain-specific divide-and-conquer schemas: the emerging subnets are fed back into the synthesis process and the resulting programs are composed to achieve a program for the original problem. AUTOBAYES is thus able to automatically synthesize larger programs by composition of different schemas. The next layer comprises more localized decomposition schemas which work on products of *i.i.d.* variables. Their application is also guided by the network structure but they require more substantial symbolic computations. The core layer of the library contains statistical algorithm schemas as for example *expectation maximization* (EM) [7], [12] and k-Means (i.e., nearest neighbor clustering); these generate the skeleton of the program. The final layer contains standard numeric optimization methods as for example the Nelder-Mead simplex method or different conjugate gradient methods.

Symbolic Subsystem. AUTOBAYES relies significantly on symbolic computations to support schema instantiation and code optimization. The core part of the symbolic subsystem implements symbolic-algebraic computations, similar to those in Mathematica [17]. It is based on the concept of term rewriting [2] and uses a small but reasonably efficient rewrite engine. Expression simplification and symbolic differentiation are implemented as sets of rewrite rules for this rewrite engine. The basic rules are straightforward; however, the presence of vectors and matrices introduce a few complications and require a careful formalization. In addition, AUTOBAYES contains a rewrite system which implements a domain-specific refinement of the standard sign abstraction where numbers are not only abstracted into *pos* and *neg* but also into *small* (i.e., $|x| < 1$) and *large*. AUTOBAYES then uses a relatively simple symbolic equation solver built on top of these rewrite systems.

Backend. The code constructed by schema instantiation and composition is represented in an imperative intermediate language. This is essentially a "sanitized" subset of C (e.g., no pointers), which is extended by a number of domain-specific constructs like vector and matrix operations, finite sums, and convergence-loops. Since straightforward schema application can produce suboptimal code, AUTOBAYES interleaves synthesis and advanced code optimization (cf. [14] for an overview). Schemas can explicitly trigger aggressive large-scale optimizations like code motion, common subexpression elimination, and memoization which can take advantage of information from the model and the synthesis process. Traditional low-level optimizations like constant propagation or loop fusion, however, are left to the compiler. In a final step, AUTOBAYES translates the intermediate code into code tailored for a specific run-time environment. Currently, AUTOBAYES includes code generators for the Octave and Matlab environments; it can also produce stand-alone C and Modula-2 code.

V. EXPERIMENTS AND RESULTS

A. Sloan Digital Sky Survey (SDSS)

Mapping the large scale structure of the universe is necessary in order to better constrain formation scenarios of structures of all scales (from galaxies to large walls) in the universe. To this end, measuring the "distances" and x-y projections on the sky of the largest number of objects possible is necessary. Thus far it has been difficult to use only broad band color data to accurately map mass on a broad range of scales. Astronomers have only been successful in doing this on small numbers of spectroscopically measured galaxies (of order 10^5). If the errors on what we call "Photometric Redshifts" can be driven sufficiently low enough we can, for the first time, use a sample of order 10^8. This three orders of magnitude improvment could have very significant implications for contemporary theories of the Universe.

SDSS photometry (five broad band filters/colors) with calculated accurate photometric redshifts is our goal. For example, the SDSS will have 10^6 (to date $\approx$ $125,000$ measured) galaxy redshifts. The next largest survey has approximately 220,000. All of the rest of the redshifts surveys do not add up to that of the 2dFGRS alone. SDSS photometry will eventually consist of 10^8 objects (53×10^6 currently). Again, no survey approaches this quantity of data. Another survey is closest with 400 million objects, but only two "colors" are measured, it is spread across entire sky, is a much shallower survey and consists mostly of stars within our own galaxy, rather than external galaxies as in the SDSS. The results from the latest methods used to attack the Photometric Redshifts in the SDSS range from root mean squared errors from 0.034 - 0.066 and show considerable variability due to sampling. To be able to map the filamentary structures in the Universe we need a significant improvement in the root mean square error of the competing methods.

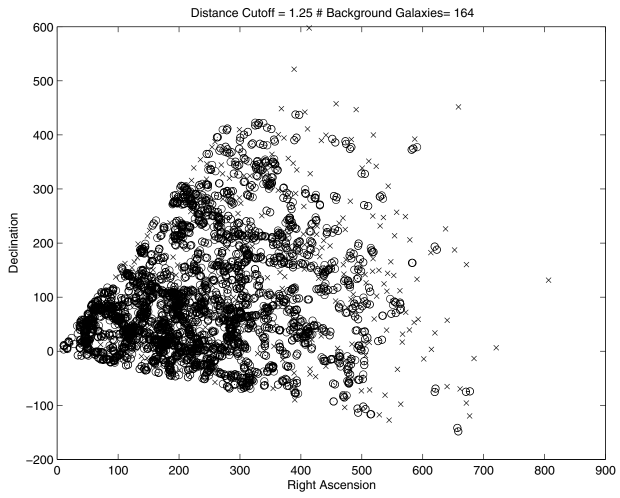

Fig. 2. Example clustering of one small section of the sky using spectroscopically determined redshifts. Crosses indicate field-galaxies, i.e., those that are not on filaments. Dots indicate galaxies that are on filaments.

B. Choosing Parameters for the Mixture Model

In a first set of experiments, clustering using AUTOBAYES generated code was used. Our model here is a multivariate mixture of Gaussians without priors. Figure 1 shows the entire AUTOBAYES specification. We ran this model and varied the desired number of classes from 3 to 30. Because our EM-algorithm uses a randomized initialization, 10 independent runs were carried out. Figure 3 shows the log-likelihood for the given parameters after clustering, the solid line shows its mean. ¿From this graph it is obvious that the initialization plays an important role as it strongly influences the result. From this figure one can also deduce that the best number of clusters for the given data set is around 12. For larger numbers of clusters, the log-likelihood does not change much, indicating that the increased model complexity does not appreciably increase the fit of the model. For each of the clustering runs, EM needed between 5 and 55 iterations, with mean of 14.2 iterations to converge.

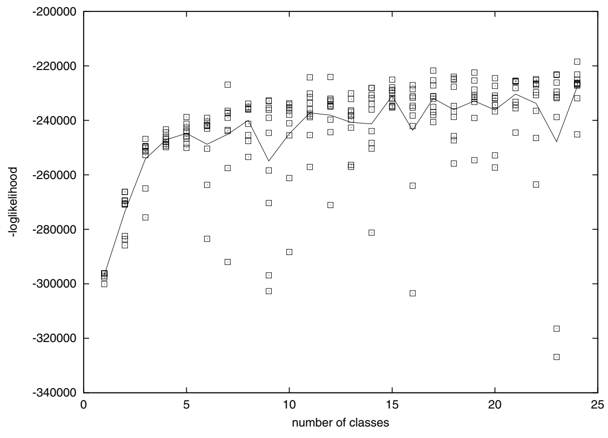

Fig. 3. Log-likelihood of a Gaussian mixture model with no priors as a function of the number of components in the model. Each box corresponds to one run. Notice that there is substantial variation in the terminal value of the likelihood function, which is due to the well-known sensitivity of the EM-algorithm to initial conditions.

The redshift of a galaxy has a strong connection on how its spectral features are mapped onto the 5 spectral bands given by the symbols (u, g, r, i, and z). If, for example, a significant spectral feature of a near galaxy shows up in band r, the corresponding feature of a similar, but distant galaxy would be shifted toward the next band, i. Thus, we can assume that the data points in the different bands are not uncorrelated (as in the previous model), but that they have correlations with the neighboring band. This extended model, which has a band covariance matrix, includes a simple transformation of the data: the new clustering algorithm gets the original 5 bands, but also the difference signal between adjacent bands: $u-g, g-r, r-i, i-z$. Figure 4 shows the results of this clustering in terms of the likelihood function. The likelihood function shows similar variation, and when penalized for the additional model complexity, also indicates that the correct number of clusters is around 12.

In the next experiment, a subset of our training data set was used. It contained all data points, for which the measured redshift was larger that 0.3. This data set contains 4530 of the 52744 data points. A similar clustering experiment (with the above AUTOBAYES model) revealed that the best number of clusters for distant galaxies is much lower (around 5). Figure 5 illustrates this.

C. Incorporating Prior Knowledge

In order to incorporate prior information in the AUTOBAYES model, we specified conjugate priors on the mean values μ. The only changes of the specification are the declarations of the prior parameters μ_0 and κ_0, their relationship to the mean, and a new optimization goal:

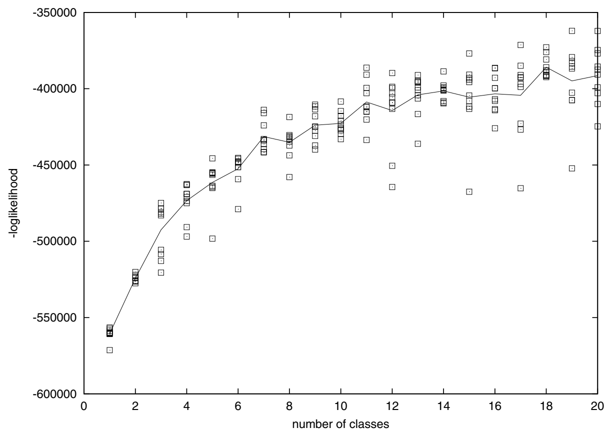

Fig. 4. Log-likelihood over number of components (shifted case)

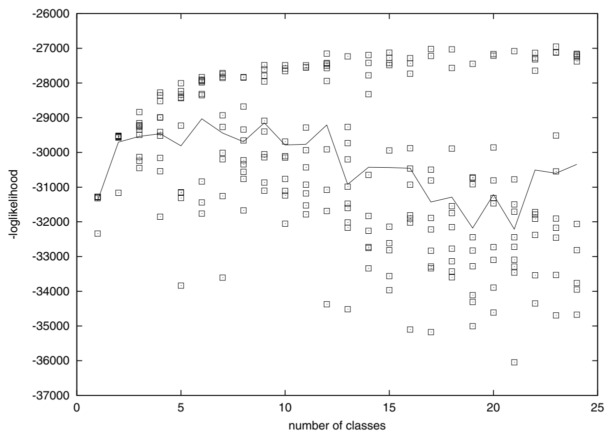

Fig. 5. Log-likelihood over number of components; distant galaxies only

```
double mu_0(0..D-1, 0..C-1);
double kappa_0(0..D-1,0..C-1);
mu(_i,_j) ~ gauss(mu_0(_i,_j),
              sqrt(sigma(_i,_j))*kappa_0(_i,_j));

max pr({mu,x} | {sigma,rho}) wrt {rho,mu,sigma};
```

The rest of the specification remains unchanged. AUTOBAYES automatically instantiates the appropriate EM-algorithm with the complex log-likelihood function given in Section 2. A visual investigation of the data displayed in Figure 6 indicated that cluster centers need to be placed in spectral regions which would model high redshift galaxies. We placed 5 clusters, based on the results from the clustering of distant galaxies only, at the spectroscopic inputs corresponding to those galaxies. Those clusters had priors associated with them on the r spectral band, since that has maximum correlation with the redshift. The remaining input dimensions had no priors. Furthermore, we did not place priors on the mixture weights or the covariance matrices. Non-isotropic diagonal covariance matrices were used in this study. With this model, we trained 20 mixture models to build the Mixture Density Mercer Kernel.

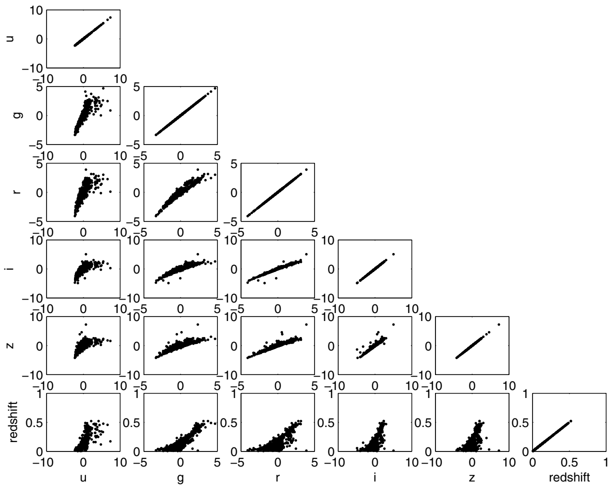

Fig. 6. This figure shows a multivariate scatter plot between the bands (u, g, r, i, z) and the redshift. Galaxies which are farther away, i.e., those with higher redshift have lower spectral energy in the band, as expected. Nearby galaxies have high spectral content. This information was used to set priors in subsequent models.

D. Evaluation of Results

The Mixture Density Mercer Kernel was built using probabilistic models that included priors as well as those without priors on a training set of 1500 galaxies and a test set of 5000 galaxies. We first submitted the MDMK along with the 5000 test galaxies to a single CART decision tree module available in Matlab. The resulting confusion matrix indicated that only 77% of the distant galaxies (those with a redshift greater that 0.3) were classified correctly. Thus, the model had a true positive rate of 77%. Using the MDMK, this rate was dramatically improved to approximately 93% using the same training and test data. The tree and the MDMK classified approximately 99% and 97% of the nearby galaxies correctly. This however, is an easy problem since nearby galaxies may have high spectral energy content, whereas distant galaxies never have high spectral content. It is much more difficult to distinguish far galaxies from those that are dim and near.

The MDMK performed significantly better than the benchmark classifier that we used regardless of the use of prior information. It turned out that in this application, prior information only improved the results of the false negative rate by about 1%, which is within the variation due to the model uncertainty. Subsequent research into the specific location of the priors and the shape of the covariance matrices will be performed.

In order to further test the quality of the SVM based on MDMK, we built an ensemble of 20 bagged trees that were built on bootstrap replicates drawn from the training set. For this scenario, we found that the true positive rate increased from 77% for the single tree to 86%. The true negative rate remained the same. However the true positive rate, although appreciably higher, was still lower than the result for the MDMK SVM.

The next experiment that we performed increased the training population for the bagged trees from the original scenario, where we were drawing bootstrap samples from 1500 points to 45,000 points, representing a 30 fold increase in the amount of training data. This dramatic increase in training data helped the bagged trees true positive rate, bringing it to approximately 91%, still 2% lower than the result for the MDMK SVM, *which was built from a data set 30 times smaller in size, thus requiring less computation time.* Note that for all experiments described here, we report the best results out of several runs for all models. Note that the performance increase for the bagged trees with training set size is likely due to the fact that the data set size is much larger than the training set size, and that the number of trees used was relatively small. Figure 8 shows the degree of variation for the models for different training sets.

The significant increase in classification accuracy can be attributed to the structure induced in the kernel matrix by the mixture modelling process. We computed a kernel matrix using the procedure outlined in this paper, and evaluated the matrix entries using a data set that was sorted in increasing order of redshift. The resulting matrix clearly shows that high redshift galaxies are generally not confused with lower redshift galaxies by the model. There are two notable exceptions in the matrix. Confusion would be indicated by large off diagonal elements in the matrix. Note that we have displayed the kernel matrix in sorted order only for illustrative purposes. The support vector machine's classification accuracy is independent of the order in which the data is presented; the underlying mathematics is invariant subject to the permutation of the data and the corresponding kernel values.

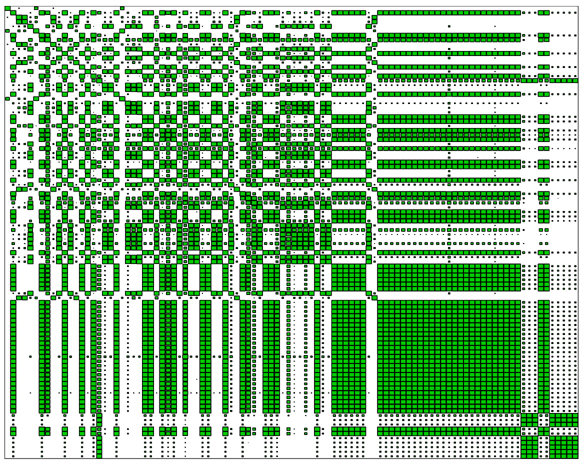

Fig. 7. This figure illustrates the reason that the Mixture Density Mercer Kernel performs so well on the classification task of identifying nearby galaxies from those that are far away. Galaxies that are far away are bunched together in the lower right hand corner of the matrix.

We ran the Mixture Density Mercer Kernels in a support vector regression machine in order to directly estimate redshift. For this problem, regardless of the use of priors, we were able to obtain a root mean squared error of approximately

0.057 on test data, which is comparable to the error rates of published methods on large samples. This data shows a great deal of sample-to-sample variability. A CART regression tree realized a root mean squared error of approximately 0.045, which is about 10% better than the MDMK described here. These results, however, are not surprising since the MDMK is built to have high performance on classification tasks.

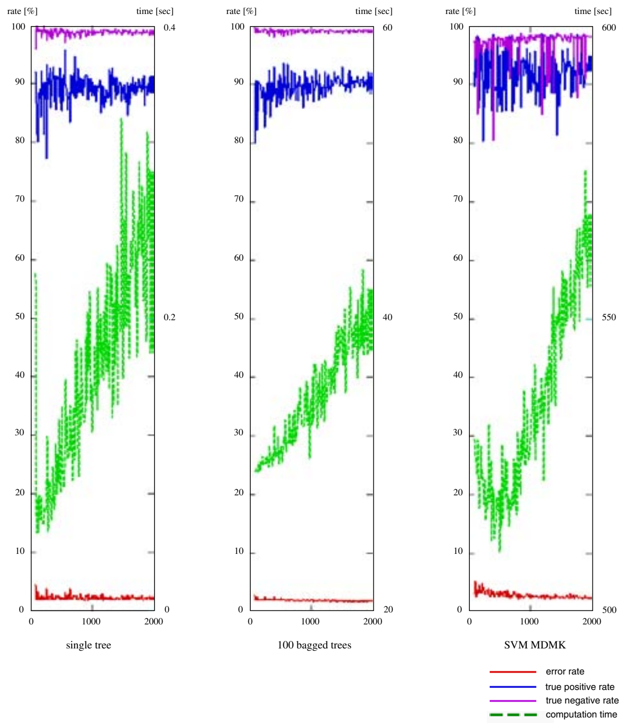

Fig. 8. Comparison of the error rates, true positive / negative rates and computation time as a function of the size of the training set. Each panel shows the results for between 100 and 2000 training points. The first panel shows the results for a single tree, the second for 100 bagged trees, and the third for the SVM built with the MDMK kernel. Notice that the true positive rates are higher for the MDMK kernel at the expense of a slightly lower true negative rate and higher computation time. We did not average the data to show the degree of variation in rates and computation times.

VI. Conclusions

Our results indicate that for a difficult, real-world classification task, the Mixture Density Mercer Kernel performs better 16% better than a decision tree. We have developed a method to incorporate prior knowledge into the model which is a novel approach to learning kernels directly from data. The MDMK with priors was built with AutoBayes, which automatically generates code to model mixture densities with prior information. The AutoBayes system generates code to model the mixture density based on high-level specifications, automatically instantiates the associated EM-algorithm schema, performs all necessary optimizations, and generates the symbolic solution along with the likelihood function.

We plan to further investigate the use of prior information in the Mixture Density Mercer Kernel framework on other real world and synthetic problems and compare it with tuned ensembles of bagged trees, where the number of trees in the ensemble is optimized. The dramatic increase in classification accuracy that is exhibited here is most likely due to the way the kernel function is constructed. The use of prior information may prove to be very useful as new understandings about the data generating process and the associate physics arise.

The results also indicate that the Mixture Density Mercer Kernel can be an excellent representation for classification problems using very small samples of data although it requires significant computational power. We plan to explore this avenue further to see how the MDMK behaves under constrained conditions. We also plan to generalize the MDMK to multiclass problems.

VII. Acknowledgements

The authors would like to thank Mike Way, Brett Zane-Ulman, Bill Macready, Nikunj Oza and the reviewers for valuable discussions and feedback. This work was supported by a grant from the NASA Intelligent Systems Intelligent Data Understanding Program.

References

[1] A. Strehl and J. Ghosh, *Cluster ensembles a knowledge reuse framework for combining multiple partitions*, JMLR **3** (2002), 583–617.

[2] F. Baader and T. Nipkow, *Term rewriting and all that*, Cambridge Univ. Press, Cambridge, 1998.

[3] W. L. Buntine, *Operations for learning with graphical models*, JAIR **2** (1994), 159–225.

[4] W.L. Buntine, B. Fischer, and T. Pressburger, *Towards automated synthesis of data mining programs*, Proc. 5th KDD (S. Chaudhuri and D. Madigan, eds.), ACM Press, August 15–18 1999, pp. 372–376.

[5] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman, *Autoclass: A bayesian classification system*, Proc. 5th Intl. Conf. Machine Learning (J. E. Laird, ed.), Morgan Kaufmann, July 1988, pp. 54–64.

[6] P. Cheeseman and J. Stutz, *Bayesian classification (AutoClass): Theory and results*, Proc. 2nd KDD (U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, eds.), AAAI Press, 1996, pp. 153–180.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin, *Maximum likelihood from incomplete data via the EM algorithm (with discussion)*, J. Royal Statistical Society B **39** (1977), 1–38.

[8] B. Fischer and J. Schumann, *AutoBayes: A system for generating data analysis programs from statistical models*, JFP **13** (2003), 483–508.

[9] C. Fraley and A. E. Raftery, *MCLUST: Software for model-based clustering, density estimation, and discriminant analysis*, Tech. Report 415, Department of Statistics, University of Washington, October 2002.

[10] A. G. Gray, B. Fischer, J. Schumann, and W. Buntine, *Automatic derivation of statistical algorithms: The EM family and beyond*, NIPS 15 (S. Becker, S. Thrun, and K. Obermayer, eds.), MIT Press, 2003, pp. 689–696.

[11] D. Haussler, *Convolution kernels on discrete structures*, Tech. report, University of California Santa Cruz, 1999.

[12] G. McLachlan and T. Krishnan, *The EM algorithm and extensions*, John Wiley & Sons, New York, 1997.

[13] G. McLachlan, D. Peel, K. E. Basford, and P. Adams, *The EMMIX software for the fitting of mixtures of normal and t-components*, J. Statistical Software **4** (1999).

[14] S. S. Muchnick, *Advanced compiler design and implementation*, Morgan Kaufmann Publishers, San Mateo, CA, USA, 1997.

[15] J. Pearl, *Probabilistic reasoning in intelligent systems: Networks of plausible inference*, Morgan Kaufmann Publishers, San Mateo, CA, USA, 1988.

[16] A. N. Srivastava, *Mixture density mercer kernels: A method to learn kernels directly from data*, Proc. 2004 SIAM Data Mining Conference (2004).

[17] S. Wolfram, *The Mathematica book*, 4th ed., Cambridge Univ. Press, Cambridge, UK, 1999.