BT* – An Advanced Algorithm for Anytime Classification

Philipp Kranen, Marwan Hassani, and Thomas Seidl

RWTH Aachen University, Germany
{lastname}@cs.rwth-aachen.de

Abstract. In many scientific disciplines experimental data is generated at high rates resulting in a continuous stream of data. Data bases of previous measurements can be used to train classifiers that categorize newly incoming data. However, the large size of the training set can yield high classification times, e.g. for approaches that rely on nearest neighbors or kernel density estimation. Anytime algorithms circumvent this problem since they can be interrupted at will while their performance increases with additional computation time. Two important quality criteria for anytime classifiers are high accuracies for arbitrary time allowances and monotonic increase of the accuracy over time. The Bayes tree has been proposed as a naive Bayesian approach to anytime classification based on kernel density estimation. However, the employed decision process often results in an oscillating accuracy performance over time. In this paper we propose the BT* method and show in extensive experiments that it outperforms previous methods in both monotonicity and anytime accuracy and yields near perfect results on a wide range of domains.

1 Introduction

Continuous experimental data in scientific laboratories constitutes a stream of data items that must be processed as they arrive. Many other real world applications can be associated with data streams as large amounts of data must be processed every day, hour, minute or even second. Examples include traffic/network data at web hosts or telecommunications companies, medical data in hospitals, statistical data in governmental institutions, sensor networks, etc. Major tasks in mining data streams are classification as well as clustering and outlier detection. Optimally, an algorithm should be able to process an object in a very short time and use any additional computation time to improve its outcome. The idea of being able to provide a result regardless of the amount of available computation time led to the development of anytime algorithms. Anytime algorithms have been proposed e.g. for Bayesian classification [19,23] or support vector machines [5], but also for anytime clustering [13] or top-k queries [2].

The Bayes tree proposed in [19] constitutes a statistical approach for stream classification. It can handle large amounts of data through its secondary storage index structure, allows for incremental learning of new training data and is

A. Ailamaki and S. Bowers (Eds.): SSDBM 2012, LNCS 7338, pp. 298–315, 2012.

capable of anytime classification. It uses a hierarchy of Gaussian mixture models, which are individually refined with respect to the object to be classified. In [14] we proposed in the MC-Tree a top down construction of the mixture hierarchy that led to an improved anytime classification performance. However, the employed decision process yields strong oscillations of the accuracy over time in several domains, which contradicts the assumption that the performance increases monotonically.

In this paper we propose BT* as an advanced algorithm for anytime classification. We investigate three methods to improve the parameters in a given Bayes tree and propose two alternative approaches for the decision design. The goals of the research presented in this paper are: *maintain the advantages* of the Bayes tree, including the applicability to large data sets (index structure) and the individual query dependent refinement, *overcome the oscillating behavior* of the anytime accuracy and achieve a monotonically increasing accuracy over time, and *increase the accuracy* of the classifier both for the ultimate decision and for arbitrary time allowances. The BT* algorithm is one approach to Bayesian anytime classification that we investigate in this paper and whose effectiveness is clearly shown in the experiments. Other solutions can be investigated, such as SPODEs (see Section 2) for categorical data, which are beyond the scope of this paper. In the following section we review related work on anytime algorithms. In Section 3 we provide details on the BT* algorithm, Section 4 contains the experimental evaluation and Section 5 concludes the paper.

2 Related Work

Anytime algorithms have first been discussed in the AI community by Thomas Dean and Mark Boddy in [4] and have thereafter been an active field of research. Recent work includes an anytime A* algorithm [16] and anytime algorithms for graph search [15]. In the data base community anytime measures for top-k algorithms have been proposed [2], in data mining anytime algorithms have been discussed for clustering [13] and other mining tasks.

Anytime classification is real time classification up to a point of interruption. In addition to high classification accuracy as in traditional classifiers, anytime classifiers have to make best use of the limited time available, and, most notably, they have to be interruptible at any given point in time. This point in time is usually not known in advance and may vary greatly. Anytime classification has for example been discussed for support vector machines [5], nearest neighbor classification [21], or Bayesian classification on categorical attributes [23]. Bayes classifiers using kernel density estimation [11] constitute a statistical approach that has been successfully used in numerous application domains. Especially for huge data sets the estimation error using kernel densities is known to be very low and even asymptotically optimal [3].

For Bayesian classification based on kernel densities an anytime algorithm called Bayes tree has been proposed in [19]. The Bayes tree maintains a hierarchy of mixture densities that are adaptively refined during query processing

to allow for anytime density estimation. An improved construction methods has been discussed in [14] that generates the hierarchy top down using expectation maximization clustering. Our proposed BT* algorithm builds upon this work.

An important topic in learning from data streams is the handling of evolving data distributions such as concept drift or novelty. A general approach to building classifiers for evolving data streams has been proposed in [22]. The main idea is to maintain a weighted ensemble of several classifiers that are build on consecutive chunks of data and are then weighted by their performance on the most recent test data. The approach is applicable to any classifier and can hence be combined with our proposed method in case of concept drift and novelty.

A different line of research focuses on anytime learning of classifiers, e.g. for Bayesian networks [17] or decision tree induction [6]. BT* allows for incremental insertion and can thereby be interrupted at will during training. Our focus in this paper is on varying time allowances during classification to allow processing newly incoming data at varying rates.

3 BT*

We start by describing the structure and workings of the Bayes tree in the following section. We recapitulate the top down build up strategy proposed in [14] along with its performance. In Section 3.2 we develop three approaches to improve the parameters in a given model hierarchy and Section 3.3 introduces two alternative decision processes for anytime classification. Finally we evaluate the proposed improvements in Section 4, individually as well as combined, to find BT* as the best performing alternative.

3.1 Anytime Bayesian classification

Let $\mathcal{L} = \{l_1, \ldots, l_{|\mathcal{L}|}\}$ be a set of class labels, $\mathcal{T}$ a training set of labeled objects and $\mathcal{T}_l \subseteq \mathcal{T}$ the set of objects with label l. A classifier assigns a label $l \in \mathcal{L}$ to an unseen object x based on its features and a set of parameter values Θ. The decision function of a Bayes classifier is generally

$$f_{Bayes}(\Theta, x) = \arg\max_{l \in \mathcal{L}} \{P(l) \cdot p(x|\Theta, l)\} \tag{1}$$

where $P(l) = |\mathcal{T}_l|/|\mathcal{T}|$ is the a priori probability of label l and $p(x|\Theta, l)$ is the class conditional density for x given label l and Θ. The class conditional density can for example be estimated using per class a unimodal distribution or a mixture of distributions. The Bayes tree, referred to as BT in the following, maintains a hierarchy of Gaussian mixture models for each label $l \in \mathcal{L}$ (see Figure 1).

Definition 1. *The model $M = \{\mathcal{N}_1, \ldots, \mathcal{N}_r\}$ of a Bayes tree is a set of connected nodes that build a hierarchy. Each node $\mathcal{N} = \{e_1, \ldots, e_s\}$ stores a set of entries with $2 \leq s \leq maxFanout$. An entry $e = \{p_e, n_e, \boldsymbol{LS}_e, \boldsymbol{SS}_e\}$ stores a pointer p_e to the first node $\mathcal{N}_e$ of its subtree, the number n_e of objects in the*

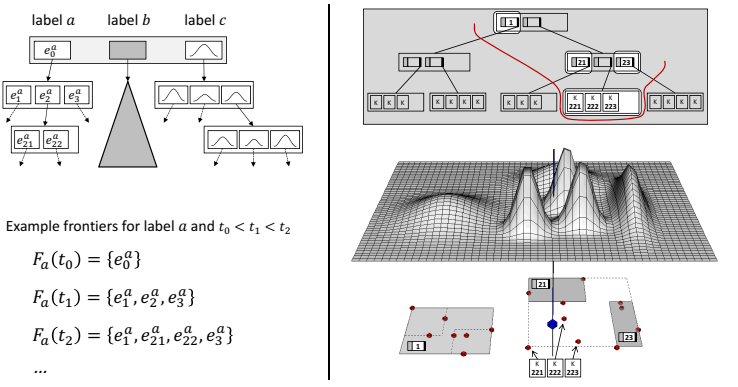


Fig. 1. The Bayes tree uses per class a hierarchy of entries that represent Gaussians (top left). For classification, the class conditional density is computed using the entries in the corresponding frontier (right). The initial frontier contains only the root entry; in each refinement one frontier entry is replaced by its child entries (bottom left).

subtree and their linear and quadratic sums per dimension. The root node $\mathcal{N}_{root}$ stores exactly one entry e_o^l for each label $l \in \mathcal{L}$ that summarizes all objects with label l. Entries in leaf nodes correspond to d-dimensional Gaussian kernels.

We refer to the set of objects stored in the subtree corresponding to entry e as $\mathcal{T}_{|e}$. Figure 1 illustrates the structure of a Bayes tree. Each entry e is associated with a Gaussian distribution

$$g(x, \mu_e, \Sigma_e) = \frac{1}{(2\Pi)^{d/2} \cdot |\Sigma_e|} \cdot e^{-\frac{1}{2}(x-\mu_e)\Sigma_e^{-1}(x-\mu_e)^T} \tag{2}$$

where μ_e is the mean, Σ_e the covariance matrix and $|\Sigma_e|$ its determinant. The parameters of the Bayes tree are

$$\Theta = \{(\mu_e, \Sigma_e) \,|\forall e \in \bigcup_{i=1}^{r} \mathcal{N}_i\} \tag{3}$$

i.e. the set of parameters for all Gaussian distributions in the tree structure. These can be easily computed from the information that is stored in the entries. The mean values can be computed as

$$\mu_e = \boldsymbol{LS}_e / n_e \tag{4}$$

Since the Bayes tree constitutes a naive Bayes approach, the covariance matrix $\Sigma_e = [\sigma_{e,ij}]$ is a diagonal matrix with $\sigma_{e,ij} = 0 \; \forall_{i \neq j}$ and

$$\sigma_{ii} = \boldsymbol{SS}_e[i]/n_e - (\boldsymbol{LS}_e[i]/n_e)^2 \tag{5}$$

where $\boldsymbol{LS}_e[i]$ is the i-th component in $\boldsymbol{LS}_e$, $i, j \in \{1, \ldots, d\}$.

To estimate the class conditional density $p(x|\Theta, l)$, the Bayes tree maintains at each time t one mixture model for each label l. The mixture model is composed

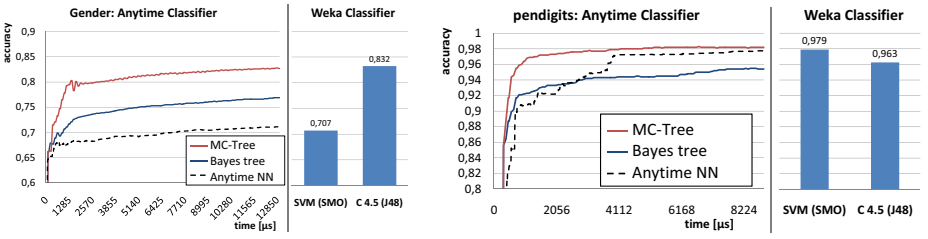


Fig. 2. MC-Tree [14] shows constantly best anytime accuracy performance against the other Bayes tree classifier variant [19], the anytime nearest neighbor [21] and the Weka classifier implementations of SVM and decision tree

of the most detailed entries that have been read up to that point in time such that each training object is represented exactly once. This set of entries is called a frontier $\mathcal{F}_l(t)$. Initially $\mathcal{F}_l(0) = \{e_0^l\}$ (see Figure 1), i.e. the initial frontier for label l contains only the root node entry $\{e_0^l\}$ corresponding to a unimodal Gaussian distribution for that class. In each improvement step one frontier is refined by

$$\mathcal{F}_l(t+1) = \mathcal{F}_l(t) \setminus \{\hat{e}\} \cup \mathcal{N}_{\hat{e}} \tag{6}$$

where $\hat{e} = \arg\max_{e' \in \mathcal{F}_l(t)} \{g(x, \mu_{e'}, \sigma_{e'})\}$ is the entry in $\mathcal{F}_l(t)$ that yields the highest density for x. To decide which frontier is refined in the next improvement, the labels are sorted according to the posterior probability with respect to the current query. In this order the top $k = \log(|\mathcal{L}|)$ frontiers are consecutively refined before resorting the labels. The decision rule of the Bayes tree at time t is then

$$f_{BT}(\Theta, x, t) = \arg\max_{l \in \mathcal{L}} \Big\{ P(l) \cdot \sum_{e \in \mathcal{F}_l(t)} \frac{n_e}{n_l} g(x, \mu_e, \sigma_e) \Big\} \tag{7}$$

where n_e/n_l is the fraction of objects from class l in the subtree corresponding to entry e. Hence, the Bayes tree can provide a classification decision at any time t and has an individual accuracy after each refinement.

The original Bayes tree builds separate hierarchies for each class label which are created using the incremental insertion from the R-tree [9]. In [14] we investigated combining multiple classes within a single distribution and exploiting the entropy information for the refinement decisions. While it turned out that separating the classes remained advisable, the novel construction method EM-TopDown proposed in [14] yielded constantly the best performance (see Fig. 2).

However, despite the improved anytime accuracy, the resulting anytime curves exhibit strong oscillations on several domains (see Figures 2 or 9), which does not constitute a robust performance. The oscillation results from alternating decisions between the individual refinement steps. With our proposed BT* algorithm we achieve near perfect results in terms of monotonicity and at the same time successfully improve the anytime accuracy performance. In our experiments we will use both the original incremental insertion (denoted as R) as well as the EM-TopDown construction (denoted as EM) as baselines for comparison. We use the

$\diamond$ operator to denote combinations of the baselines with different optimizations, e.g. $EM \diamond BN$ (see below).

3.2 Parameter Optimization

The Bayes tree determines the parameters (μ_e, Σ_e) for the Gaussians corresponding to the entries according to Equations 4 and 5. In this section we develop both discriminative and generative approaches to optimize the parameters of the Bayes tree. The first approach works on a single inner entry of the tree, the second approach processes an entire mixture model per class, and the third approach considers only leaf node entries.

BN. Our first strategy constitutes a generative approach. The goal is to fit the Gaussian distribution of an entry e better to the data in its corresponding subtree. So far the Bayes tree only considers variances and sets all covariances to zero, i.e. Σ constitutes a diagonal matrix of a naive Bayesian approach. Hence, the resulting distributions functions can only reflect axis-aligned spread of the training data. The advantage is that the space demand with respect to the dimensionality d is $O(d)$ compared to $O(d^2)$ for a full covariance matrix. Using full covariance matrices would mean that $O(d^2)$ covariances have to be stored for every single entry. Moreover, the time complexity for the evaluation of the Gaussian density function (see Equation 2) increases from $O(d)$ to $O(d^2)$. However, not all covariances might be useful or necessary. Small and insignificant correlations and corresponding rotations of the Gaussians can be neglected.

Our goal is to add important correlations at low time and space complexity. To this end we fix a maximal block size $\mathbf{s}$ and constrain Σ to have a block structure, i.e.

$$\Sigma = \mathrm{diag}(B_1, \ldots, B_u) \tag{8}$$

where $B_i \in \mathbb{R}^{\mathbf{s}_i \times \mathbf{s}_i}$ are quadratic matrices of block size $\mathbf{s}_i$ and $\mathbf{s}_i \leq \mathbf{s}\ \forall i = 1 \ldots u$. The resulting space demand is in $O(d \cdot s)$. So is the time complexity, since the exponent in the Gaussian distribution (see Equation 2) can be factorized as

$$z\Sigma^{-1}z^T = \textstyle\sum_{i=1}^{u} z[L_i..U_i]B_i^{-1}z[L_i..U_i]^T \tag{9}$$

with $z = (x - \mu)$, $L_i = 1 + \sum_{j=1}^{i-1} \mathbf{s}_j$, $U_i = L_i + \mathbf{s}_i - 1$ and $z[L_i..U_i]$ selects dimensions L_i to U_i from z.

To decide which covariances to consider and which to ignore we adapt a hill climbing method for Bayesian network learning (as e.g. proposed in [12]) that finds the most important correlations. We first describe how we derive a block structured matrix from a Bayesian network and detail the learning algorithm thereafter.

A Bayesian network $\mathcal{B} = \langle G, \Theta \rangle$ is characterized by a directed acyclic graph G and a set of parameter values Θ. In our case $G = (V, E)$ contains one vertex for the class label and one vertex $i \in V$ for each attribute. The class vertex is connected to every attribute vertex. An edge $(i, j) \in E$ between attribute i and j induces a dependency between the corresponding dimensions. We derive an undirected graph G' from G by simply removing the orientation of the edges.

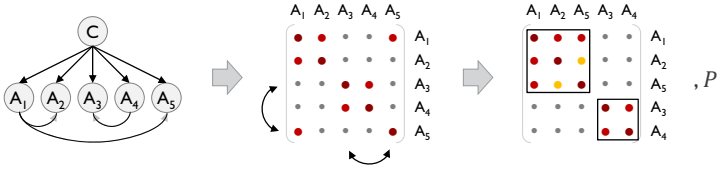


Fig. 3. An example for deriving a block structured covariance matrix (right) from a Bayesian network (left) performed on each inner entry

Edges $(i, j) \in E'$ between two attributes i and j are then transferred to a covariance σ_{ij} and its symmetric counter part σ_{ji} in Σ (see Figure 3). Since there are no constraints on the dependencies in the Bayesian network, the resulting matrix is unlikely to contain non-zero entries only in blocks along the diagonal. To ensure a block structure, we apply a permutation P to Σ by $P\Sigma P^{-1}$ that groups dimensions, which are connected in G', as blocks along the diagonal. In the resulting blocks, covariances that have not been set before are added, since these harm neither the space nor the time complexity. During classification, P is then also applied to z before computing Equation 9. Since P is just a reordering of the dimensions, it can be stored in an array of size d and its application to z is in $O(1)$ per feature.

To create the block covariance matrix for an entry e we start with a naive Bayes, i.e. initially $(i, j) \notin E' \; \forall i, j$ and $\Sigma_e = \text{diag}(\sigma_{e,11}, \ldots, \sigma_{e,dd})$. From the edges that can be added to G' without violating the maximal block constraint in the resulting block matrix, we iteratively select the one that maximizes the likelihood of e given $\mathcal{T}_{|e}$. Since all objects $x \in \mathcal{T}_{|e}$ have the same label l, the log likelihood is

$$LL(e|\mathcal{T}_{|e}) = \sum_{x \in \mathcal{T}_{|e}} \log\left(p(l, x|(\mu_e, \Sigma_e))\right) \tag{10}$$

We stop when either the resulting matrix does not allow for further additions or no additional edge improves Equation 10.

In general, we determine a block covariance matrix for each entry in the Bayes tree. However, on lower levels of the Bayes tree the combination of single components is likely to capture already the main *directions* of the data distribution. This might render the additional degrees of freedom given by the covariances useless or even harmful, since they can lead to overfitting. We therefore evaluate in Section 4 in addition to **s** the influence of restricting the BN optimization to the upper m levels of the Bayes tree.

MM. In the previously described approach we fitted a single Gaussians to the underlying training data using a generative approach. The approach we propose next considers an entire mixture model per class and tries to optimize the mixture parameters simultaneously in a discriminative way. To this and we adapt an approach for margin maximization that has been proposed in [18] (referred to as MM). It seeks to improve the classification performance of Bayesian classifiers based on Gaussian mixtures and is therefore a good starting point for the hierarchical mixtures in the Bayes tree. In the following we first describe how

we derive the mixture models per class from the tree structure and then explain the optimization procedure.

We need a heuristic to extract one mixture for each label $l \in \mathcal{L}$ from the Bayes tree. The mixture models are described by a set of entries $\mathcal{E} \subset \bigcup_{\mathcal{N}_i \in M} \mathcal{N}_i$ and we define

$$\Theta(\mathcal{E}) = \bigcup_{e \in \mathcal{E}} \{\mu_e, \Sigma_e\} \tag{11}$$

as the corresponding set of parameter values. Initially we set $\mathcal{E}_0 = \mathcal{N}_{root}$, i.e. for each label $l \in \mathcal{L}$ the unimodal model describing $\mathcal{T}_l$ is represented by $\Theta(\mathcal{E}_0)$. After optimizing the parameters in $\Theta(\mathcal{E}_0)$ we update the corresponding parameters in the Bayes tree (see Figure 4). In subsequent steps we descend the Bayes tree and set

$$\mathcal{E}_{i+1} = \bigcup_{e \in \mathcal{E}_i} \begin{cases} \mathcal{N}_e & \text{if } p_e \neq \text{null} \\ \{e\} & \text{otherwise.} \end{cases} \tag{12}$$

As above, the sets $\mathcal{E}_i$ are optimized and the Bayes tree parameters are updated. Similar to the previously described BN approach we test as an additional parameter the maximal number m of steps taken in Equation 12 in our evaluation in Section 4. For example, for $m = 2$ we only optimize the upper two levels of the Bayes tree.

We explain the MM approach for a given set of entries $\mathcal{E}$ representing one mixture model for each label $l \in \mathcal{L}$. Let $\Theta = \Theta(\mathcal{E})$ be the corresponding set of parameter values and $l_x \in \mathcal{L}$ the label of an object x. The goal is to find parameters such that for each object $x \in \mathcal{T}$ the class conditional density $p(l_x, x|\Theta)$ of its own class is larger than the maximal class conditional density among the other labels. The ratio between the two is denoted as the multi class margin $d_\Theta(x)$:

$$d_\Theta(x) = \frac{p(l_x, x|\Theta)}{\max_{l \neq l_x} p(l, x|\Theta)} \tag{13}$$

If $d_\Theta(x) > 1$ the object is correctly classified. The optimization then strives to maximize the following global objective

$$D(\mathcal{T}|\Theta) = \prod_{x \in \mathcal{T}} \bar{h}((d_\Theta(x))^\lambda) \tag{14}$$

where the hinge function $\bar{h}(y) = \min\{2, y\}$ puts emphasis on samples with a margin $d_\Theta(x) < 2$, and samples with a large positive margin have no impact on the optimization. The optimization steps require the global objective to be differentiable. To this end the multi class margin $d_\Theta(x)$ is approximated by

$$d_\Theta(x) \approx \frac{p(l_x, x|\Theta)}{\left[\sum_{l \neq l_x} p(l, x|\Theta)^\kappa\right]^{1/\kappa}} \tag{15}$$

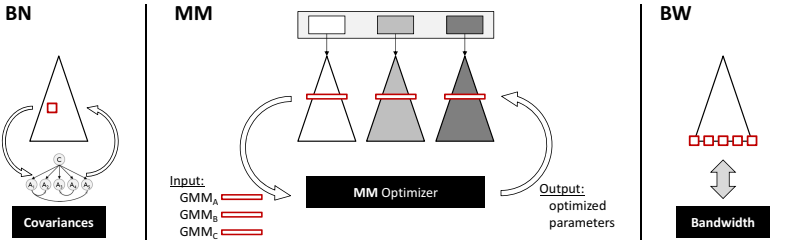


Fig. 4. Left: adding selected covariances individually to inner entries. Center: discriminative parameter optimization for entire mixture models using the margin maximization concept (MM). Right: changing the bandwidth parameter for leaf entries.

using $\kappa \geq 1$. In the global objective the hinge function is approximated by a smooth hinge function $h(y)$ that allows to compute the derivative $\partial \log D(\mathcal{T}|\Theta)/\partial\Theta$:

$$h(y) = \begin{cases} y + \frac{1}{2} & \text{if } y \leq 1 \\ 2 - \frac{1}{2}(y-2)^2 & \text{if } 1 < y < 2 \\ 2 & \text{if } y \geq 2 \end{cases} \tag{16}$$

The derivatives with respect to the single model parameters $\theta_i \in \Theta$ are then used in an extended Baum-Welch algorithm [8] to iteratively adjust the weights, means and variances of the Gaussians. Details for the single derivations as well as the implementation of the extended Baum Welch can be found in [18].

BW. In its leaves the Bayes tree stores d-dimensional Gaussian kernels (see Definition 1). The kernel bandwidth h_i, $i = 1 \ldots d$, is a parameter that is chosen per dimension and that can be optimized by different methods for bandwidth estimation. A method discussed in [20,10] uses

$$h_i = \sqrt[d+4]{\frac{4}{(d+2)\cdot|\mathcal{T}|}} \cdot \widehat{\sigma}_i \tag{17}$$

where $\widehat{\sigma}_i$ is the variance of the training data $\mathcal{T}$ in dimension i. A second method [11] determines the bandwidth as

$$h_i = \frac{max_i - min_i}{\sqrt{|\mathcal{T}|}} \tag{18}$$

where max_i and min_i are the maximal and minimal values occuring in $\mathcal{T}$ in dimension i. Additionally we test a family of bandwidths

$$h_i = \alpha \cdot \widehat{\sigma}_i \tag{19}$$

in Section 4, where a factor α is multiplied to $\widehat{\sigma}_i$. We refer to the three methods in Equations 17, 18 and 19 as *haerdle*, *langley* and $f\alpha$ respectively.

3.3 Decision Design

In the previous section we discussed different approaches to optimize distribution parameters in a given Bayes Tree. In this section we investigate alternatives for making decisions over time given an optimized tree structure. The original decision function for the Bayes tree is given in Equation 7. To estimate the class conditional density for a class l at time t the entries that are stored in the current frontier $\mathcal{F}_l(t)$ are evaluated and summed up according to their weight. We propose two approaches that both change the set of entries that are taken into consideration for the classification decision.

ENS. The first approach constitutes a special kind of ensemble methods. Ensembles are frequently used both for a single paradigm, as e.g. in Random Forests where multiple decision trees are created and employed, or for several different paradigms. The basic concept of ensembles is simple and can easily be transferred to any classification method. A straightforward way for the Bayes tree would be to build several tree structures, e.g. using different samples of the training data, and combine the individual outcomes to achieve a classification decision. The method we propose here uses a single Bayes tree and builds an ensemble over time.

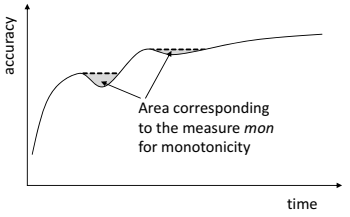
In the Bayes tree so far only the most recent frontiers $\mathcal{F}_l(t)$ were used in the decision function. To create an ensemble over time using the Bayes tree, we combine all previous frontiers in the modified decision function

$$f_{BT\diamond ENS}(\Theta, x, t) = \arg\max_{l \in \mathcal{L}} \left\{ P(l) \cdot \sum_{s=0}^{t} \sum_{e \in \mathcal{F}_l(t-s)} \frac{n_e}{n_l} g(x, \mu_e, \sigma_e) \right\} \tag{20}$$

The additional computational cost when using the ensemble decision from Equation 20 compared to the original decision from Equation 7 is only a single operation per class. More precisely, we just have to add the most recent density, which we compute also in the original Bayes tree, to an aggregate of the previous densities. Since we sum up the same amount of frontiers for each label, we can skip the normalization without changing the decision and do not have to account for additional operations. The ensemble approach widens the basis on which we make our decision in the sense that it takes mixture densities of different granularities into account for the classification decision. The approach we propose next takes the opposite direction in the sense that it narrows the set of Gaussian components that are used in the decision process.

NN. The NN approach is inspired by the nearest neighbor classifier. The nearest neighbor classifier finds a decision based on the object that is the closest to the query object x, i.e. it selects only the most promising object from $\mathcal{T}$. This concept can be transferred to the Bayes tree in a straightforward way by using the modified decision function

$$f_{BT\diamond NN}(\Theta, x, t) = \arg\max_{l \in \mathcal{L}} \left\{ P(l) \cdot \max_{e \in \mathcal{F}_l(t)} \left\{ \frac{n_e}{n_l} g(x, \mu_e, \sigma_e) \right\} \right\} \tag{21}$$



name	#obj	d	$\|\mathcal{L}\|$	name	#obj	d	$\|\mathcal{L}\|$
page-blocks	5473	10	5	pendigits	10992	16	10
optdigits	5620	64	10	vowel	990	10	11
letter	20000	16	26	spambase	4601	57	2
segment	2310	19	7	gender	189961	9	2
kr-vs-kp	3196	36	2	covtype	581012	10	7

Fig. 5. Left: Illustration of the monotonicity measure. The larger the area resulting from decreasing accuracy over time, the worse the monotonicity performance. Right: Data sets used for evaluation and their corresponding number of objects (#obj), dimensions (d), and classes ($|\mathcal{L}|$).

The NN approach comes at no additional cost since we replace the addition by a comparison in Equation 21. Variants of the nearest neighbor classifier use k closest objects. The actual label can then be assigned based on a simple majority voting among the neighbors or taking their distance or the prior probability of the labels into account. We test the standard nearest neighbor concept with $k = 1$ for $f_{BT \diamond NN}$ in our experiments.

4 Experiments

We evaluate our improvements over both [19] (called R) and [14] (called EM). Comparisons of the Bayes tree to anytime nearest neighbor, decision tree and SVM can be found in [14] (see Section 3.1 and Figure 2).

In classifier evaluation mostly the accuracy acc or the error rate, i.e. $1-acc$, is used as a measure. Since the Bayes tree is an anytime classifier that incrementally refines its decision, we get an individual accuracy $acc(n)$ for each number n of refinements (see for example Figure 7 right). In all experiments we report the results for the first $r = 200$ refinements. To compare the different approaches we use the average accuracy avg as well as the maximal accuracy max over all refinements. As a third objective, which penalizes descending or oscillating anytime curves, we use the monotonicity

$$mon = 1 - \frac{1}{r} \sum_{n=1}^{r} \widehat{acc}(n) - \min\{\widehat{acc}(n), acc(n)\}$$

where $\widehat{acc}(n) = \max_{1 \leq n' < n} acc(n')$ is the maximal accuracy over all $n' < n$. Figure 5 illustrates the monotonicity measure. The larger the sum of all areas resulting from decreasing anytime accuracy, the worse the monotonicity.

When choosing best results we select according to a linear combination of all three measures with equal weights. For the Bayes tree we set $maxFanout = 7$ and use the bandwidth estimation from [11] (*langley*) for the baselines. All experiments use 10-fold cross validation. For all objects in the test set we evaluate the classification decision after each improvement and report the average accuracy

	Bandwidth Estimation (BT ◊ BW)						**Bayesian Network** (BT ◊ BN)						**MaxMargin** (BT ◊ MM)					
	diff. to R baseline			**diff. to EM baseline**			**diff. to R baseline**			**diff. to EM baseline**			**diff. to R baseline**			**diff. to EM baseline**		
	avg	max	mon	avg	max	mon	avg	max	mon	avg	max	mon	avg	max	mon	avg	max	mon
page-blocks	**1.5%**	**1.1%**	**5.2%**	**0.6%**	-0.3%	**9.8%**	**0.2%**	**0.3%**	-18.0%	**0.1%**	-0.2%	-8.9%	-0.9%	-0.5%	-1.1%	-0.5%	-0.4%	**4.7%**
optdigits	**0.7%**	**0.8%**	**2.4%**	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	-1.0%	-0.3%	**8.3%**	-0.6%	-0.3%	**0.2%**
letter	**7.3%**	**9.5%**	**2.9%**	**5.5%**	**4.7%**	**3.7%**	**3.6%**	**4.0%**	-1.7%	**1.5%**	**1.5%**	-1.1%	-1.3%	**0.1%**	**0.4%**	-1.5%	-0.6%	**1.3%**
segment	**3.9%**	**2.1%**	**24.8%**	**6.2%**	**3.1%**	**23.5%**	**0.0%**	-0.3%	**10.8%**	-0.1%	-0.1%	**6.4%**	-5.8%	-2.6%	**23.9%**	-1.3%	-0.7%	**20.2%**
kr-vs-kp	0.0%	0.0%	0.0%	**1.1%**	**0.5%**	-1.4%	**0.0%**	0.0%	-0.5%	**0.0%**	0.0%	-0.1%	-0.5%	-1.1%	-2.3%	-18.7%	-13.1%	-17.4%
pendigits	**2.7%**	**2.9%**	**11.4%**	**1.1%**	**0.8%**	**3.0%**	**1.8%**	**2.0%**	-3.4%	**0.2%**	**0.1%**	-2.5%	0.0%	**0.3%**	**3.3%**	-0.6%	-0.5%	**1.1%**
vowel	**6.0%**	**4.1%**	**8.0%**	**4.9%**	**3.8%**	**6.4%**	**0.2%**	**0.9%**	-5.1%	**0.3%**	**0.1%**	-1.9%	-0.8%	-1.1%	**4.7%**	-1.8%	-1.5%	**4.4%**
spambase	**0.8%**	-0.3%	**1.2%**	0.0%	0.0%	0.0%	0.0%	0.0%	**0.7%**	0.0%	0.0%	**0.1%**	-7.1%	-5.1%	-0.2%	-6.3%	-3.1%	-1.9%
gender	**2.6%**	**3.9%**	**1.2%**	**3.5%**	**3.9%**	**1.3%**	0.0%	0.0%	0.0%	**0.0%**	0.0%	-0.2%	**1.1%**	**1.3%**	**5.9%**	-4.7%	-3.3%	**0.7%**
covtype	**3.4%**	**5.6%**	**3.0%**	**14.3%**	**12.3%**	**5.4%**	-2.4%	-2.4%	**2.3%**	-0.3%	-0.3%	**1.7%**	-	-	-	-	-	-
averages	**2.9%**	**3.0%**	**6.0%**	**3.7%**	**2.9%**	**5.1%**	**0.3%**	**0.5%**	-1.5%	**0.2%**	**0.1%**	-0.7%	-1.8%	-1.0%	**4.8%**	-4.0%	-2.6%	**1.5%**

Fig. 6. Approaches for parameter optimization.

over all folds per improvement. The employed data sets and their characteristics are listed in Figure 5 (right). They are available at [7] (and [1] for *gender*) where further details and background information can be found. We summarize the results and our findings in Section 4.5.

4.1 Parameter Optimization

Figure 6 shows the improvements in all three measures for the three proposed parameter optimization approaches. The numbers are absolute differences to the corresponding baseline method, highlighted cells indicate improved performance. The additional row contains the average values over all domains. We report the actual values of the measures for the baselines and the final BT* in Figure 9 below. In Figure 7 (right) we show the resulting anytime accuracy plots for the single approaches using the *letter* data set as an example.

The results shown for the bandwidth estimation in Figure 6 are the best results over the three heuristics from Equations 17 to 19, where we tested for the latter $\alpha \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$. In the 20 results ($EM$ and R on 10 data sets) *haerdle* and *langley* were both chosen three times, twice $f0.01$ was the best choice, and the remaining results were achieved by $f0.05$. By the optimized bandwidth estimation all accuracy values, i.e. avg and max on both R and EM, could be improved with the exception of max for EM on *page-blocks* and max for R on *spambase*. The largest improvement is achieved for the monotonicity with the single exception of EM on *kr-vs-kp*. The anytime plot in Figure 7 illustrates the good performance of $BT \diamond BW$ in all three measures.

The performance of using block covariance matrices in the $BT \diamond BN$ approach is hardly better than any of the two baselines. As above, the results shown in Figure 6 are the best among all parameter settings for $BT \diamond BN$, i.e. over all block sizes $\mathbf{s}$ and numbers of levels m (see Section 3.2). The largest improvements are achieved on the *letter* data set with $\mathbf{s} = d$ and $m = 1$. The performance gain was less for smaller block sizes (results not shown). As mentioned in Section 3.2, the additional degrees of freedom gained through the covariances seem to be useless or even harmful on lower levels of the tree: the displayed results, which

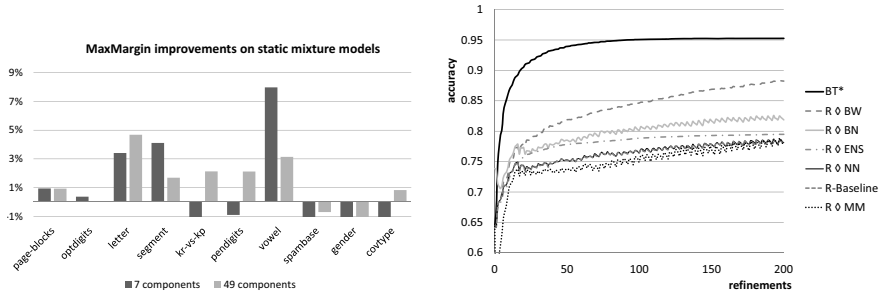


Fig. 7. Left: Results for MM on static mixture models of size 7 and 49. Right: Effects of single approaches on *letter*.

are the best among all m and $\mathbf{s}$, all use $m = 1$ and $\mathbf{s} = d$. Nonetheless, as stated above, the performance gain is only marginal on 9 of 10 data sets.

The margin maximization approach $BT \diamond MM$ does not add covariances but only seeks to improve the weights, means and variances of the Gaussians in the Bayes tree. The results on the 10 tested data sets are shown in Figure 6 (The results for *covtype* could not be achieved with 4GB RAM). As above we show the best results over all parameter settings where $\kappa \in \{1..10\}$, $\lambda \in [0, 10]$ and m from 1 to the maximal tree height. On average $BT \diamond MM$ improves the monotonicity over the baselines, but neither of the accuracy measures. The detailed results show slight improvements over the R baseline on three data sets. This result is surprising at first sight, since the original concept from [18] is designed for improving Gaussian mixture models. We discuss possible reasons below.

To exclude the possibility that the poor performance of $BT \diamond MM$ is solely due to the data set characteristics, we tested our implementation of MM on static Gaussian mixture models. Using the same expectation maximization clustering that we use for constructing the Bayes tree in the EM baseline, we created for each data set two mixtures, each contains one model per class. In the first mixture each model has 7 components, in the second 49 components per class were used. We chose multiples of 7 since it corresponds to the chosen fanout of the Bayes tree. We report the resulting absolute gain in classification accuracy from the optimized over the initial mixtures in Figure 7. MM improves the accuracy for at least one mixture on 8 data sets and for both mixtures on 5 of 10 data sets in our experiments. For the *vowel* data set the improvement is 3% for the 49 components and nearly 8% for the 7 components. However, neither avg nor max are improved by $BT \diamond MM$ on *vowel* (see Figure 6). One reason for this is the fact that $BT \diamond MM$ optimizes the mixtures in the Bayes tree level by level, but the decision f_{BT} uses arbitrary mixtures that can contain components from many different levels of the tree. These components, or rather these mixtures, were never optimized together. Optimizing all possible mixtures is not feasible, since on the one hand the sheer number of possible mixtures makes the optimization computationally infeasible, and on the other hand such an approach does not yield a single set of parameters values per Gaussian.

	Ensemble (BT ◊ ENS)						Nearest Neighbor (BT ◊ NN)					
	diff. to R baseline			diff. to EM baseline			diff. to R baseline			diff. to EM baseline		
	avg	max	mon	avg	max	mon	avg	max	mon	avg	max	mon
page-blocks	**1.2%**	**0.4%**	**10.7%**	**0.8%**	-0.2%	**9.7%**	**0.7%**	**0.5%**	**3.8%**	-46.3%	-0.8%	-47.2%
optdigits	-1.1%	-2.0%	**11.4%**	-3.2%	-3.5%	**1.1%**	**0.0%**	0.0%	**0.1%**	0.0%	0.0%	0.0%
letter	**1.7%**	**0.7%**	**3.1%**	**1.6%**	**0.0%**	**3.8%**	**0.1%**	**0.2%**	-0.3%	-0.2%	**0.3%**	**1.5%**
segment	**5.9%**	**1.5%**	**37.7%**	**4.2%**	**0.5%**	**24.3%**	**0.0%**	0.0%	-0.1%	**4.4%**	**2.4%**	**18.6%**
kr-vs-kp	-1.7%	-3.4%	**2.0%**	-2.7%	-1.6%	**0.7%**	-0.1%	-0.1%	-1.1%	**0.1%**	**0.1%**	-0.2%
pendigits	**2.0%**	**1.1%**	**12.5%**	**0.3%**	-0.1%	**3.1%**	**0.1%**	**0.0%**	**0.8%**	**0.6%**	**0.3%**	**2.7%**
vowel	**2.4%**	**0.7%**	**8.1%**	**2.8%**	**1.1%**	**4.8%**	**4.9%**	**4.0%**	**7.4%**	**3.6%**	**3.9%**	**5.4%**
spambase	**2.1%**	-0.2%	**12.9%**	-4.0%	-5.7%	-3.8%	**0.0%**	0.0%	**0.0%**	-0.2%	-0.2%	-0.6%
gender	**0.4%**	**1.5%**	**1.3%**	**2.3%**	**1.7%**	**1.4%**	-2.1%	-1.7%	-4.4%	-3.4%	-0.8%	-17.3%
covtype	**2.3%**	**3.4%**	**3.0%**	**6.2%**	**1.8%**	**5.4%**	-0.5%	-0.1%	-3.2%	-11.9%	-6.9%	-37.9%
averages	**1.5%**	**0.4%**	**10.3%**	**0.8%**	-0.6%	**5.1%**	**0.3%**	**0.3%**	**0.3%**	-5.3%	-0.2%	-7.5%

Fig. 8. Decision design approaches

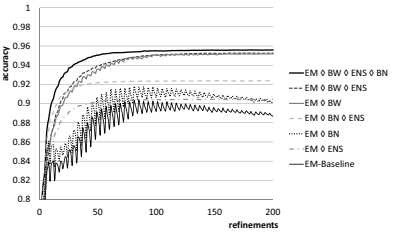
4.2 Decision Design

For the decision design we tested $f_{BT \diamond ENS}$ and $f_{BT \diamond NN}$ and show the absolute improvements for the three objectives in Figure 8. The ensemble over time in the Bayes tree (see Equation 20) yields on average a slight increase in the accuracy measures *avg* and *max* for the R baseline and is rather neutral on the EM baseline. The monotonicity, however, is drastically increased by $BT \diamond ENS$, on average by more than 10% compared to the R baseline and more than 5% compared to the EM baseline. This is underlined by the anytime accuracy plot of $BT \diamond ENS$ in Figure 7 which shows a smooth and monotonically increasing behaviour.

In contrast, the results of $BT \diamond NN$ hardly show any improvement over the R baseline except for *vowel*. The anytime plot for $BT \diamond NN$ is hardly visible in Figure 7, since it coincides with the curve of the R baseline. This is another surprising result: it indicates that taking per label only the one single Gaussian, which yields the highest class conditional density for the query object, results in almost exactly the same decisions as taking the entire mixture models. As can be seen in Figure 8, this strongly holds for 7 of the 10 tested data sets on the R baseline. Compared to the EM baseline the performance of $BT \diamond NN$ is worse on average. Summarizing the evaluation of the single approaches we can conclude that BW, ENS and BN successfully improve the anytime accuracy (see Figure 7 (right)). The first two additionally drastically improve the monotonicity on all domains.

4.3 Combining Approaches

Next we study the cumulative performance gain when improving BT by more than one concept. Figure 9 shows the anytime accuracy plots on *letter* for the EM baseline and the combined versions using ENS and/or BN and/or BW. The curve of the EM baseline exhibits a strong oscillation. $EM \diamond BN$ improves



	R baseline			EM baseline			BT*		
	avg	max	mon	avg	max	mon	avg	max	mon
page-blocks	94.0%	95.9%	87.0%	95.4%	96.8%	87.3%	**96.6%**	**96.9%**	**98.2%**
optdigits	92.9%	94.6%	87.9%	97.2%	97.8%	98.8%	96.8%	97.0%	**100.0%**
letter	76.1%	78.8%	96.8%	87.9%	90.5%	96.1%	**93.5%**	**95.3%**	**99.9%**
segment	86.3%	92.0%	59.3%	89.4%	93.6%	72.6%	**94.4%**	**95.1%**	**96.4%**
kr-vs-kp	84.4%	90.1%	96.1%	93.9%	95.2%	99.0%	**93.9%**	94.6%	**99.3%**
pendigits	93.1%	94.6%	87.3%	97.9%	98.7%	96.6%	**99.0%**	**99.4%**	**99.7%**
vowel	91.0%	94.4%	90.8%	90.8%	94.4%	92.5%	**97.4%**	**98.7%**	**99.6%**
spambase	87.4%	91.5%	81.4%	88.5%	91.4%	85.7%	**91.9%**	**92.5%**	**98.5%**
gender	73.1%	73.9%	98.4%	80.5%	81.8%	98.5%	**84.3%**	**85.7%**	**99.9%**
covtype	62.4%	63.1%	96.0%	71.4%	77.2%	94.4%	**79.8%**	**82.7%**	**99.8%**
averages	84.1%	86.9%	88.1%	89.3%	91.7%	92.2%	**92.8%**	**93.8%**	**99.1%**

Fig. 9. Left: Anytime plots for combined approaches. Right: Baseline results for R and EM and the results for the proposed BT*.

the accuracy throughout on this data set, but cannot diminish the oscillation. Near perfect monotonicity is reached when using ENS, either alone ($EM \diamond ENS$) or additionally ($EM \diamond BN \diamond ENS$). The cumulation of the two positive effects, i.e. increased accuracy and monotonicity, is clearly expressed by the corresponding anytime plots. $EM \diamond BW$ pushes up the accuracy and can also improve the monotonicity. Adding ENS yields again near perfect monotonicity (see $EM \diamond BW \diamond ENS$). Finally, combining the three concepts with EM yields the best results on this domain.

To find the globally best results we allowed all combinations of the proposed improvements and selected per data set the best setting with respect to the linear combination of all three measures. In the resulting settings ENS was used on all data sets and $f0.05$ was used eight times for BW, while other parameter optimizations were rarely employed (once BN and twice MM).

For the final setting that is used on all data sets we chose BT* $= EM \diamond f0.05 \diamond ENS$. The results are shown in Figure 9 (right) along with the two baselines R and EM. The values shown are the absolute values for the measures. Highlighted cells indicate an improvement over both baselines. On all data sets all three measures are improved by the BT* except for avg and max on optdigits and max on *kr-vs-kp*, where it shows slightly worse performance compared to EM. Overall, improving BT to BT* yields very good results on all tested data sets. Figure 10 (left) shows the anytime accuracy plots for BT* which illustrate the great performance with respect to all three measures. Figure 10 visually summarizes the average results of the single concepts and BT*, underlining the superior performance of BT*.

4.4 Scalability

To investigate the scalability of the BT* classifier we have to consider both large training data sets and large test data sets. The former affect the construction time and the storage of the data structure. For the latter, the classification time is of interest. Before we discuss these issues, we introduce three final results that are important for both training and testing. The anytime accuracy plots in

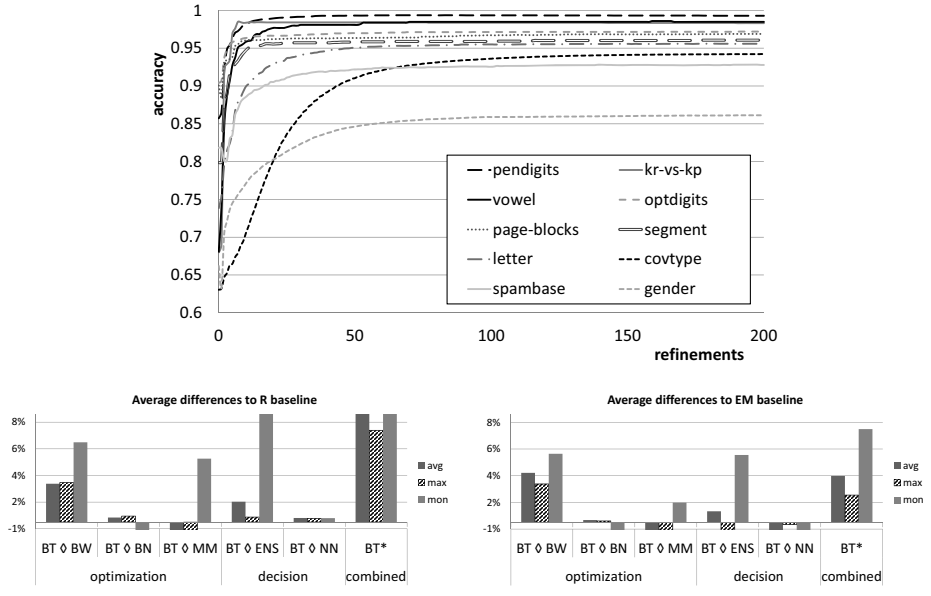


Fig. 10. Top: Anytime accuracy results for BT^*. Bottom: Average absolute differences over all 10 data sets for single and combined approaches.

Figures 7, 9 and 10 show the accuracy values for the first 200 refinements. If even more refinements are performed, the classification decision barely changes, since the density estimates change only marginally with remote mixture components being refined. Figure 11 shows for three data sets the accuracy performance for all possible refinements.

As described in the previous section, BT* uses the EMTopDown construction. For very large training data sets the complexity of the EM clustering algorithm can yield high training times. However, if the data is collected over time, BT* can be trained using the EMTopDown strategy on an initial data base and new training data can be incrementally learned in addition. This strategy can also be applied for large training data sets that are readily available. Another option in this scenario is to sample the data base and perform the EM construction on the sample. The results from Figures 10 and 11 suggest that accurate classification decisions can be achieved based on relatively small parts of the training data.

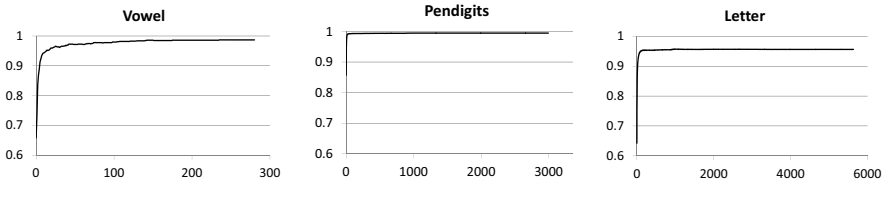


Fig. 11. Computing all possible refinements for *vowel*, *pendigits*, and *letter*

If the size of the resulting data structure exceeds the main memory, it can be stored and accessed from secondary storage. One page hosts in this scenario one node of the tree structure (as in common index structures), experiments for page accesses and different page sizes can be found in [19].

During classification, the time that is needed for a single refinement depends on the dimensionality d and on the number of previous refinements r. For every entry in the newly read node a Gaussian has to be evaluated, which is in $O(d)$. After that the entry has to be sorted into the frontier, which can be done in $O(\log_2^2(r \cdot maxFanout))$ using a heap (after r refinements the frontier contains maximally $r \cdot maxFanout$ entries). The actual times for *gender* and *pendigits* shown in Figure 2 correspond to 64 and 82 μs per refinement, respectively. From the results shown in Figures 10 and 11 we can derive that a rather small number of refinement (around 200) most often suffices to achieve a classification accuracy that is comparable to the ultimate performance. Hence, the classification time for a single object is very low even for large training data sets, which renders the BT* algorithm well suited for applications with very large test data sets.

4.5 Summary

We investigated three approaches for parameter optimization and two novel methods for the decision design. The margin maximization concept MM and the nearest neighbor like decision method NN both yielded only marginal improvements, if any. Adding covariances using the Bayesian network approach BN improved the performance only on very few domains and left it unchanged in most other cases. Two approaches that together significantly improved both accuracy and monotonicity are the bandwidth estimation BW and the ensemble over time ENS. Therefore we evaluated in Section 4.3 BT* $= EM \diamond f0.05 \diamond ENS$ (see Figure 10) that we suggest as the final variant of our anytime Bayesian classifier.

5 Conclusion

Applications for stream classification are numerous and anytime classifiers are well suited for this task since they flexibly use all available time and can provide a result after any time. Two important properties of an anytime classifier are high accuracies regardless of the available time and monotonic increase of the accuracy with additional time allowance. In this paper we have significantly improved both the monotonicity and the anytime accuracy of a recent anytime classifier. The proposed BT* algorithm achieved near perfect results on all tested data sets. It uses a special kind of ensemble that combines mixtures of different granularities resulting from the same classifier over time. The improved performance is achieved without sacrificing the time complexity, which is the same as in previous approaches. In summary, the BT* algorithm constitutes an efficient and consistent solution for anytime classification.

Acknowledgments. This work has been supported by the UMIC Research Centre, RWTH Aachen University, Germany.

References

1. Andre, D., Stone, P.: Physiological data modeling contest, ICML 2004 (2004), http://www.cs.utexas.edu/sherstov/pdmc/
2. Arai, B., Das, G., Gunopulos, D., Koudas, N.: Anytime measures for top-k algorithms on exact and fuzzy data sets. VLDB Journal 18(2), 407–427 (2009)
3. Bouckaert, R.R.: Naive Bayes Classifiers That Perform Well with Continuous Variables. In: Webb, G.I., Yu, X. (eds.) AI 2004. LNCS (LNAI), vol. 3339, pp. 1089–1094. Springer, Heidelberg (2004)
4. Dean, T., Boddy, M.S.: An analysis of time-dependent planning. In: AAAI, pp. 49–54 (1988)
5. DeCoste, D.: Anytime query-tuned kernel machines via cholesky factorization. In: Proc. of the 3rd SIAM SDM (2003)
6. Esmeir, S., Markovitch, S.: Anytime learning of anycost classifiers. Machine Learning, 25th Anniversary 82(3), 445–473 (2011)
7. Frank, A., Asuncion, A.: UCI machine learning repository (2010)
8. Gopalakrishnan, P.S., Kanevsky, D., Nadas, A., Nahamoo, D.: An inequality for rational functions with applications to some statistical estimation problems. IEEE Transactions on Information Theory 37(1), 107–113 (1991)
9. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: ACM SIGMOD, pp. 47–57 (1984)
10. Härdle, W., Müller, M.: Multivariate and semiparametric kernel regression. In: Smoothing and Regression. Wiley Interscience (1997)
11. John, G., Langley, P.: Estimating continuous distributions in bayesian classifiers. In: UAI. Morgan Kaufmann (1995)
12. Keogh, E.J., Pazzani, M.J.: Learning the structure of augmented bayesian classifiers. Intl. Journal on AI Tools 11(4), 587–601 (2002)
13. Kranen, P., Assent, I., Baldauf, C., Seidl, T.: Self-adaptive anytime stream clustering. In: ICDM, pp. 249–258 (2009)
14. Kranen, P., Günnemann, S., Fries, S., Seidl, T.: MC-Tree: Improving Bayesian Anytime Classification. In: Gertz, M., Ludäscher, B. (eds.) SSDBM 2010. LNCS, vol. 6187, pp. 252–269. Springer, Heidelberg (2010)
15. Likhachev, M., Ferguson, D., Gordon, G.J., Stentz, A., Thrun, S.: Anytime search in dynamic graphs. Artificial Intelligence 172(14), 1613–1643 (2008)
16. Likhachev, M., Gordon, G.J., Thrun, S.: ARA*: Anytime A* with provable bounds on sub-optimality. In: NIPS (2003)
17. Liu, C.-L., Wellman, M.P.: On state-space abstraction for anytime evaluation of bayesian networks. SIGART Bulletin 7(2), 50–57 (1996)
18. Pernkopf, F., Wohlmayr, M.: Large Margin Learning of Bayesian Classifiers Based on Gaussian Mixture Models. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010. LNCS, vol. 6323, pp. 50–66. Springer, Heidelberg (2010)
19. Seidl, T., Assent, I., Kranen, P., Krieger, R., Herrmann, J.: Indexing density models for incremental learning and anytime classification on data streams. In: EDBT/ICDT, pp. 311–322 (2009)
20. Silverman, B.W.: Density Estimation for Statistics and Data Analysis. Chapman & Hall/CRC (1986)
21. Ueno, K., Xi, X., Keogh, E.J., Lee, D.-Y.: Anytime classification using the nearest neighbor algorithm with applications to stream mining. In: ICDM, pp. 623–632 (2006)
22. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: KDD, pp. 226–235 (2003)
23. Yang, Y., Webb, G.I., Korb, K.B., Ting, K.M.: Classifying under computational resource constraints: anytime classification using probabilistic estimators. Machine Learning 69(1) (2007)