MC-Tree: Improving Bayesian Anytime Classification

Philipp Kranen, Stephan Günnemann, Sergej Fries, and Thomas Seidl

Data management and data exploration group, RWTH Aachen University, Germany
{kranen,guennemann,fries,seidl}@cs.rwth-aachen.de

Abstract. In scientific databases large amounts of data are collected to create knowledge repositories for deriving new insights or planning further experiments. These databases can be used to train classifiers that later categorize new data tuples. However, the large amounts of data might yield a time consuming classification process, e.g. for nearest neighbors or kernel density estimators. Anytime classifiers bypass this drawback by being interruptible at any time while the quality of the result improves with higher time allowances. Interruptible classifiers are especially useful when newly arriving data has to be classified on demand, e.g. during a running experiment. A statistical approach to anytime classification has recently been proposed using Bayes classification on kernel density estimates.

In this paper we present a novel data structure called MC-Tree (Multi-Class Tree) that significantly improves Bayesian anytime classification. The tree stores a hierarchy of mixture densities that represent objects from several classes. Data transformations are used during tree construction to optimize the condition of the tree with respect to multiple classes. Anytime classification is achieved through novel query dependent model refinement approaches that take the entropy of the current mixture components into account. We show in experimental evaluation that the MC-Tree outperforms previous approaches in terms of anytime classification accuracy.

1 Introduction

Classification is one of the most frequently used data mining techniques in scientific processes. The classifier is trained on data repositories that often comprise large data bases. A common application is continuous online classification or monitoring, e.g. during a running experiment, using measured sensor data. Examples from chemistry, physics or biology include the monitoring of concentrations or mixing ratios, temperature, pressure, intensity, velocity (movement), and so forth. Rather than simply checking values against thresholds, classification using multiple measurements simultaneously can exploit correlations among attributes and build more sophisticated classifiers.

The data stream resulting from the consecutive measurements differs depending on the application. In mechanical engineering for example, one might be

M. Gertz and B. Ludäscher (Eds.): SSDBM 2010, LNCS 6187, pp. 252–269, 2010.

interested in measurements including pressure, temperature, etc. at a certain position of the piston of an engine. Taking these measurements at varying revolutions per minute results in a varying data stream, i.e. the data tuples arrive in varying time intervals (e.g. every 60 ms at 1000 rpm and every 20 ms at 3000 rpm). In chemistry or biology, measuring concentrations or movements is often done in fixed intervals. Hence, these processes yield constant data streams, where the time interval between two arriving data tuples is constant. Advanced sensors, however, only send measurements if the actual value changed resulting once again in varying data streams.

Algorithms, such as classification algorithms, traditionally had either no time limitation or they got a fixed time budget for a certain task. The budget was known in advance, i.e. they were tailored to the specific application. These budget algorithms can neither provide a result in less time nor exploit additional time to improve their result. In contrast, anytime algorithms can provide a result after a very short initialization, improve their result incrementally when more time is available and hold the most recent result ready at any time. In data mining anytime solutions have been proposed for many tasks such as clustering [13] and classification [6,8,21].

Anytime algorithms are the natural choice for varying data streams since they flexibly exploit all available time to improve the quality of their result. Recently it has been shown in [14] that also on constant data streams anytime classifiers can improve the classification accuracy over that of traditional budget approaches. With their superiority on varying and constant data streams, applications for anytime classifiers are numerous and range from science over industrial applications to robotics and health applications.

In this paper we propose a novel approach to Bayesian anytime classification called MC-Tree. It can provide a very fast first result after evaluating just one Gaussian normal distribution per class at the root level and it can improve the classification accuracy as long as time permits by refining its current model incrementally. On the finest level a kernel density estimator is evaluated for each object in the training set (database). In between, the MC-Tree stores a hierarchy of mixture models that allows effective and query adaptive anytime density estimation. The mixture components contain objects from several (potentially all) classes. During tree construction data transformations are used that take both the locality of the data and the class distribution into account. Our novel descent strategies, which exploit the entropy information available through the MC-Tree, achieve parallel model refinement for several classes. Our experiments confirm the effectiveness of the MC-Tree and show significant improvements over previous approaches in terms of anytime classification accuracy.

2 Related Work

Traditional classification aims at determining the class label of unknown objects based on training data. Different classification approaches are discussed in the literature including nearest neighbor classifiers [17], decision trees [18] or support

vector machines [4]. Bayes classifiers constitute a statistical approach that has been successfully used in numerous application domains [2,7]. The naive Bayes classifier uses a simple model to estimate the data distribution by assuming strong statistical independence of the dimensions. Other models do not make this strong independence assumption, but estimate the density for each class by taking the dependencies of the dimensions into account. Another approach to Bayesian classification is represented by kernel density estimation [12]. Especially for huge data sets the estimation error using kernel densities is known to be very low and even asymptotically optimal [3].

Anytime classification is 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 [23]. Anytime classification has for example been discussed for decision trees [8], support vector machines [6] or nearest neighbor classification [21]. While decision trees, and often also support vector machines (SVM), provide a classification result after very short time, classification using a nearest neighbor approach (NN) might take considerably longer, especially with growing training set size. In [21] an anytime version of the NN classifier is achieved by ordering the items of the training set during the training phase and processing the items in that fixed order during classification as long as time permits. The ordering is done based on a leave-one-out cross validation on the training data, where an item is given one point if it contributes to the correct decision. Otherwise it gets subtracted $2/(m-1)$ points, where m is the number of classes. Besides a recent approach to Bayesian anytime classification from [19], we compare our approach against the anytime NN from [21] as well as the decision tree and SVM implementation from Weka [22].

For Bayesian classification based on kernel densities an anytime algorithm called Bayes tree has been proposed in [19]. The Bayes tree is a balanced tree structure and is basically an extension of the R-tree [10]. It stores in each entry a pointer and a minimum bounding rectangle and additionally a cluster feature representing the corresponding subtree. In [19] one tree structure is build per class using the standard R-tree insert, i.e. no optimization is done with respect to overlapping or effectiveness in terms of density estimation. For anytime classification several heuristics are proposed that first have to determine a class, i.e. a tree (out of all trees), whose model is refined in the next step. As a consequence, for m classes, m steps are necessary to reach one refinement per class model.

Our novel approach takes up on the Bayes tree idea, removes its drawbacks and shows significantly better anytime classification performance. More precisely, we improve the tree construction by a top down approach that tries to optimize the construction with respect to the eventual classification task. Also, we combine all classes in one Multi-Class tree whereby we only need one step to refine all class models simultaneously. Finally we introduce novel improvement strategies that exploit the MC-tree structure by incorporating the class distribution of the mixture components into the descent decision.

3 The MC-Tree

Our Multi-Class Tree (MC-Tree) is a generative model that tries to find a good representation of the underlying data distribution for a classification with high accuracy. We do not analyze each class on its own, but we describe several classes simultaneously if their objects show similar characteristics. At the same time we present a data transformation approach to consider the class information. By this we reach a stronger discrimination between the classes. The structure and construction of our MC-Tree is presented in Section 3.2. Also, within the classification and refinement process, which is necessary for the anytime behavior of our algorithm, the possible mixing of classes is considered. The classification process is described in Section 3.3. Section 3.4 concludes with our novel refinement techniques for anytime classification to improve the classification accuracy.

3.1 Preliminaries

Given a set of classes C and an object x a classifier is a function G that assigns to x the class label $G(x)$. Based on a statistical model of the distribution of class labels the Bayes classifier assigns to an object the class c_i with the highest posterior probability $P(c_i|x)$. With Bayes rule it holds:

$$G(x) = \underset{c_i \in C}{argmax}\{P(c_i|x)\} = \underset{c_i \in C}{argmax}\{P(c_i) \cdot p(x|c_i)\}$$

Estimating the class-conditional density $p(x|c_i)$ is the challenging task. One method is to assume a certain distribution of the data, e.g. a unimodal Gaussian. In general this approach yields no good representation of the true distribution. An improvement of this model is realized by the mixture of densities, i.e. a combination of several probability density functions (pdfs). In our work we use Gaussian mixture densities $p(x|c_i) = \sum_{j=1}^{k} w_j \cdot g(x, \mu_j, \Sigma_j)$ with

$$g(x, \mu_j, \Sigma_j) = \frac{1}{\sqrt{(2\pi)^d \cdot det(\Sigma_j)}} e^{\left(-\frac{1}{2}(x-\mu_j)^T \Sigma_j^{-1}(x-\mu_j)\right)}$$

where μ_j is the mean of the j-th Gaussian component, w_j its weight and Σ_j its covariance matrix. For a valid model the weights w_j must sum up to 1. Another approach is kernel density estimation that defines an influence function for each object separately. The pdf is obtained by replacing the Gaussians g with the kernels of each object. We use Gaussian kernels, i.e. $K(x) = \frac{1}{(2\cdot\pi)^{d/2}} e^{-\frac{x^2}{2h_i}}$, where h_i is the bandwidth corresponding to the variance of a Gaussian component. To set the bandwidth for our kernel estimators we use a common data independent method according to [20]. Kernel density estimation is known to perform well for traditional classification [3] and anytime classification [19].

3.2 Tree Structure and Construction

The general idea of our Multi-Class Tree (MC-Tree) is to store a hierarchy of mixture densities. The hierarchical structure allows us to represent data in more

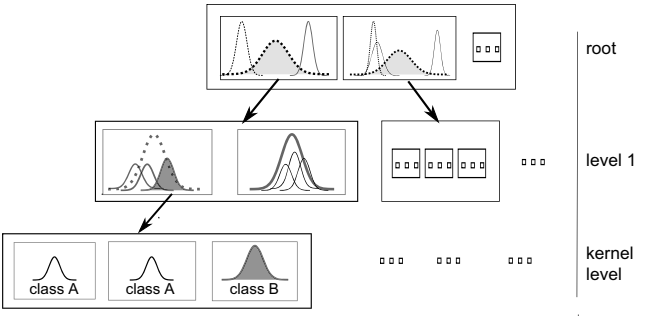


Fig. 1. Example of an MC-Tree. Gaussian components of an entry may represents entities from various classes.

or less detail. Figure 1 shows an exemplary MC-tree structure with three levels. The root node of the tree consists of two entries and represents the coarsest data model. The grey depicted Gaussians in these entries represent the aggregated statistical information of the points in the subtrees. They are constructed from the Gaussians in the level below. For better illustration these lower-level components are shown once again in the entries of the root node but are actually not stored twice in our MC-Tree. The Gaussians of the first level on their part are constructed from kernel based Gaussian components in the leaves, which represent the finest data model. As one can see at the left most entry in level 1, our MC-Tree permits to represent objects from different classes (A and B) in one Gaussian component. This is beneficial if the spatial similarity of the underlying objects is high and thus our model remains compact. Gaussians that comprise objects from several classes are represented by dotted lines in the figure.

Furthermore, unbalanced MC-Trees are possible. If an area of the data space needs a more detailed representation, paths can get longer than for other areas. Hence the level of detail in our object representation is adjusted to the actual data set.

Our MC-Tree consists of interlinked nodes, where each node contains a set of entries. We develop and analyze two types of entries for our MC-Tree. In the first version, we store the necessary information for calculating the mean and variance for each class contained in an entry separately. Thus if $O(e, c)$ is the set of objects from class c in the entry e we can derive the corresponding Gaussian. This can be done for each class in the entry individually. Furthermore we can calculate the mean and variance of the overall entry, i.e. independent to which classes the objects belong.

Definition 1. *MC-Tree node entry (type A)*
Let $O(e)$ be the objects that are represented by an entry e of the MC-Tree and $O(e, c) \subseteq O(e)$ the objects belonging to the class $c \in C$. The entry e of type A stores the following information:

- *A pointer to its subnode Sub_e (set of entries)*
- *For each class $c \in C$ a cluster feature $CF_c = (n_{e,c}, LS_{e,c}, SS_{e,c})$ representing the objects $O(e,c)$ with their number $n_{e,c} = |O(e,c)|$, their linear sum $LS_{e,c}$ and their squared sum $SS_{e,c}$*

The algebraic measures mean and variance can be calculated out of the linear and squared sums. For class c and entry e we get the mean via $\mu_{e,c} = \frac{1}{n_{e,c}} \cdot LS_{e,c}$ and the variance by $\sigma_{e,c} = \frac{1}{n_{e,c}} \cdot SS_{e,c} - (\frac{1}{n_{e,c}} \cdot LS_{e,c})^2$. Accordingly, we can calculate these values for the overall entry.

In our second approach we use a technique called variance pooling. Instead of storing the squared sum for each class we only use the squared sum of all objects in the entry. By this we assume for all classes the same variance within the entry, but we use less space. Please note that the linear sum, the number of objects and hence the mean are still used for each class on its own.

Definition 2. *MC-Tree node entry (type B)*
The entry e of type B stores the following information:

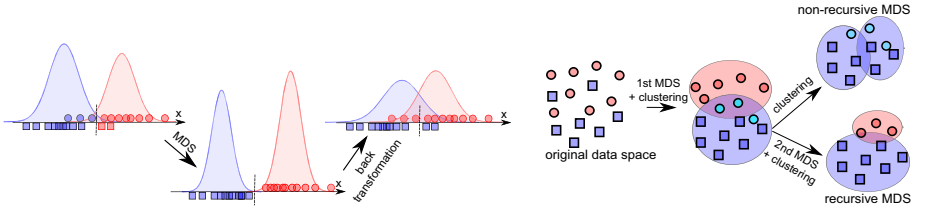
- *A pointer to its subnode Sub_e (set of entries)*
- *For each class $c \in C$ a cluster feature $CF_c = (n_{e,c}, LS_{e,c})$ representing the objects $O(e,c)$ with their number $n_{e,c} = |O(e,c)|$ and their linear sum $LS_{e,c}$*
- *The squared sum SS_e for all objects $O(e)$*

An *inner* node of our MC-Tree is a set of the beforehand introduced entries. As in [19] a *leaf* node of our tree is a set of kernels. Thus a kernel can be regarded as a special kind of entry that is only possible in leaf nodes. This special entry e represents only one object and hence $O(e)$ is just a single object. The overall definition of our MC-Tree is:

Definition 3. *MC-Tree*
Let DB be a database of objects. An MC-Tree with a fanout of θ is a tree that fulfills the following properties:

- *each inner node is a set of maximally θ entries (type A or B)*
- *each leaf node is a set of maximally θ kernels*
- *the objects $O(e)$ represented by an entry e correspond to the objects of its subnode Sub_e, i.e. $O(e) = \bigcup_{e_i \in Sub_e} O(e_i)$*
- *the entries of a single node N represent disjoint object sets, i.e. $\forall e_i, e_j \in N: \ O(e_i) \cap O(e_j) = \emptyset$*
- *the root-node R represents the whole database, i.e. $\bigcup_{e_i \in R} O(e_i) = DB$*

Tree construction. With the definition of the tree structure we know how to represent a certain subset of objects. In the next step we have to determine which objects should be grouped together to get a high classification accuracy based on our MC-Tree. We use a top-down approach to divide the whole database DB in smaller subsets S_i. For each subset one entry e_i is constructed that represents the objects S_i. All the entries together represent an inner node of the MC-Tree.



(a) Transformation for better cluster detection; Gaussians in original space

(b) Recursive vs. non-recursive MDS

Fig. 2. Multidimensional scaling (MDS) for class discrimination

The subsets are recursively divided in smaller subsets and hence further inner nodes are constructed until the kernel level is reached.

One aspect for a good partitioning of the objects is using their spatial order. The mixture of densities calculated from the entries should represent the underlying data distribution well. For dividing a set of objects in reasonable subsets we use the EM clustering algorithm [16]. The EM algorithm tries to represent objects with the best possible Gaussians. Similar to our approach it is a generative model, so we can directly use its clusters as our division of objects in entries. Because both models, our MC-Tree and the EM algorithm, use Gaussians to describe the data the clustering results of EM are well suited for our index construction. Other methods like the density-based clustering paradigm [9] optimize different criteria to obtain the clusters. Thus, a subsequent description of these clusters by Gaussians in our MC-Tree might result in poor results. The number of clusters used in the EM algorithm determines the branching factor of our MC-Tree (explicit branching factors are given in Section 4).

Using this clustering method, our MC-Tree is able to mix up several classes in one entry and we can achieve compact representations of the data. However, very impure clusters with respect to their class labels could result in low classification accuracy even if the underlying objects show a similar spatial relationship. In an impure cluster we cannot discriminate between the classes and thus a precise class prediction is problematic. Hence its preferable to find pure clusters with respect to their class labels if they also show good spatial compactness/similarity. The MC-Tree has to form a trade-off between pure entries with respect to the class labels and compact spatial clusters.

The EM algorithm does not consider the classes in the clustering process. It simply gets all objects neglecting their labels and hence pure clusters cannot be expected. To take care of this fact, objects from the same class must be considered as more similar than objects from different classes. Hence we use a new distance function that combines the spatial similarity of the objects and their class similarity:

$$d_\delta(x,y) = \begin{cases} \delta \cdot ||x-y||_2 & \text{if } class(x) \neq class(y) \\ ||x-y||_2 & else \end{cases}$$

where $||x-y||_2$ is the Euclidean distance between the objects x and y and $\delta > 1$. However, EM cannot work on arbitrary distance functions and requires a vector space.

Our approach is to transform the data space, such that the desired similarity/distance between the objects results. After the transformation the class information should also be reflected by the spatial similarity, i.e. the Euclidean distance between x and y in the new space is approximately the value $d_\delta(x, y)$. As an advantage, we still have a vector space on which EM can work but we also implicitly use the class information.

We make use of multi-dimensional scaling (MDS [5]) to transform the data space. MDS is a non-linear transformation technique for representing or visualizing objects in any d-dimensional vector space. Given the original distances between the objects, MDS iteratively tries to find a mapping into the new d-dimensional space such that the distances match best. In our technique we do not transform our objects to a lower-dimensional space but we use the same dimensionality as our original objects and change the distances to $d_\delta(x, y)$. Furthermore, the initial coordinates which are used for the MDS algorithm are the original coordinates of the objects. By this we keep to the most parts the original spatial structure of the objects and achieve by selective adjustments the consideration of the class information.

In Figure 2(a) (left) we see two classes (squares/circles) which are mixed together in the 1d space. Its not possible for the EM to identify pure clusters, i.e. both clusters contain squares and circles. The clusters are marked in red and blue, respectively. After transformation with $\delta > 1$ the situation in the middle is obtained, for which EM gets the two marked clusters. Keep in mind that the transformation is only performed for the detection of clusters, i.e. which subsets should be grouped together. The Gaussians of the corresponding entries in the MC-Tree are still calculated in the original space. An object to be classified (cf. Sec. 3.3) cannot be transformed because of its unknown class. The resulting clusters/entries are presented in Figure 2(a) (right). Both clusters represent only objects from one class. Hence, Gaussians could overlap in the original space if by this we get purer clusters.

We do not employ our MDS technique only once at the beginning of the tree construction, but recursively for each subtree. By application of MDS only for a subset of objects a better discrimination of the classes in further steps is possible. An example is depicted in Figure 2(b), where it is not possible to separate all objects from class 1 to those from class 2 with the first transformation. If we do not perform MDS recursively (right upper case) the clusters are still impure. However, if we apply MDS on the remaining smaller subset (right lower case), we can further discriminate the classes. Thus Gaussians which represent only objects from one class are more likely to show in higher levels of our MC-Tree with this recursive application.

With our method we can generate clusters and thus entries which account for the trade-off between compact spatial representations and pure cluster with

respect to the class labels. Due to the flexibility of the MC-Tree construction we can use any other method that also considers this trade-off.

3.3 Classification

Given our MC-Tree we are able to perform classification of objects with unknown labels based on the mixture of densities. We distinguish two steps in the classification process which are alternately performed. First, given a mixture of models, i.e. in our MC-Tree a set of entries, we have to determine the probability of a class a novel objects belongs to. Second, to realize the anytime property, in each step we refine our mixture to get a more fine-grained model. For this we have to replace an entry with the entries in its subtree. In the following we discuss the first step, the second step is presented in Section 3.4.

Each entry represents a Gaussian, which is associated to a set of objects. Not every set of entries is a valid mixture of densities for our classification task. We have to make sure that each object in our training set is represented by exactly one Gaussian, i.e. we need a complete model. Representing objects multiple times would favor some objects in the classification process. Objects which are not represented by Gaussians consequently are not considered for the classification. We call a set of entries $\mathcal{F} = \{e_1, \ldots, e_r\}$ a frontier if it is a complete model. Formally:

Definition 4. *Frontier*
Let $O(e)$ be the set of objects represented by the entry e and DB the whole training set. A set of entries $\mathcal{F} = \{e_1, \ldots, e_r\}$ is a frontier iff

1. $\bigcup_{i=1}^{r} O(e_i) = DB$
2. $\forall\ e_i, e_j \in \mathcal{F} :\ O(e_i) \cap O(e_j) = \emptyset$

The idea of the frontier is demonstrated in Figure 3. In this example you see a frontier (highlighted entries) which consists of two entries in the root node, one entry in the level 1 and three kernels. Obviously the set of all entries from the root-node is a valid frontier. It corresponds to the coarsest model. The same holds for the set of all leaf nodes, which is the finest model. The example in Figure 3 shows a case in between these extremes. This frontier results from refinement of the most left entry in the root node and the most left entry in the level 1 and represent a data model where one area is represented in a higher resolution. The mentioned refinement step is discussed in the Section 3.4. Given the frontier we can calculate the density of an object with respect to one class. Keep in mind that our MC-Tree can store objects from different classes in one entry. Hence, given an entry we must use only the class-specific information.

Definition 5. *Probability density query pdq in MC-Tree*
Given a frontier $\mathcal{F} = \{e_1, \ldots, e_r\}$ and a class c. Let $n_{e,c}$ be the number of objects from class c in the entry e. The pdq returns the density of an object x with respect to c and $\mathcal{F}$, i.e.

$$pdq(x|c, \mathcal{F}) = \sum_{e \in \mathcal{F}} \frac{n_{e,c}}{|DB|} \cdot g(x, \mu_{e,c}, \sigma_{e,c})$$

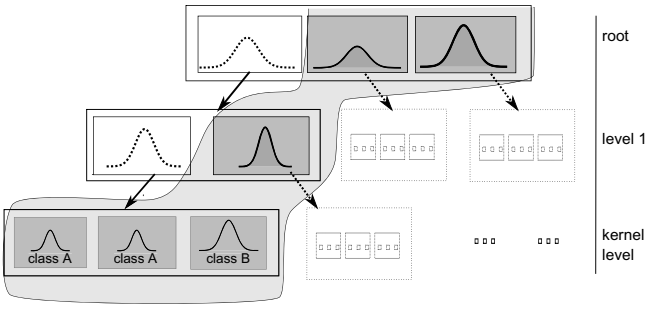


Fig. 3. Example of a frontier. The gray entries contribute to the current mixture model for the density estimation and can be refined in further steps.

where $\mu_{e,c}$ and $\sigma_{e,c}$ are calculated based on the stored information within the entries (cf. Def. 1 or Def. 2).

This is a weighted mixture of densities according to distribution of number of objects from the current class. For the leaf entries a kernel estimator is used. By our two different types of entries we have also two versions of our pdq depending on the beforehand chosen entry type.

Let n_{c_i} be the total number of objects from class c_i and let the a priori probability $P(c_i)$ be estimated from the training data as the relative frequency of each class. Then it holds:

$$\begin{aligned} pdq(x|c_i, \mathcal{F}) &= \sum_{e \in \mathcal{F}} \frac{n_{e,c_i}}{|DB|} \cdot g(x, \mu_{e,c_i}, \sigma_{e,c_i}) \\ &= \frac{n_{c_i}}{|DB|} \cdot \sum_{e \in \mathcal{F}} \frac{n_{e,c_i}}{n_{c_i}} \cdot g(x, \mu_{e,c_i}, \sigma_{e,c_i}) \\ &= P(c_i) \cdot p(x|c_i) \end{aligned}$$

The term $p(x|c_i)$ is a valid probability density function because the weights associated to the Gaussians sum up to 1. With our MC-Tree the class-specific weighting $P(c_i)$ is directly integrated in the pdq. According to Bayes classification the class label assigned to an object can be calculated by the rule:

$$\underset{c_i \in C}{argmax}\{P(c_i) \cdot p(x|c_i)\} = \underset{c_i \in C}{argmax}\{pdq(x|c_i, \mathcal{F})\}$$

3.4 Refinement

Given a frontier $\mathcal{F}$ we are able to determine the class of a new object. For anytime processing we have to generate a chain of such frontiers, starting with the coarsest model from the root, down to the leaf nodes. In each step we replace one entry $e \in \mathcal{F}$ with the entries in the subnode of e. By this we perform a refinement of the underlying spatial data distribution and hence a refinement of the probability density query pdq.

Definition 6. *Anytime pdq processing in MC-Tree*
Given a frontier $\mathcal{F}_i$. Let $e \in \mathcal{F}_i$ and $\{e_1, \ldots, e_m\}$ the entries of the subnode of e. We refine $\mathcal{F}_i$ to $\mathcal{F}_{i+1}$ by removing e and inserting the child entries:

$$\mathcal{F}_{i+1} = (\mathcal{F}_i \setminus \{e\}) \cup \{e_1, \ldots, e_m\}$$

The pdq of an object x with respect to a class c and the new frontier $\mathcal{F}_{i+1}$ is calculated by:

$$pdq(x|c, \mathcal{F}_{i+1}) = pdq(x|c, \mathcal{F}_i) - \frac{n_{e,c}}{|DB|} \cdot g(x, \mu_{e,c}, \sigma_{e,c}) + \sum_{i=1}^{m} \frac{n_{e_i,c}}{|DB|} \cdot g(x, \mu_{e_i,c}, \sigma_{e_i,c})$$

Calculating the new density for x is based on the previous result and hence the cost is low. At the beginning of the anytime processing ($i = 1$) we initialize $\mathcal{F}_1$ to the entries of the root node. From each time step i to $i + 1$ we have to decide which $e \in \mathcal{F}_i$ should be replaced. This decision is important for the performance in terms of anytime classification. If an entry with low information with respect to the current query is refined, the classification result may change only slightly but we have wasted time which could be spend for an improvement with more useful entries. We present different strategies for choosing the next entry in the following. Because we perform only local refinements during our anytime processing, it is very unlikely that we reach the leaf level for all objects in the available time. Overfitting is mitigated because we generalize huge parts of the data via aggregated information.

Quality measure. During our anytime processing the result is not only incrementally improved but our model is refined individually based on the current object to be classified. The classification is based on densities. A high density increases the probability of a class being selected. Hence a first approach, as presented in [19], is to refine the entry $e \in \mathcal{F}$ with the highest density to hopefully increase the density further. Doing this in our MC-Tree is not straightforward. Each entry subsumes several classes and densities of each class could vary. The question is, how to decide which entry is the best for refinement without favoring single classes. To make a fair selection, we use the density resulting from all objects by disregarding their class labels. The next entry to refine is defined by

$$\underset{e \in \mathcal{F}}{argmax}\{\frac{n_e}{|DB|} \cdot g(x, \mu_e, \sigma_e)\}$$

with the mean μ_e, variance σ_e and number of objects n_e based on all objects in the entry e.

Similar to the tree construction this first approach only considers the spatial order of the data. During construction we had to consider the trade-off between pure clusters with respect to their class labels and the spatial similarity of the objects. A similar observation can also used for our refinement method. Consider

an entry e with several classes that splits up in the subtree in complete pure clusters, i.e. each sub-entry of e contains only objects from one class. If we refine e we can make a clear decision in the following steps and hence our accuracy could increase. This entry e should be preferred to an entry whose sub-entries are still impure. Based on this intuition we need a measure that assesses the skew of the class label distribution in an entry and the possible gain if we refine it.

We use the well known information gain that is already used for decision trees [15,18]. The information gain for an entry e with sub-entries $\{e_1, \ldots, e_m\}$ is defined as:

$$IG(e) = entropy(e) - \sum_{i=1}^{m} \frac{n_{e_i}}{n_e} \cdot entropy(e_i)$$

where $entropy(e)$ measures the entropy of the class label distribution in e:

$$entropy(e) = \sum_{c \in C} \left(-\frac{n_{e,c}}{n_e} \cdot \log \left(\frac{n_{e,c}}{n_e} \right) \right)$$

The information gain measures the reduction of the entropy, i.e. the reduction of the class label skew, if we replace the entry e by the entries in its subnode. This information can be calculated before the query processing because it is independent of the actual query object x.

The higher the information gain the better is the refinement of e with respect to the class purity of the subtree. The higher the beforehand defined density measure the better is the refinement of e with respect to the spatial similarity. Both measures are important for our MC-Tree and hence we realize this trade-off by building a linear combination of these terms for our quality measure.

Definition 7. *Refinement quality of an entry*
Given a query x and a frontier $\mathcal{F}$. The refinement quality of an entry $e \in \mathcal{F}$ with respect to x and $\mathcal{F}$ is defined as:

$$quality_\alpha(e, x) = \alpha \cdot \frac{IG(e)}{\log |C|} + (1 - \alpha) \cdot \frac{n_e \cdot g(x, \mu_e, \sigma_e)}{\max_{w \in \mathcal{F}} n_w \cdot g(x, \mu_w, \sigma_w)}$$

We normalize both measures to the range $0 - 1$, so that a fair comparison is possible. The information gain is at most $\log |C|$ if C is the set of all possible classes in our database. The user can control the influence of both measures by changing α.

Meta strategies. Formally our quality measure defines a ranking of the entries in the current frontier. We select the first entry out of this ranking, i.e. the entry resulting out of

$$\underset{e \in \mathcal{F}}{argmax} \{quality_\alpha(e, x)\}$$

and we perform our refinement step. Afterwards the ranking is adapted based on the newly inserted entries and we can select the next best entry after updating our pdq. We call this method *first-best*, as in each step the best entry is refined.

One possible problem of this approach is that only a small local area around the query object is refined. We can stick to one path of refined entries, while other entries that could show a strong influence on the query in later steps are not refined. This problem is related to greedy algorithms which choose at each step the best local solution and hence can run into local optima and not resulting to a global optimal solution. To avoid this we present two further meta strategies for the selection of entries.

The *k-best* method is a direct extension of *first-best*. Instead of choosing the best entry we mark the k best entries in the current frontier. While there are marked entries in the frontier we select the best out of these and perform our refinement step and pdq update. Other non-marked entries are ignored even if they show better quality values. Only when all marked entries are processed for refinement, we again consider all entries in the frontier for marking the k currently best ones based on their quality measures. By this method we widen the search space because a currently second best entry is refined and can advance to a top choice for the following steps. If we set $k = 1$ we get the *first-best* method.

The *k-best* method simply chooses the k best entries based on their quality measures. The method does not consider the classes within the entries which is the important characteristic of the MC-Tree. Hence for *k-best* it is possible to select k entries all belonging to one class. This single class is favored in the classification process and all remaining classes are neglected. If the true class of the query object belongs to one of the neglected classes its unlikely to reach a correct classification. Therefore our last method tries to consider all classes equally within the refinement steps, i.e. in k successive refinement steps we favor k different classes. Similar to k-best we mark the k best entries with respect to the quality measure, but now with the additional constraint that the strongest represented class of each marked entry is always different among the k entries. This method is similar to the *qbk* heuristic, which yielded the best results in [19]. Formally we select for each class c_i the entry with the highest quality and for which c_i is also the strongest representative:

$$e^*_{c_i} = \underset{e \in \mathcal{F}}{argmax}\{quality_\alpha(e, x) \mid c_i = \underset{c \in C}{argmax}\{n_{e,c}\}\}$$

Out of the set $\{e^*_{c_i}\}_{c_i \in C}$ we mark the k best entries for the next refinement steps. Note that the strongest represented class of each entry can be calculated before the actual query processing. This method, called *k-class*, accounts for different classes in each refinement step and hence our classification decision is more likely to be changed to the correct class if it is underestimated so far.

4 Experiments

To evaluate the performance of the MC-Tree we ran experiments on different real world data sets with varying dimensionality, cardinality and number of classes. Table 1 summarizes their characteristics (Covtype = Forest Covertype). All experiments were run on Windows machines with 3 GHz and 2 GB RAM using

Table 1. Data sets used in the experiments

name	size	classes	features	ref.
Vowel	990	11	10	[11]
Pendigits	10,992	10	16	[11]
Letter	20,000	26	16	[11]
Gender	189,961	2	9	[1]
Covtype	581,012	7	10	[11]

Java 6.0. Mostly we will use an implementation invariant time measure (as used e.g. in [21]), i.e. in the graphs we report the classification accuracy over the number of Gaussians that have been evaluated. When comparing against anytime nearest neighbor [21], SVM [22] and C4.5 [22] we report the actual time on the x-axis. Please note that our goal is to improve the accuracy of anytime Bayesian classification rather than showing statistical evidence for the superiority of one or the other classifier in different domains. We perform 4-fold cross validation on the data sets and report the average accuracy value. The tree structures are constructed once using all training data of the current fold. For updates of the trees using additional new training data one can employ the incremental insertion proposed in [19] or adapt other split strategies. Since we focus on the anytime classification performance, we do not study these aspects here. In the next section we first evaluate the influence of the parameters on the anytime classification accuracy of the MC-Tree and in Section 4.2 we show the improvement of anytime Bayesian classification that is gained through our novel approach over previous results from [19].

4.1 MC-Tree Parameter Evaluation

To find a good parameter setting for the MC-Tree we start by evaluating the influence of the entropy level during descent by varying the parameter α in Figure 4 (left). We use the *k-class* strategy because in primary experiments it performs best among all meta strategies. In addition, we use zero penalty for the MDS ($\delta = 1$) and the classification decision is based on $\mu_{e,c}$ and $\sigma_{e,c}$. The maximal number of entries in a node and hence the number of clusters for the EM algorithm is set to the number of classes in the respective data set.

We find a clear winner in the results indicating that $\alpha = 0$ yields the best results. This means that the local density of the individual Gaussian components in the frontier is sufficient to best determine the next entry for a refinement. Deprecating the importance of entries that yield a high probability density with respect to the query object by promoting other entries due to their higher entropy obviously delays beneficial model refinements and thereby reduces the anytime accuracy performance. Hence we only use the entry's probability density and set α to zero in the following.

In Figure 4 (right) we evaluate the influence of the penalty for the construction using MDS on the Gender data set. The purple line corresponding to $\delta = 1$

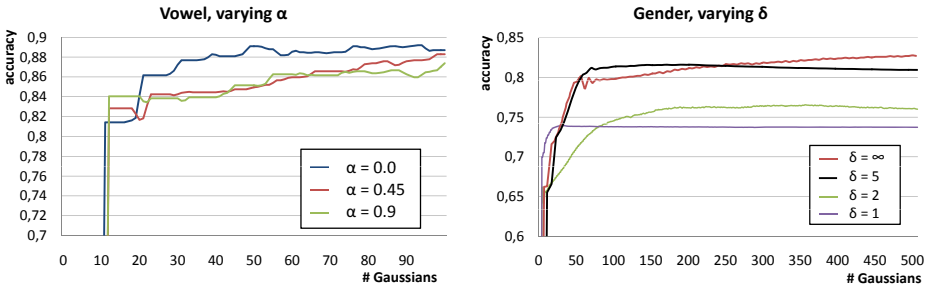


Fig. 4. Left: Varying the influence of entropy during descent via the parameter α. Right: Varying the penalty for MDS construction via the parameter δ.

(zero penalty) rises quickly to an accuracy of around 74%, but does not improve it afterwards. We found similar results, i.e. steep increase early on and little improvement in later stages, for most data sets when using the same parameter setting.

Increasing the penalty through a higher δ yields continuous improvement in terms of anytime accuracy. While the curve for $\delta = 1$ shows better performance for the first Gaussians that are read, the other settings can soon improve the accuracy significantly. The tree build with $\delta = \infty$ penalty shows an accuracy that is up to 9% higher for this data set. The superior performance for this setting was confirmed on the other data sets. Moreover, the stagnating behaviour disappeared throughout as will be shown in the next section.

$\delta = \infty$ constitutes a special case, since this transformation can be considered as dividing our database in subsets, such that all objects with the same class label are in the same subset. Afterwards we perform clustering only on the separated classes. Hence we do not need the actual transformation but can directly cluster in the original space on these subsets. This special case is more similar to the Bayes tree, but the resulting hierarchy is still very different due to the top down clustering instead of the balanced R-tree split as in [19]. We will see in the next section that the MC-tree shows significantly better anytime accuracy.

As a consequence of the high penalty and the resulting tree structure, all classification options provided by the different node types (cf. Def. 1 and 2) yield the same results and are therefore not displayed here.

4.2 Comparison: Improvement of Anytime Bayesian Classification

We compare our novel MC-Tree to a recent anytime Bayes classifier described in [19] (Bayes tree). For the Bayes tree we use the global best descent strategy and the *qbk* refinement strategy as they were reported to yield the best results. For the MC-Tree we set the fanout, i.e. the maximal number of entries per node, to the same amount as in the Bayes tree where it is dictated through the page size (2KB pages for all experiments as was used in [19]). Note that eventually the results of both approaches will be equal, i.e. if the entire leaf level has been

read the decision of both classifiers is based on the same model. However, in the graphs we focus on the interesting part, i.e. the anytime performance in the beginning of the classification process. On the right hand side of the Figures we show the corresponding accuracy reached by the SVM and C4.5 implementation from Weka [22]. These methods do not constitute an anytime approach. Clearly, for each individual application there will be a best performing classifier, but no single classification approach will be the best choice for all application domains. Our goal in this paper is to improve anytime Bayesian classification, hence we focus our comparison on the Bayes tree [19].

Figure 5 (left) shows the results for the Pendigits data set. The Bayes tree performs only slightly better than the anytime nearest neighbor in the beginning before it falls behind. The MC-tree shows a steep increase in the very beginning and outperforms the other two anytime classifiers throughout. While the accuracy of the MC-tree is similar to the nearest neighbor in later stages, it quickly shows a performance gain over the Bayes tree of three to four percent accuracy. Support vector machine (SVM) and decision tree (C4.5), which are considered very fast classifiers, perform well with 97.9% and 96.3% accuracy respectively. These approaches, however, can not improve their accuracy once they computed their result while the anytime classifiers will use additional time for further computations.

Similar results can be seen for the letter data set in Figure 5 (right). For this domain C4.5 (86.8%) performs better than the SVM (81.8%). The results of the

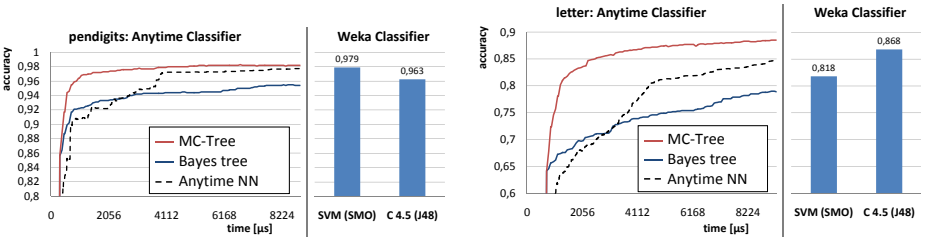


Fig. 5. Classification accuracy on pendigits (left) and letter (right) for anytime classifiers and static classifier

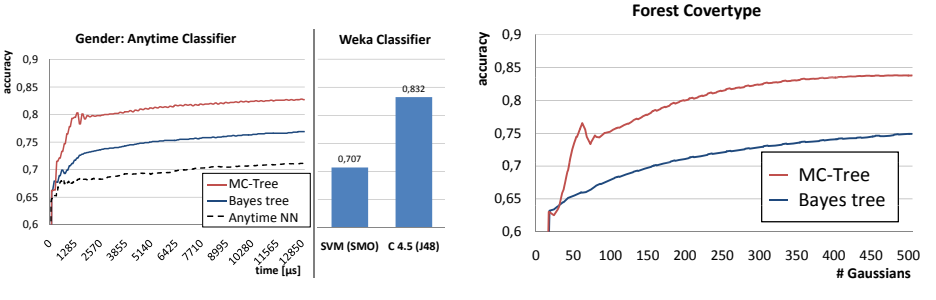


Fig. 6. Classification accuracy on gender (left) and Forest Covertype (right)

anytime NN and the Bayes tree are again similar, after a short time the nearest neighbor bypasses the accuracy of the Bayes tree. The MC-tree outperforms both approaches and reaches accuracy values which are up to 15% higher as compared to the Bayes tree constituting a major performance gain. Moreover, it soon reaches higher accuracy than both SVM and C4.5.

On the gender data set (cf. Figure 6 (left)) the Bayes tree shows constantly better performance than anytime NN. Once more, the MC-tree constantly outperforms both approaches by roughly 6% compared to the Bayes tree and 10% compared to anytime NN. As was the case for the letter data set, the decision tree (83.2%) reaches higher accuracy than the support vector machine (70.7%). This performance is once again met by the MC-tree after short time and additional time can be used for further improvement. More importantly, as was our mayor goal, the new concepts of the MC-tree prove to be effective through constantly better performance in comparison with the Bayes tree.

In Figure 6 (right) we show our results for the MC-tree and Bayes tree on the Forest Covertype data set. The results for the anytime nearest neighbor, SVM and C4.5 could not be computed due to memory issues. We report in this Figure the number of Gaussians that have been evaluated until classification. As stated above, the goal in this paper was to improve the performance of Bayesian anytime classifiers, which is again clearly reached in this experiment.

Theoretically, the accuracy curve of an anytime approach corresponds to a non-decreasing function. In Figure 6 we observe a slight up and down in the anytime curves for the MC-tree. Note that in both cases $k = 2$, i.e. the two most probable classes are refined in turns. Obviously the classification decision changes for some queries after each node that is evaluated. The amplitude of the oscillation indicates the percentage of queries behaving as just described. Those query points fall into a region of the data space where two classes overlap and where a correct decision is difficult to find. Refining the one class' model by reading an additional node increases its probability in that step, while in the next refinement the other class' model is refined (due to $k = 2$) and its corresponding probability prevails. However, the slight oscillation of the anytime curves does not diminish the dominance of the MC-Tree in terms of anytime accuracy.

5 Conclusions

Anytime algorithms and in particular anytime classification received a lot of attention over the last years. Different classification approaches are proposed in the literature which all have certain domains where they perform best. In this paper we focused on Bayesian classification and proposed a novel data structure called MC-Tree that significantly improves the anytime classification performance over previous approaches. We investigated various strategies for tree construction, descent and classification. In experimental evaluation on real world data sets we showed that the MC-Tree outperforms previous Bayesian anytime classifiers and that it improves the accuracy constantly by up to 15%.

Acknowledgments. This work has been supported by the UMIC Research Centre, RWTH Aachen University, Germany and the Federal Ministry of Economics and Technology on the basis of a decision by the German Bundestag.

References

1. Andre, D., Stone, P.: Physiological data modeling contest In: ICML 2004 (2004), http://www.cs.utexas.edu/users/pstone/workshops/2004icml/
2. Bayes, T.: An essay towards solving a problem in the doctrine of chances. Philosophical Transactions of the Royal Society 53, 370–418 (1763)
3. Bouckaert, R.: Naive Bayes Classifiers that Perform Well with Continuous Variables. In: AI (2004)
4. Burges, C.J.C.: A tutorial on support vector machines for pattern recognition. DMKD Journal 2(2), 121–167 (1998)
5. de Leeuw, J.: Applications of convex analysis to multidimensional scaling. In: Recent Developments in Statistics, pp. 133–146 (1977)
6. DeCoste, D.: Anytime interval-valued outputs for kernel machines: Fast support vector machine classification via distance geometry. In: ICML, pp. 99–106 (2002)
7. Duda, R., Hart, P., Stork, D.: Pattern Classification, 2nd edn. Wiley, Chichester (2000)
8. Esmeir, S., Markovitch, S.: Anytime induction of decision trees: An iterative improvement approach. In: Proc. of the 21st AAAI (2006)
9. Ester, M., Kriegel, H.-P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases. In: ACM KDD (1996)
10. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: SIGMOD, pp. 47–57 (1984)
11. Hettich, S., Bay, S.: The UCI KDD archive (1999), http://kdd.ics.uci.edu
12. John, G., Langley, P.: Estimating continuous distributions in bayesian classifiers. In: UAI. Morgan Kaufmann, San Francisco (1995)
13. Kranen, P., Assent, I., Baldauf, C., Seidl, T.: Self-adaptive anytime stream clustering. In: Proc. of the 9th IEEE ICDM (2009)
14. Kranen, P., Seidl, T.: Harnessing the strengths of anytime algorithms for constant data streams. DMKD Journal, ECML PKDD Special Issue 19(2), 245–260 (2009)
15. Kullback, S.: Information Theory and Statistics. Wiley, New York (1959)
16. Lauritzen, S.: The EM algorithm for graphical association models with missing data. Comp. Statistics & Data Analysis 19, 191–201 (1995)
17. Patrick, E., Fischer, F.: A generalized k-nearest neighbor rule. Information and Control 16(2), 128–152 (1970)
18. Quinlan, J.R.: Induction of decision trees. Machine Learning 1(1), 81–106 (1986)
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, pp. 311–322 (2009)
20. Silverman, B.: Density Estimation for Statistics and Data Analysis. Chapman & Hall/CRC, Boca Raton (1986)
21. Ueno, K., Xi, X., Keogh, E.J., Lee, D.-J.: Anytime classification using the nearest neighbor algorithm with applications to stream mining. In: ICDM (2006)
22. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)
23. Zilberstein, S.: Using anytime algorithms in intelligent systems. The AI magazine 17(3), 73–83 (1996)