Indexing Density Models for Incremental Learning and Anytime Classification on Data Streams

Thomas Seidl[1], Ira Assent[2], Philipp Kranen[1], Ralph Krieger[1], Jennifer Herrmann[1]

[1]Data Management and Exploration Group, RWTH Aachen University, Germany
{seidl,kranen,krieger,herrmann}@cs.rwth-aachen.de
[2]Department of Computer Science, Aalborg University, Denmark, ira@cs.aau.dk

ABSTRACT

Classification of streaming data faces three basic challenges: it has to deal with huge amounts of data, the varying time between two stream data items must be used best possible (anytime classification) and additional training data must be incrementally learned (anytime learning) for applying the classifier consistently to fast data streams. In this work, we propose a novel index-based technique that can handle all three of the above challenges using the established Bayes classifier on effective kernel density estimators. Our novel Bayes tree automatically generates (adapted efficiently to the individual object to be classified) a hierarchy of mixture densities that represent kernel density estimators at successively coarser levels. Our probability density queries together with novel classification improvement strategies provide the necessary information for very effective classification at any point of interruption. Moreover, we propose a novel evaluation method for anytime classification using Poisson streams and demonstrate the anytime learning performance of the Bayes tree.

1. INTRODUCTION

Anytime classification has gained importance with ubiquitous streams in applications ranging from sensor data in monitoring systems to speech recognition. In monitoring systems, immediate reaction to specific events is necessary, e.g. for emergency detection. Likewise, interactive voice response implies direct recognition of incoming speech signals.

Applications of anytime classification have the following characteristics in common: First, typically large volumes of data need to be processed. Second, immediate reaction is required at interruption, where in many applications the time allowance is not known in advance, but depends on varying stream volume or user interruption. To classify an instance from a bursty stream, the time available for classification can vary from milliseconds to minutes as Ueno et al. noted in [30]. In their recent work they classify insects based on audio sensor monitoring using embedded computers with very limited computational resources. The insects' inter-arrival rate varies from tenths of seconds to tenths of minutes. Another example stems from the robotics domain, where shapes have to be classified for robot navigation. Action is often required in less than the time necessary for exact computation so that the available time before an interrupt should be utilized as efficiently as possible. For classification of objects that are placed on a moving conveyor, as in fruit sorting and grading, the available time can vary by up to three orders of magnitude [30].

In a recent project we worked on a health application to remotely monitor medical patients via body sensor networks [21]. The amount and frequency of data sent by the patients depends on their condition as preclassified locally on the sensor controller. The receiving server has to classify all incoming patient data as accurately as possible. With many patients potentially sending aggregated or detailed data, the amount of data and their arrival rate may vary widely. Hence, to treat each patient in a global emergency situation, e.g. an earthquake, classification for each individual observation must be performed quickly and be improved as long as time permits.

These tasks thus call for classification under varying time constraints. The streaming rate may vary widely yielding possibly large differences in the amount of time available for computation. The enormous amounts of data arriving in streaming applications can neither be stored prior to classification nor can classification of one item take longer than the time until the next item arrives.

Traditional classification aims at determining the class label of unknown objects based on training data. Different classification approaches are discussed in the literature, including Bayes classification [2, 12], neural networks [7, 27], decision trees [24, 20], support vector machines [6] and nearest neighbor techniques [25].

In contrast to anytime algorithms, so called *budget* or *contract* algorithms [36, 8] use a fixed (*budgeted*) time limit for their calculation. If the available time is less than the contracted budget, they cannot guarantee to give a result at all. Another severe limitation is their inability to improve the result if more time is available, as their classification model is simplified to the degree required to fulfill their contract. Their only chance is to restart computation with a more detailed model. This constitutes the major drawback, since they cannot use the results they obtained so far, but have to start computation from scratch.

Finally, when dealing with data streams, not only the amount of data to be classified becomes very large but there

EDBT 2009, March 24–26, 2009, Saint Petersburg, Russia.

is often also constantly new training data available. There has been some research on exploiting this new training data by incrementally training the classifier as the data stream proceeds [10, 11, 33]. As applications one may think of monitoring systems, where an expert sporadically inspects the system and checks the current status manually to provide new training data. Another example could be a production site where only goods of a certain type or class arrive for a period of time which can then be used for incremental learning of the classifier.

So far, classifiers that focused on anytime learning built a contract classifier [5, 29, 32], i.e. they are not able to perform anytime classification. Anytime classifiers [9, 13, 30] on the other hand so far have not been able to profit from novel training data unless they were given a large amount of time to retrain their classifier. For a stream classification application it is important to support both anytime learning and anytime classification to allow consistent use on fast data streams. To the best of our knowledge, no such classifier exists so far.

1.1 Contribution

Our work enables anytime Bayes classification using non-parameterized kernel density estimators. Consistent with classifying fast data streams our technique is able to incrementally learn from data streams at any time. We propose our novel Bayes tree indexing structure that provides aggregated model information about the kernels in all subtrees on all hierarchy levels of the tree. In our new *probability density query*, descending the tree is based on strategies that favor classification accuracy. As different granularities are available and compact models are located at the top levels, the probability density query can be interrupted early on (starting with a unimodal model at root level) and refines the model as long as time permits. Our novel anytime classifier does not only improve its result incrementally when more time is granted, but also adapts the model refinement individually to the object to be classfied.

The design goals of our anytime approach include

- **Statistical foundation.** Our classification uses the established *Bayes classifier* based on kernel estimators.
- **Adaptability.** The model is *adapted to the individual query object*, i.e. the mixture model is refined locally with respect to the object to be classified.
- **Anytime classification and anytime learning** for applying the Bayes tree consistently in applications on fast data streams.

Our contributions on a technical level include

- **Novel hierarchical organization of mixture models.** Our novel index structure provides a hierarchy of mixture density models and allows *fast access* to very fine-grained and individually adapted models.
- **Probability density queries.** Novel access strategies where traversal decisions are based on the *probability density of the query object* with respect to the current mixture model.
- **Poisson stream evaluation.** We propose a novel evaluation method for anytime classification on data streams using *Poisson processes*.

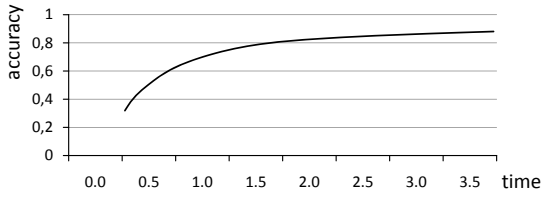

Figure 1: Prototypical anytime classification accuracy: accuracy increases over time.

2. PRELIMINARIES

In this section we first review properties of anytime classification and its related work. Next we decribe Bayes classification and density estimation and section 2.3 discusses anytime learning and hierarchical indexing.

2.1 Anytime classification

Anytime classifiers are capable of dealing with the varying time constraints and high data volumes of stream applications. The advantages of anytime classifiers can be summarized as **flexibility** (exploit all available time), **interruptibility** (provide a decision at any time of interruption) and **incremental improvement** (continue classification improvement without restart).

The usefulness of an anytime classifier depends on its quality, which can be determined with respect to **accuracy** and **accuracy increase**. **Accuracy** refers to the quality of the resulting class label assignment and is measured as the ratio between correctly classified objects and the total number of classified objects $|\{\mathcal{G}(x_i) = y_i | x_i \in TS\}|/|TS|$, where $\mathcal{G}$ is a classifier, x_i is an object in the test set TS and y_i its correct label. In each time step the accuracy for an anytime classifier can be measured as the absolute classification accuracy (as in traditional classifiers) or as the relative classification accuracy with respect to unbound time classification.

Accuracy increase is the improvement of accuracy over time. Ideally, even for little time, classification accuracy is close to the best answer. With more time, classification accuracy should soon reach the classification accuracy of the infinite time classifier [36, 30]. This is illustrated in Figure 1. Early on, the ideal anytime classifier provides high accuracy with rapid increase.

In stream classification the available computation time varies, therefore the accuracy of an anytime classifier is given by the average classification accuracy with respect to the inter-arrival rate (speed) of the data stream. The accuracy increase also influences the stream specific anytime classification accuracy, since a steeper increase yields a better accuracy even if marginally more time is available and thus a higher average accuracy for the same inter-arrival rate. Therefore we propose a novel stream specific anytime classification evaluation. Details are given in Section 4.

Anytime classification has been discussed for boosting techniques [23], decision trees [13], SVMs [9], nearest neighbors [30] and Bayesian networks [19, 22]. An anytime algorithm on discrete-valued data is discussed in [34]. As stated above, non of these classifiers allows for incremental learning. For Bayes classifiers based on kernel densities no anytime classifier exists to the best of our knowledge. We review anytime density estimation in the following section.

2.2 Bayes classification, mixture densities and kernels

A classifier is a function $\mathcal{G}$ which maps an object $\mathbf{x}$ to a label $c_i \in \mathbf{C}$. In our work, we focus on the Bayesian classifier which uses the statistical Bayesian decision theory for classifying objects [2, 12]. The fundamental theory of the Bayesian classifier provides a broad range of theoretical analyses ranging from asymptotic behavior to mean error rates for many applications. The well studied properties of Bayes classifiers make it a widely used approach for classification.

The Bayesian classifier is based on probabilistic distribution estimation of existing data. Based on a statistical model of the class labels, the Bayes classifier chooses the class with the highest *posterior probability* $P(c_i|\mathbf{x})$ for a given query object $\mathbf{x}$ according to the Bayes rule $P(c_i|\mathbf{x}) = (P(c_i) \cdot p(\mathbf{x}|c_i))/p(\mathbf{x})$.

DEFINITION 1. ***Bayes classification.***

Given a set of class labels $\mathbf{C} = \{c_1, \ldots, c_m\}$ *and training data* $\mathbf{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ *with objects* $x_i \in \mathbb{R}^d$ *and class labels* $y_j \in \mathbf{C}$ *assigned to each object, the Bayes classifier* $\mathcal{G}_{Bayes}$ *estimates the a priori probability* $P(c_i)$ *and the class-conditional density* $p(\mathbf{x}|c_i)$ *based on the training data* $\mathbf{D}$ *and assigns an object* $\mathbf{x}$ *to the class with the highest a posteriori probability:*

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

We also define $\mathbf{D}_{c_i} = \{(\mathbf{x}_j, y_j) \in \mathbf{D} \,|\, c_i = y_j\}$ *as the set of objects belonging to a specific class* c_i*.*

The a priori probability $P(c_i)$ can be easily estimated from the training data as the relative frequency of each class $P(c_i) = \frac{|\mathbf{D}_{c_i}|}{|\mathbf{D}|}$. Note that $p(\mathbf{x})$ is left out in the last term, because it does not effect the *argmax* evaluation. Since $\mathbf{x}$ is typically multidimensional, the task of estimating the class-conditional density $p(\mathbf{x}|c_i)$ is not trivial. The naive Bayes classifier uses a straight forward 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. A simple method is to assume a certain distribution of the data. Any model assumption (e.g. a single multivariate normal distribution) may not reflect the true distribution. Mixture densities relax this assumption that the data follows exactly one unimodal model by assuming that the data follows a combination of probability density functions. In our work we use Gaussian mixture densities.

DEFINITION 2. ***Gaussian mixture densities.***

A multivariate Gaussian mixture model $\mathcal{M}$ *combines* k *Gaussian probability density functions* $g(\mathbf{x}, \mu, \Sigma)$ *of the form*

$$g(\mathbf{x}, \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} \cdot det(\Sigma)^{1/2}} e^{\left(-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)\right)}$$

where Σ *is a covariance matrix,* Σ^{-1} *its inverse and* $det(\Sigma)$ *the determinate of* Σ*. When combined, the functions are weighted with a weight* w_j *where* $\sum_{j=1}^{k} w_j = 1$*. The class conditional density probability for an object* $\mathbf{x}$ *and class* $c_i \in \mathbf{C}$ *represented by* k *Gaussian components is then calculated using a Gaussian mixture model* $\mathcal{M}$ *as follows:*

$$p_{\mathcal{M}}(\mathbf{x}|c_i) = \mathcal{M}_{c_i}(\mathbf{x}) = \sum_{j=1}^{k} w_j \cdot g(\mathbf{x}, \mu_j, \Sigma_j)$$

If only variances $(\sigma_1, \ldots, \sigma_d)$ are considered and covariances are neglected, the covariance matrix Σ is replaced by a diagonal matrix Σ' and $det(\Sigma') = \prod_{i=1}^{d} \sigma'_i$.

Another approach to density estimation are kernel densities, which do not make any assumption about the underlying data distribution (thus often termed "model-free" or "non-parameterized" density-estimation). Kernel estimators can be seen as influence functions centered at each data object. To smooth the kernel estimator a *bandwidth* or *window-width* h_i is used. Comparing Definition 2 with kernel densities, the bandwidth corresponds to the variance of a Gaussian component.

DEFINITION 3. ***Kernel density estimation.***

The class conditional probability density for an object $\mathbf{x}$ *and class* c_i *based on a training set* $\mathbf{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ *and kernel* $\mathbf{K}$ *with bandwidth* h_i *is given by:*

$$p_{\mathbf{K}}(\mathbf{x}|c_i) = \frac{1}{|\mathbf{D}_{c_i}| \cdot h_i^d} \sum_{\mathbf{x}_j \in \mathbf{D}_{c_i}} \mathbf{K}\left(\frac{\|\mathbf{x} - \mathbf{x}_j\|}{h_i}\right)$$

Thus, the class conditional probability density for any object $\mathbf{x}$ is the weighted sum of kernel influences of all objects $\mathbf{x_j}$ of the respective class. In this paper, we use the Gaussian kernel $\mathbf{K}_{Gauss}(\mathbf{x}) = \frac{1}{(2 \cdot \pi)^{d/2}} e^{-\frac{\mathbf{x}^2}{2h_i}}$ along with Gaussian mixture models in a consistent model hierarchy to support mixing of models and kernels in the Bayes tree.

There has been some research on anytime density estimation [35, 14]. It is however not described how to use these approaches for anytime classification. As we will clearly see in section 4, this is not a trivial task. Neither a naive solution nor either of the straight forward solutions delivers competitive results.

In terms of classification accuracy, Bayes classifiers using kernel estimators have shown to perform well for traditional classification tasks. Especially for huge training data sets the estimation error using kernel densities is known to be very low and even asymptotically optimal. In section 3 we propose a novel indexing structure that enables kernel density estimation on huge data sets (e.g. as in stream applications) and incremental learning at any time.

2.3 Anytime learning using hierarchical indexing

Typically, supervised classification is performed in two steps: learning and classification. So far we discussed the quality criteria for anytime classification. However, in stream applications it is often essential to efficiently learn from new labeled objects. As the time to learn from new objects is typically limited, classifiers are required which can be interrupted at any time during learning. The classification model which has been learned up to this point in time is then used for all further classification tasks. We call a classifier which can learn from new objects at any time an anytime learning algorithm (also called incremental learning).

An important quality criterion for anytime learning algorithms is the runtime complexity of updating the learned

model. While using lazy learning (e.g. nearest neighbor) allows for updating the decision set in constant time, the lack of a concise model of the data stops this approach from being effective in the anytime classification context. An anytime classifier based on the nearest neighbor approach has been proposed in [30]. However, to perform anytime classification, [30] uses an non-lazy learning method that has a superlinear complexity for new objects.

We propose a hierarchy of models for Bayes classification that allows incremental refinement to meet the requirements of anytime classification and anytime learning. Our novel Bayes tree indexes density models for fast access. Using multidimensional indexing techniques enables our Bayes tree to learn from new objects in logarithmic time.

Multidimensional indexing structures like the R-tree [15] have been shown to provide substantial efficiency gains in similarity search. The basic idea is to organize the data efficiently on disk pages such that only relevant parts of the database have to be accessed. This is based on the assumption that in similarity search, query processing requires only a small portion of the data. This assumption does not hold in Bayes kernel classification. As the entire model potentially contributes to the class label decision, the entire database has to be accessed in order to perform Bayes classification without loss of accuracy (see Section 3 for details). Consequently, simply storing objects for kernel estimation in multidimensional indexes does not suffice for anytime classification. The Gauss-tree [4] builds on the R*-tree structure [3] to processes unimodal probabilistic similarity queries. As the application focus is on similarity search, it does neither allow for anytime classification, nor for management of multimodal density estimators. For anytime classification, we therefore propose a hierarchical indexing of mixture densities. It includes kernel densities as the most detailed level and allows for interruption at any level of the hierarchy, using as much detail as was processed until the point of interruption.

3. THE BAYES TREE

Having introduced anytime algorithms and Bayes classification using kernel density estimation, we are now ready to propose our novel anytime approach. Our overall goal is a classification algorithm that can be interrupted at any given point in time and that produces a meaningful classification result that improves with additional time allowances. The Bayes classifier on kernel densities can not provide a meaningful class conditional density $p(\mathbf{x}|c_i)$ at an arbitrary (early) point in time, since the entire model may potentially contribute to the class label decision (cf. Section 2.3). In the next section we give an overview of our technique for estimating the class conditional density before giving the technical details in the following sections. Finally, we describe how the classification decision is made and improved using our novel index structure.

3.1 Outline

As discussed in Section 1, any anytime classifier has to provide an efficient way to improve its results with additional time. Indexing provides means for efficiency in similarity search and retrieval. By grouping similar data on hard disk and providing directory information on disk page entries, only the relevant parts of the data are accessed during query processing. Similarity search queries are usually

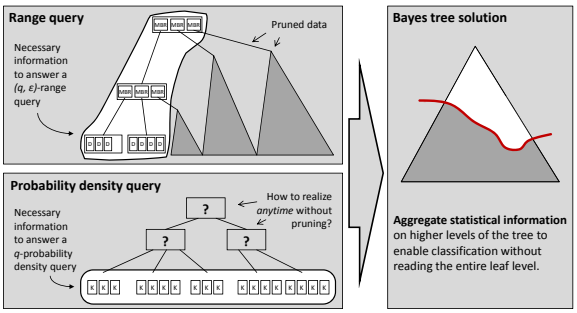

Figure 2: Pruning as in similarity search is not possible for probability density queries, since all kernels have to be accessed to answer a q-probability density query.

specified via a query object and a similarity tolerance given by a threshold range ε. Using this ε range, irrelevant parts of the data are pruned based on directory information during query processing. Thus the amount of data that has to be accessed is reduced, which in turn greatly reduces runtimes. This is illustrated in Figure 2 (top left). The query and the ε range allow for straightfoward pruning of database objects whose directory entries are at more than ε distance (with respect to Euclidean distance). This scenario also applies for k nearest neighbor queries [26].

In the Bayes tree, the data objects are stored at leaf level as in similarity search applications. As classification requires reading all kernel estimators of the entire model, accuracy would be lost if a subset of all kernel densities was ignored. Consequently, there is no irrelevant data, and hence the pruning as in similarity search is infeasible when dealing with density estimation. As for the accuracy increase, accessing the entire kernel density model is not only inefficient but also not interruptible. Moreover, kernel densities provide only a single model of the data, i.e. no incremental improvement of classification is possible as required for anytime classification.

As illustrated in Figure 2 (bottom left), the upper levels of an index that supports anytime classification need to provide information that can be used for assigning class labels even before the leaf level containing the kernels is reached. The Bayes tree solves this problem by storing aggregated statistical information in its inner nodes (Figure 2 right).

Our approach is therefore a hierarchy of mixture models built on top of kernel densities. Each hierarchy level preserves as detailed information on the underlying observations as the node capacity permits, while allowing interruption and adaptive query-based refinement as long as more time is available.

Thus, our Bayes tree approach comprises:

- **Hierarchical indexing** of mixture densities that enables fast access for refinement of the current model
- **Adaptive refinement** of the most relevant parts of the model depending on the individual query
- **Kernel densities for Bayes classification** at the finest level of the hierarchy

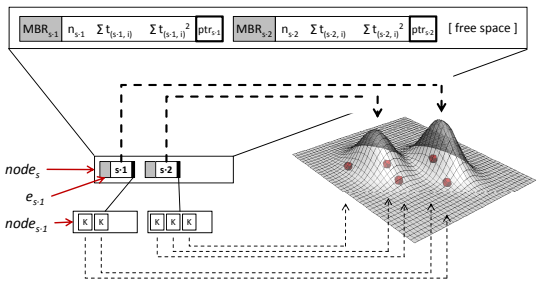

Figure 3: Nodes and entries in the Bayes tree (here: above the kernel level). Entries e_s store minimum bounding rectangles (MBR_s) and child pointers (ptr_s) and additionally information to compute mean and variance, i.e. number of objects (n_s), linear sum ($\sum t_{s,i}$), quadratic sum ($\sum t_{s,i}^2$). Kernel positions are depicted as dots, the higher level mixture density is represented as density curves.

3.2 Structure of the Bayes tree

In this section we describe the Bayes tree structure for a single class c_i. Section 3.3 details the processing of probability density queries.

The general idea of the Bayes tree is a hierarchy of mixture densities stored in a multidimensional index. Each level of the tree stores at a different granularity a complete model of the entire data. To this end, a balanced structure as in R-trees [15, 3] is used to store the kernels at leaf level. The hierarchy on top is built in a bottom-up fashion, providing a hierarchy of node entries, each of which is a Gaussian that represents the entire subtree below it. The index is dynamic and easily adapts to novel data as not the actual parameters of the Gaussian, its mean and its variance, are stored, but easily maintainable information for their computation.

Multidimensional indexing structures like R-trees summarize data recursively in minimum bounding regions (MBRs). Leaves store the actual data objects and the subsequently higher levels provide directory information via the MBRs. For kernel densities or mixture densities, MBR information is not sufficient for describing the probability density distribution of subtrees. Consequently, we propose storing the necessary information to compute parameters of the mixture densities at any given level of the hierarchy.

Recall that the parameters of mixture densities are the mean and variance of Gaussians (μ_i and σ_i, respectively). Storing this information directly would require accessing the actual objects on the leaf level to generate any higher dimensional model representation. For example, assume we have two mixture densities. Building a coarser mixture density that represents both is not possible without additional information, since mean and variance are not distributive, i.e. from two means alone we cannot infer the overall mean. Instead, the number of instances is required to weight the two means accordingly. Mean and variance are thus algebraic measures that require the number of instances, their linear sum and quadratic sum for incremental computation, which is exactly what Bayes tree entries store:

DEFINITION 4. ***Bayes tree node entry.***

A subtree $\mathbf{T_s}$ of a d-dimensional Bayes tree is associated with the set of objects stored in the leaves of the subtree: $\mathbf{T_s} = \{\mathbf{t}_{(s,1)}, \ldots, \mathbf{t}_{(s,n_s)}\}$. An entry e_s then stores the following information about the subtree $\mathbf{T_s}$:

- *The **minimum bounding rectangle** enclosing the objects stored in the subtree $\mathbf{T_s}$ as $MBR_s = ((l_1, u_1), \ldots, (l_d, u_d))$*
- *A **pointer** ptr_s to the subtree $\mathbf{T_s}$*
- *The **number** n_s of objects in $\mathbf{T_s}$*
- *The **linear sum** $\sum_{j=1}^{n_s} \left(\mathbf{t}_{(s,j)}\right)$ of all objects in $\mathbf{T_s}$*
- *The **quadratic sum** $\sum_{j=1}^{n_s} \left(\mathbf{t}_{(s,j)}{}^2\right)$ of all objects in $\mathbf{T_s}$*

Please note that all objects stored in the leaves of the Bayes tree are d-dimensional kernels. Figure 3 illustrates the structure of a Bayes tree node entry. From the information stored in each entry according to Definition 4, mean and variance are computed as follows [17]:

LEMMA 1. ***Computing mean and variance.***

The mean μ_s and the variance vector σ_s^2 for a subtree $\mathbf{T_s}$ can be computed from the stored values of the respective node entry e_s as

$$\mu_s = \frac{1}{n_s} \sum_{j=1}^{n_s} \left(\mathbf{t}_{(s,j)}\right)$$

$$\sigma_s^2 = \frac{1}{n_s} \sum_{j=1}^{n_s} \left(\mathbf{t}_{(s,j)}{}^2\right) - \left(\frac{1}{n_s} \sum_{j=1}^{n_s} \left(\mathbf{t}_{(s,j)}\right)\right)^2$$

Our Bayes tree extends the R*-tree to store model specific information in the following manner:

DEFINITION 5. ***Bayes tree.***

A Bayes tree with fanout parameters $\mathbf{m}, \mathbf{M}$ and leaf node capacity parameters $\mathbf{l}, \mathbf{L}$ is a balanced multidimensional indexing structure with the following properties:

- *Each inner node $node_s$ contains between $\mathbf{m}$ and $\mathbf{M}$ entries (see Def. 4). The root has at least a single entry.*
- *Each inner node with ν_s entries has exactly ν_s child nodes*
- *Leaf nodes store between $\mathbf{l}$ and $\mathbf{L}$ observations (d-dimensional kernels).*
- *A path from the root to any leaf node has always the same length (balanced).*

This structure has some very nice and intuitive benefits: since the number of objects, their linear sum as well as their quadratic sum are all distributive measures, they can efficiently be computed bottom up. More precisely, we simply use the build procedure of any standard R*-tree to create our hierarchy of mixture densities. Additionally, using an index structure from the R-tree family automatically allows both processing of huge amounts of data and incremental learning at any time.

Recall that R*-trees, or any other tree from the B-tree or R-tree family, grow in a bottom-up fashion. Whenever

there are more entries assigned to any node than allowed by the fanout parameters $\mathbf{M}$, which in turn reflects the page size on hard disk, an overflow occurs. This overflow leads to a node split, i.e. the entries of this overfull node are separated onto two new nodes. Their ancestor node then stores aggregate information on these two nodes. In the R*-tree case, this was simply the MBR of the two nodes, for the Bayes tree additionally the overall number, linear and quadratic sum have to be computed and stored in the ancestor node. This way, in a very straightforward manner, coarser mixture density models are created.

Consequently, building Bayes trees is a simple procedure that starts from the original kernel density estimators that are stored in a leaf node until a split is necessary. This first split leads to creation of a second level in the tree, with a coarser mixture density model derived through R*-tree-style split [3]. Successive splitting of nodes (as more kernel densities are inserted) leads to further growth of the tree, eventually yielding a hierarchy of mixture density models as coarser representations of the kernel densities.

3.3 Query processing

Answering a probability density query requires a complete model as stored at each level of the tree. Besides these full models, local refinement of the model (to adapt flexibly to the query) provides models composed of coarser and finer representations. This is illustrated in Figure 4. In any model, each component corresponds to an entry that represents its subtree. This entry may be replaced by the entries in its child node yielding a finer representation of its subtree. This idea leads to query-based refinement in our anytime algorithm. Each mixed granularity model corresponds to a frontier in the tree, i.e. a set of entries in the tree, such that each kernel estimator is represented exactly once. Taking a complete subtree of all nodes starting from the root, the frontier describes a non-redundant model that combines mixture densities at different levels of granularity.

Recall that an entry e_s represents all objects in its corresponding subtree by storing the necessary information to calculate its mean and variance. Hence, a set $E = \{e_i\}$ of entries defines a Gaussian mixture model $\mathcal{M}$ according to Definition 2, which can then be used to answer a probability density query.

DEFINITION 6. ***Probability density query pdq.***

Let $E = \{e_i\}$ be a set of entries, $\mathcal{M}_E$ the corresponding Gaussian mixture model and $\mathbf{n} = \sum_i n_{e_i}$ the total number of objects represented by E. A probability density query pdq returns the density for an object $\mathbf{x}$ with respect to $\mathcal{M}_E$ by

$$pdq(\mathbf{x}, E) = \sum_{e_s \in E} \frac{n_{e_s}}{\mathbf{n}} \cdot g(\mathbf{x}, \mu_{e_s}, \sigma_{e_s})$$

where μ_{e_s} and σ_{e_s} are calculated according to Lemma 1.

For a leaf entry a kernel estimator as discussed in Section 2.2 is used and obviously μ_{e_s} is the object itself.

Figure 4 b) shows the resulting mixture density for the example frontier from part a). The leftmost Gaussian stems from the entry e_1 which is located at root level. The rightmost Gaussian and the one in the back correspond to entries e_{23} and e_{21} respectively, the remaining represent kernel densities at leaf level. Part c) of the image depicts the underlying R*-tree MBRs and the kernels as dots. The bigger blue

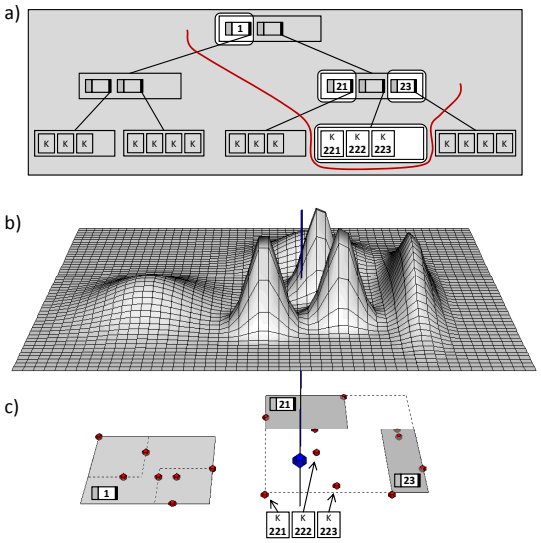

Figure 4: Tree frontier: In a) an exemplary frontier in a Bayes tree that corresponds to a model with mixed levels of granularity is depicted: nodes 2 (root level) and 22 (inner level) are refined, yielding 1 (root level), 21 (inner level), 221, 222, 223 (all leaf level), 23 (inner level). The resulting mixture density model is depicted in b), the actual kernel positions and the hierarchy are depicted in c).

dot and the vertical line represent the query object from which the above frontier originated.

A frontier is thus a model representation that consists of node entries such that each kernel estimator contributes and such that no kernel estimator is represented redundantly. To formalize the notion of a frontier, we need a clear definition for the enumeration of nodes and entries.

DEFINITION 7. ***Enumeration of nodes and entries.***

To refer to each entry stored in the Bayes tree we use a label s with the following properties:

- *The initial starting point is labeled $e_\emptyset$ with $\mathbf{T}_\emptyset$ being the complete set of objects stored in the Bayes tree.*
- *The child node of an entry e_s has the same label s as its parent entry, i.e. the child of e_s is $node_s$. T_s denotes the set of objects stored in the respective subtree of e_s.*
- *The ν_s entries stored in $node_s$ are labeled $\{e_{s\circ 1} \ldots e_{s\circ \nu_s}\}$, i.e. the label s of the predecessor concatenated with the entry's number in the node. (recall Figure 3: $node_s$ has two entries $e_{s\circ 1}$ and $e_{s\circ 2}$ with their child nodes $node_{s\circ 1}$ and $node_{s\circ 2}$).*

Using the above enumeration a frontier can be derived from any prefix-closed set of nodes.

DEFINITION 8. ***Prefix-closed subset and frontier.***

A set of nodes $\mathcal{N}$ is prefix-closed iff:

- *$node_\emptyset \in \mathcal{N}$ (the root is always contained in a prefix-closed subset)*
- *$node_{s\circ i} \in \mathcal{N} \Rightarrow node_s \in \mathcal{N}$ (if a node is contained in $\mathcal{N}$, its parent is also contained in $\mathcal{N}$)*

Let $\mathcal{E}(\mathcal{N})$ be the set of entries stored in the nodes contained in a prefix-closed node set $\mathcal{N}$: $\mathcal{E}(\mathcal{N}) = \bigcup_{node \in \mathcal{N}} (e \in node)$. The frontier $\mathcal{F}(\mathcal{N}) \subseteq \mathcal{E}(\mathcal{N})$ contains all entries $e \in \mathcal{E}(\mathcal{N})$ that satisfy

- *$e_{s \circ i} \in \mathcal{F}(\mathcal{N}) \Rightarrow e_s \notin \mathcal{F}(\mathcal{N})$: no predecessors of a frontier entry is in $\mathcal{F}(\mathcal{N})$ (**predecessor-free**)*
- *$e_s \in \mathcal{F}(\mathcal{N}) \Rightarrow \nexists i \in \mathbb{N}$ with $e_{s \circ i} \in \mathcal{E}(\mathcal{N})$: a frontier entry has no successors in $\mathcal{E}(\mathcal{N})$ (**successor-free**)*

Figure 4 a) illustrates both a prefix-closed subset of nodes and the corresponding frontier. The subset, separated by the curved red line, contains the root, $node_2$ and $node_{23}$, hence it is prefix-closed. Its corresponding frontier (highlighted in white) contains the entries e_1, e_{21}, e_{23}, e_{221}, e_{222} and e_{223} where the last three represent kernels. Note that the frontier is both predecessor-free and successor-free.

Using the above definitions we now define the processing of a probability density query on the Bayes tree.

Let $T_\emptyset$ be the set of objects stored in a Bayes tree containing t_{max} nodes. Anytime *pdq* processing for an object $\mathbf{x}$ processes a prefix-closed subset $\mathcal{N}_t$ in each time step t with $\mathcal{N}_0 = \{node_\emptyset\}$, i.e. the processing starts with the root node, and $|\mathcal{N}_{t+1}| = |\mathcal{N}_t| + 1$, i.e. in each time step one more node is read. The probability density for the query object $\mathbf{x}$ at time step t is then calculated using the mixture model corresponding to the frontier $\mathcal{F}(\mathcal{N}_t)$ of the current prefix-closed subset $\mathcal{N}_t$ as in Definition 6, that is $pdq(\mathbf{x}, \mathcal{F}(\mathcal{N}_t))$. Thereby all objects of the tree are accounted for: $\sum_{e_s \in \mathcal{F}(\mathcal{N}_t)} n_{e_s} = |T_\emptyset|$.

DEFINITION 9. ***Anytime pdq processing.***
From time step t to $t+1$, $\mathcal{N}_t$ becomes $\mathcal{N}_{t+1}$ by adding the child node $node_s$ of one frontier entry $e_s \in \mathcal{F}(\mathcal{N}_t)$. If $node_s$ has ν_s entries, then the frontier $\mathcal{F}(\mathcal{N}_t)$ changes to $\mathcal{F}(\mathcal{N}_{t+1})$ by

$$\mathcal{F}(\mathcal{N}_{t+1}) = (\mathcal{F}(\mathcal{N}_t) \setminus \{e_s\}) \cup \{e_{s\circ 1}, \ldots, e_{s \circ \nu_s}\}$$

i.e. e_s is replaced by its children. The probability density for $\mathbf{x}$ in time step $t+1$ is then calculated by

$$\begin{aligned} pdq(\mathbf{x}, \mathcal{F}(\mathcal{N}_{t+1})) \quad = \quad & pdq(\mathbf{x}, \mathcal{F}(\mathcal{N}_t)) \\ & - \frac{n_s}{|T_\emptyset|} \cdot g(\mathbf{x}, \mu_{e_s}, \sigma_{e_s}) \\ & + \sum_{i=1}^{\nu_s} \frac{n_{s \circ i}}{|T_\emptyset|} \cdot g(\mathbf{x}, \mu_{e_{s \circ i}}, \sigma_{e_{s \circ i}}) \end{aligned}$$

Following the above equation, the probability density for $\mathbf{x}$ in time step $t+1$ is calculated taking the probability density for $\mathbf{x}$ in time step t (row 1), subtracting the contribution of the refined entry's Gaussian (row 2) and adding the contributions of its children's Gaussians (row 3). Hence, the cost for calculating the new probability density for $\mathbf{x}$ after reading one additional node is very low due to the information stored for mean and variance.

Note that after reading all t_{max} nodes $pdq(\mathbf{x}, \mathcal{F}(\mathcal{N}_{t_{max}}))$ is a full kernel density estimation taking all kernels at leaf level into account.

Each possible frontier represents a possible model for our anytime algorithm. The number of possible models in any Bayes tree depends on the actual status of the tree, i.e. the degree to which it is filled and the height of the tree. For example, there are four possible models for a Bayes tree of height two and fanout two (root, leaves, and two mixed models). For realistic trees, with increasing height and fanout, the number of possible models is exponential in the height and in the fanout.

Choosing among all these models is crucial for anytime algorithms, where the time available (typically unknown a priori) should be spent such that the most promising model information is used first. For tree traversal we propose three basic descent strategies to answer probability density queries on a Bayes tree. Descent in a breadth first (*bft*) fashion refines each model level completely before descending down to the next level. Alternatively, we could descent the tree in a depth first (*dft*) manner, refining a single subtree entirely down to the actual kernel estimators before refining the next subtree below the root. The choice of the subtree to refine is made according to a priority measure. The third approach, which we call global best first (*glo*), orders nodes globally with respect to a priority measure and refines nodes in this ordering.

The priority measure in R-Trees, which we denote as *geometric*, gives highest priority to the node with the smallest (Euclidean) distance based on its minimum bounding rectangle.

In the Bayes tree, the focus is not on distances, but on density estimation. We therefore propose a *probabilistic* priority measure. It awards priority with respect to the actual density value for the current query object based on the statistical information stored in each node. More specifically, at time step t, having read the prefix-closed set of nodes $\mathcal{N}_t$, the probabilistic approach descends at the next time step $t+1$ into the subtree $\mathbf{T_{\hat{s}}}$ belonging to the entry $e_{\hat{s}}$ with

$$e_{\hat{s}} = \underset{e_s \in \mathcal{F}(\mathcal{N}_t)}{argmax} \left\{ g(\mathbf{x}, \mu_{e_s}, \sigma_{e_s}) \cdot \frac{n_s}{\mathbf{n}} \right\}$$

where $\mathbf{x}$ is the current query object and g is a Gaussian probability density function as in Definition 2.

In our experiments we evaluate all combinations between the three descent strategies and the two priority measures, where in the breadth first approach the priority measure determines the order of nodes per level.

So far, we described descent strategies in a single Bayes tree, i.e. refining mixture models for just one class. Now, we discuss refinement with respect to several classes $\{c_1, \ldots, c_m\}$. The time for classification is divided between several Bayes trees, one per class (cf. Section 3.2). The question is, having read l nodes so far, i.e. being at time step t_l, which of the m trees should be granted the right to read the next node in the following time step $t_l + 1$?

Initially, we evaluate the coarsest model for each class c_i, i.e. the model consisting of only one Gaussian probability density function representing all objects of c_i.

A naive improvement strategy chooses the classes in an arbitrary order $C = (c_1, \ldots, c_m)$. First, the class c_1 is refined according to one of the above strategies. In the next step, c_2 is refined and so on. After m time steps all classes have been refined and the Bayes tree of the first class is processed again. As the classes are not sorted in any predefined way, we refer to this method as *unsorted*.

Intuitively, it is very likely that for an object $\mathbf{x}$ the true class c_i shows a high posterior probability independent of the time t_{c_i} spent decending its corresponding tree: $P(c_i|\mathbf{x}) = pdq(\mathbf{x}, \mathcal{F}(\mathcal{N}_{t_{c_i}})) \cdot P(c_i)/p(\mathbf{x})$.

Consequently, we suggest a second strategy for improv-

ing classification which only chooses the k classes having the highest a posteriori probability. This technique stores these k classes in a priority queue (*queue best k*). Assume the sorted priority queue contains the classes $(c_{i_1}, \ldots, c_{i_k})$. The next node is read in the Bayes tree of class c_{i_1} and the probability model is updated accordingly. If more time is permitted class c_{i_2} is refined and so on. After k steps all k classes have been improved and the queue is filled again using the new a posteriori probabilities. We evaluate the influence of the parameter k in the experiment section. Setting $k = 1$, always the model of the most probable class is refined. For $k = m$ (the number of classes) all classes are refined in the order of their probability before the queue is refilled. We refer to these approaches as *qb1*, *qbm*, respectively, and *qbk* in general.

One observation which is also known from hierarchical indexing is that the root node often only contains a few entries. We use this free space for storing the entries from different classes on the same page if it does not exceed the page size. We call these new root pages the *compressed root*. The total size of the compessed root varies since the number of root entries in a single Bayes tree varies as in all index structures. Clearly, the total size also grows linearly in the number of dimensions and the number of classes.

4. EXPERIMENTS

We ran experiments on real world and synthetic data to evaluate our Bayes tree for both anytime classification and anytime learning. Table 1 summarizes the data sets and their characteristics. The synthetic data sets were generated using a random Gaussian mixture model as in [17]. We created one small data set containing 4 dimensions, 2 classes and eleven thousand objects and one large data set containing 5 dimensions, 3 classes and one million objects. Furthermore we use real world data of various characteristics which were taken from different sources such as the UCI KDD archive [18].

In all experiments we use a 4 fold cross validation. To evaluate anytime accuracy we report the accuracy (averaged over the four folds) of our classifiers after each node (corresponding to one page) that is read. The first accuracy value corresponds to the classification accuracy of the unimodal Bayes classifier that we use as an initialization before reading the compressed root (cf. Section 4.1). Experiments were run using 2KB page sizes (except 4KB for Verbmobil and 8KB for the USPS data set) on Pentium 4 machines with 2.4 Ghz and 1 GB main memory. To set the bandwidth for

name	size	classes	features	ref.
Synthetic (11K)	11,000	2	4	
Synthetic (1M)	1,000,000	3	5	
Vowel	990	11	10	[18]
USPS	8,772	10	39	[16]
Pendigits	10,992	10	16	[18]
Letter	20,000	26	16	[18]
Gender	189,961	2	9	[1]
Covtype	581,012	7	10	[18]
Verbmobil	647,585	5	32	[31]

Table 1: Data sets used in our experiments.

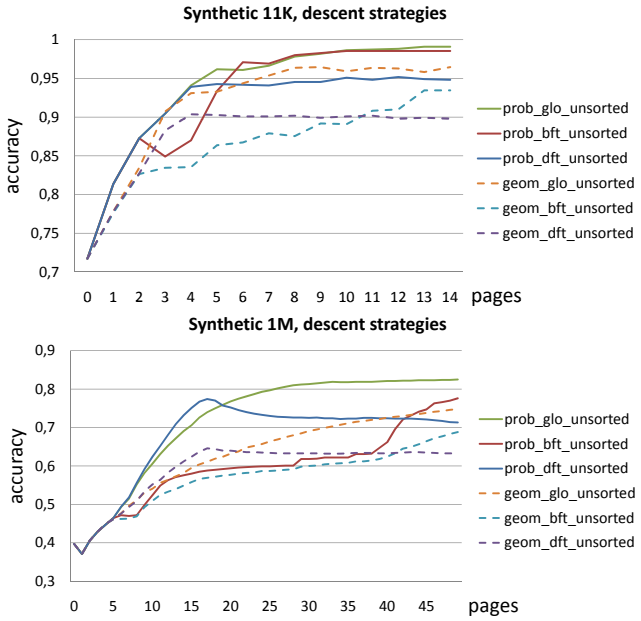

Figure 5: Comparing descent strategies along with priority measures on synthetic data.

our kernel estimators we use a common data independent method according to [28].

In the following we first evaluate the descent strategies along with the priority measures introduced in Section 3.3. In Section 4.2 we analyze various improvement strategies to find that none of the three simple approaches leads to competitive results. After that we introduce our novel evaluation method for anytime classifiers on Poisson streams in Section 4.3. Finally we demonstrate the anytime learning performance of the Bayes tree for both anytime accuracy and stream accuracy.

4.1 Descent strategies

For tree traversal we evaluate depth first (*dft*), breadth first (*bft*) and global best first (*glo*) strategies. Combined with both the geometric (*geom*) and probabilistic (*prob*) priority measure we compare six approaches for model refinement in a Bayes tree. Figure 5 shows the results for all six approaches on the small and on the large synthetic data set using the *unsorted* improvement strategy. After 5 pages on the small synthetic data set the locality assumption of the depth first approach fails to identify the most relevant refinements for both geometric and probabilistic priority. For the large synthetic data set the accuracy even decreases after 17 pages. The processing strategy of the breadth first approach causes the classification accuracy to only increase if a next level of the tree is reached.

We found that the performance of the different descent strategies is independent of the employed improvement strategy (i.e. *unsorted* or *qbk*). As an example for the independence of the improvement strategy Figure 6 shows the performance on the Gender data set using *qbm*. The *glo* descent strategy with probabilistic priority measure shows the overall best anytime classification accuracy for all data sets. We therefore use global best first descent and the probabilistic priority measure in all further experiments.

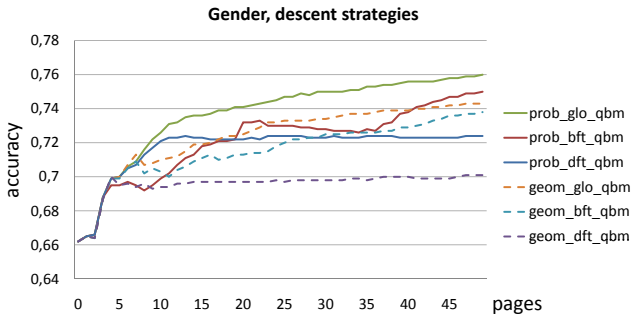

Figure 6: Comparing descent strategies along with priority measures on the Gender data set.

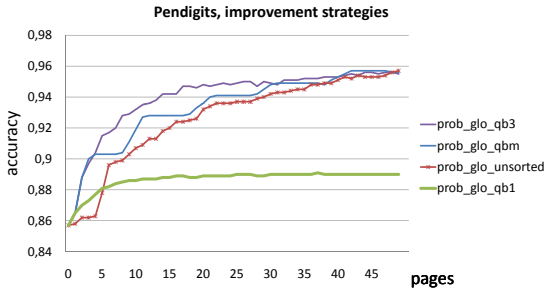

Figure 7: Comparing improvement strategies on Pendigits.

4.2 Improvement strategies

We evaluated the improvement strategies *unsorted*, *qb1* and *qbm* on all seven real world data sets from Table 1. Besides that we used *qb3* for a first evaluation, i.e. we set k to 3 (2 for Gender) as it follows our intuition of "some relevant candidates". (A thorough evaluation of various values for k follows later in this section.)

We picked the results from Pendigits and Covtype as they represent the two sorts of outcome we found when we analyzed the results (cf. Figures 7 and 8). The naive approach, i.e. the unsorted improvement strategy, does not provide a competitive solution (third rank on Pendigits and last rank on Covtype). However, after reading the entire compressed root this approach has read the same information as *qbm*. This can best be seen on the results for Covtype. After four page accesses the compressed root is read and both strategies deliver the same classification accuracy. From then on, their accuracy is the same after every seven steps, i.e. 11, 18, 25, ... since Covtype has seven classes. The graph for Pendigits does not show this behavior in such a clear fashion. This is due to the averaging over the four folds. The compressed root needs between seven and eight pages within each fold. Therefore the accuracy for *unsorted* and *qbm* is similar after seven to eight steps, yet not equal. This similarity occurs every ten steps from then on, since Pendigits has ten classes.

Regarding the other improvement strategies, one could expect that *qb1*, i.e. refining only the most probable class, delivers the best results, because it either strengthens the current decision by increasing the probability for the correct class or corrects the decision by decreasing the probability in case a false class has currently the highest density. On

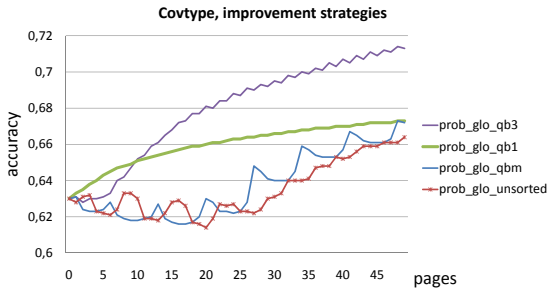

Figure 8: Comparing improvement strategies on Covtype.

Covtype *qb1* outperforms the *qbm* and *unsorted* strategies, but on Pendigits it performs way worse then both of them. Opposite success and failure would befall those in favor for *qbm*.

The results on Verbmobil and Gender were similar to those on Pendigits, i.e. *qb1* performed way worse than all other strategies. On the other hand, similar to the Covtype results, *qb1* performed better than *qbm* and *unsorted* on Letter and USPS. On all datasets all of the three strategies discussed above were clearly outperformed by the *qb3* approach.

These results prove that anytime density estimation does not extend in a naive or straight forward way to anytime classification with competitive results. Moreover, the results pose the question, how the choice of k influences the anytime classification performance. Figure 9 shows the comparison of *qb1*, *qbm* and *qbk* for $k \in \{2, 4, 6, 8, 10\}$ on the Letter data set. As stated above, *qb1* performs better than *qbm* on this data set.

The accuracy of the *qbm* strategy in Figure 9 initially increases steeply in the first three to four steps. It is clearly visible that the compressed root needs on average 19 pages for the 26 classes (recall the 4 fold cross validation). After the compressed root is read, the accuracy of the *qbm* strategy again steeply increases for the next four to five steps. A similar increase can be observed 26 steps later when all classes have been refined once more. Opposed to those three strong improvements are the rest of the steps, during which the accuracy hardly changes at all. For this data set we derive from Figure 9 that for an object that has to be classified there are on average five classes that are worth considering, the other 21 classes can be precluded.

The *qb10* strategy needs seven to eight pages to read the root information for the top ten classes from the compressed root. These steps show the same accuracy as the *qbm* approach, since the ordering according to the priority measure is equal. After that a similar pattern can be observed each ten steps, i.e. after reordering according to the novel density values.

Continuing with the *qb8* variant, the "ramps" (steep improvements) move even closer together and begin earlier. Similar improvement gains are shown by *qb6* and *qb4*. An even smaller k ($k = 2$ in Figure 9) does no longer improve the overall anytime accuracy and *qb1* shows an even worse performance than *qb10*.

For an easier comparison of the *qbk* performance when varying k, we calculate the area under the anytime curve for each k and hence receive a single number for each value of k.

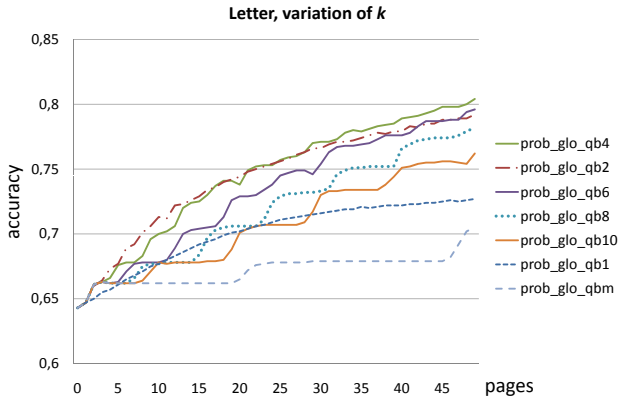

Figure 9: Variation of k on the Letter data set.

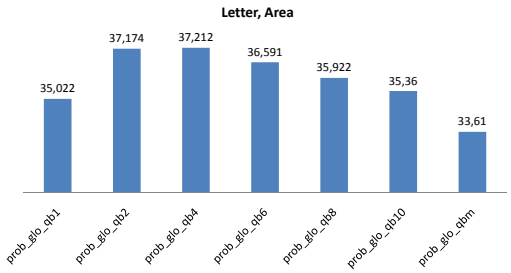

Figure 10: The area under the corresponding anytime curves in Figure 9.

The area fits the intuition of a best improvement, since qbk curves mostly ascend monotonically. Figure 10 shows the area values corresponding to the anytime curves in Figure 9. As in the above analysis $qb4$ turns out to perform best according to this measure.

We evaluated qbk for $k \in \{1 \dots m\}$ on all data sets and found the same characteristic results. Figure 11 displays the results for Vowel and Covtype. We found that the maximum value was always between $k = 2$ and $k = 4$ on our data sets. Moreover, with an increasing number of classes, the k value for the best performance also increased, yet only slightly. An exception is the Gender data set, where $k = m = 2$ showed the best performance. This is due to the bad performance of $qb1$, therefore we set k to be at least 2. To develop a heuristic for the value of k we evaluated different data sets having different number of classes. It turned out that a good choice for k is logarithmic in the number of classes, i.e. $k = \lfloor \log_2(m) \rfloor$. Using this heuristic met the maximal performance for all our evaluations. Yet we set the minimum value for k to 2 as mentioned above. We use this improvement strategy along with the best descent strategy from Section 4.1 in all further experiments.

4.3 Evaluating anytime classification using Poisson streams

So far we determined the best strategy for descending a Bayes tree to refine the density model and evaluated the proposed improvement strategies. Next we propose a novel evaluation method for anytime classifiers using Poisson streams before we demonstrate the anytime learning performance of our technique on various data streams.

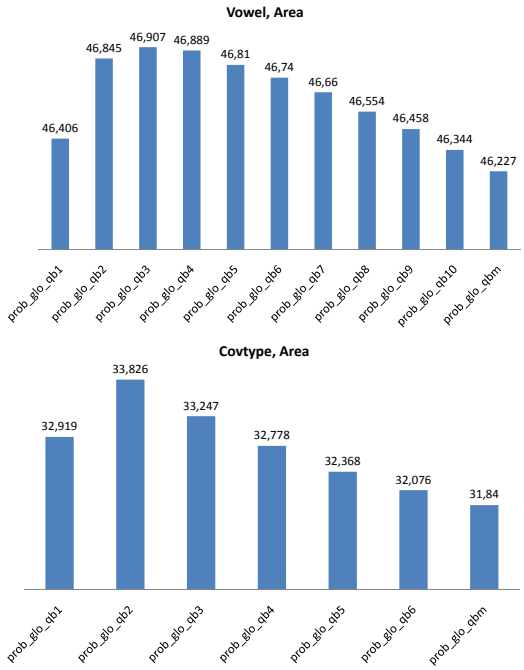

Figure 11: Area values for $k \in \{1 \dots m\}$ for Vowel (top) and Covtype (bottom).

To evaluate anytime classification under variable stream scenarios, we recapitulate a stochastic model that is widely used to model random arrivals [12]. A Poisson process describes streams where the inter-arrival times are independently exponentially distributed. Poisson processes are parameterized by an arrival rate parameter λ:

DEFINITION 10. ***Poisson stream.***

A probability density function for the inter-arrival time of a Poisson process is exponentially distributed with parameter λ by $p(t) = \lambda \cdot e^{-\lambda t}$.The expected inter-arrival time of an exponentially distributed random variable with parameter λ is $E[t] = \frac{1}{\lambda}$.

We use a Poisson process to model random stream arrivals for each fold of the cross validation. We randomly generate exponentially distributed inter-arrival times for different values of λ. If a new object arrives (the time between two objects has passed) we interrupt the anytime classifier and measure its accuracy. We repeat this experiment using different expected inter-arrival times $\frac{1}{\lambda}$, where a unit corresponds to a page as in all previous experiments. We assume that any object arrives at the earliest after the initialization phase of the previous object, i.e. after evaluating the unimodal Bayes.

Figure 12 shows the results for the stream classification accuracy as described above for different values of λ on Pendigits, USPS, Covtype, Verbmobil and the large synthetic data set. All data sets show an improvement of accuracy as the expected inter-arrival time $1/\lambda$ increases and arrival times are distributed following a Poisson process. This is an excellent feature that contrasts anytime classifiers from budget or contract classifiers. Budget classifiers would have to use a time budget that is small enough to guarantee the processing of all arriving objects which would clearly result in worse

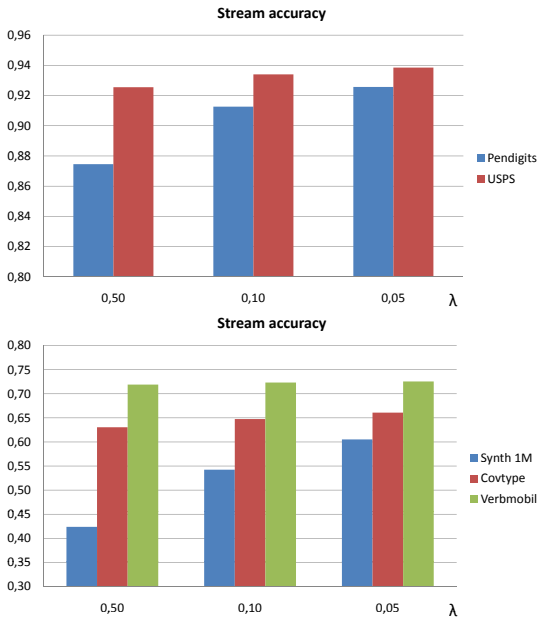

Figure 12: Stream classification accuracy using Poisson processes for data set Pendigits and USPS (top) and Synthetic 1M, Covtype and Verbmobil.

performance. Moreover, evaluating anytime classifiers using Poisson processes yields a single accuracy value for each λ making it very easy to compare performance on different underlying stream.

4.4 Incremental learning

Incremental learning in stream classification applications originates from labeled objects that newly arrive in the data stream. If we started learning/inserting a new object, we always finish the insertion for that object, i.e. no roll back will be performed. However, if the average inter-arrival time between two object is larger than the time necessary for learning an additional object, the classifier cannot process all labeled objects online. Hence, the resulting classifier is based on a varying amount of training data depending on the stream speed, i.e. λ as in Definition 10. To evaluate the incremental learning performance of the Bayes tree we varied λ such that the classifier was only able to process a certain percentage of the training data. We report the results for 50%, 75% and 100%. We first report the anytime classification accuracy for the three classifiers. To be able to compare the results, the classifiers had to classify each the same stream of test objects . As above we averaged the results over 4 folds.

Figure 13 shows the results of the three incremental learning classifiers through their anytime accuracy on the Gender data set. Even the classifier trained on the fastest stream (train 50%) shows a good performance. On slower streams, the Bayes tree can exploit its incremental learning property and learn more objects before the training phase is over. The resulting anytime accuracy (train 75%) lies constantly above train 50% from page three onwards. The Bayes tree that was trained on the slowest stream (train 100%) performs consistently better then both the others, again highlighting the benefit of the incremental learning property.

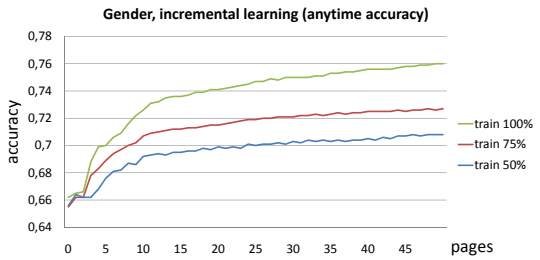

Figure 13: Incremental learning on Gender.

Finally we have evaluated the results of the incrementally trained Bayes trees on Poisson streams for various λ. Figures 14 and 15 show the resulting stream accuracy values for Vowel and Covtype respectively. Again we used the same underlying stream models during classification to be able to compare the results. The anytime learning performance can be seen within each of the groups of three bars, where immediate comparison of the classifiers is possible through our Poisson stream evaluation. For all of the evaluated λ values and data set the classifiers trained on slower streams perform better. Moreover, with smaller λ values during classification, the stream accuracy of each individual Bayes tree improves.

The evaluation shows that our novel Bayes tree efficiently supports incremental learning as well as anytime classification. Moreover, the underlying index structure ensures the ability to handle very large data sets as in the case of streaming data.

5. CONCLUSION AND FUTURE WORK

In this work we proposed a novel index-based classifier called Bayes tree. It supports anytime learning and anytime classification and can handle huge amounts of data, which makes it a consistent solution for classification on fast data streams. The Bayes tree automatically generates (adapted efficiently to the individual object to be classified) a hierarchy of mixture densities that represent kernel density estimators at successively coarser levels. Our probability density queries together with the evaluated improvement strategies provide the necessary information for very effective classification at any point of interruption. Moreover, we proposed a novel evaluation method for anytime classification using Poisson streams and demonstrated the anytime learning performance of our novel approach. In our future work we plan to expand our technology for handling concept drift in data

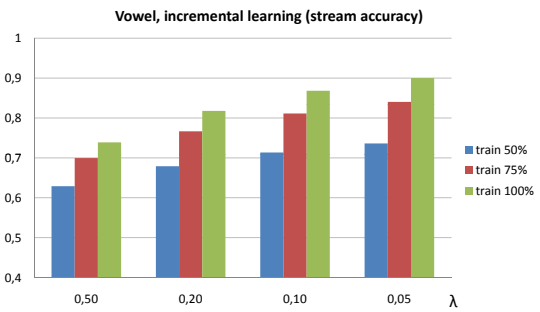

Figure 14: Incremental learning on Vowel.

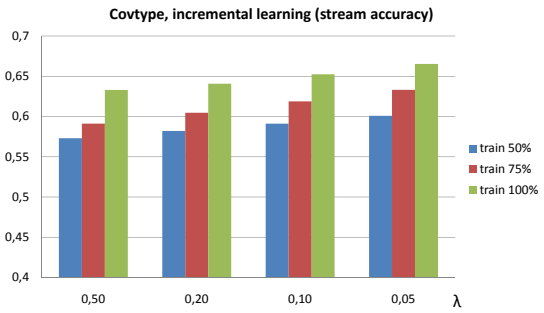

Figure 15: Incremental learning on Covtype.

streams. Other possible extensions include classification of multi-labeled data and semi-supervised learning.

Acknowledgment

This work was partially funded by DFG grant EXC 89.

6. REFERENCES

[1] D. Andre and P. Stone. Physiological data modeling contest (ICML-2004): http://www.cs.utexas.edu/users/pstone/workshops /2004icml/, 2004.

[2] T. Bayes. An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society*, 53:370–418, 1763.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[4] C. Böhm, A. Pryakhin, and M. Schubert. The Gauss-Tree: Efficient Object Identification in Databases of Probabilistic Feature Vectors. *ICDE*, 2006.

[5] J. S. Breese and E. Horvitz. Ideal reformulation of belief networks. In *UAI*, pages 129–144, 1990.

[6] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *DMKD*, 2(2):121–167, 1998.

[7] Y. Cao and J. Wu. Projective art for clustering data sets in high dimensional spaces. *Neural Networks*, 15(1):105–120, 2002.

[8] K. Crammer, J. S. Kandola, and Y. Singer. Online classification on a budget. In *NIPS*, 2003.

[9] D. DeCoste. Anytime interval-valued outputs for kernel machines: Fast support vector machine classification via distance geometry. In *ICML*, 2002.

[10] P. Domingos and G. Hulten. Mining high-speed data streams. In *KDD*, pages 71–80, 2000.

[11] P. Domingos and G. Hulten. Learning from infinite data in finite time. In *NIPS*, pages 673–680, 2001.

[12] R. Duda, P. Hart, and D. Stork. *Pattern Classification (2nd Edition)*. Wiley, 2000.

[13] S. Esmeir and S. Markovitch. Interruptible anytime algorithms for iterative improvement of decision trees. In *UBDM workshop at KDD*, 2005.

[14] A. G. Gray and A. W. Moore. Nonparametric density estimation: Toward computational tractability. In *SDM*, 2003.

[15] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[16] T. Hastie, R. Tibshirani, and J. H. Friedman. Datasets for "The Elements of Statistical Learning": http://www-stat.stanford.edu/~tibs/elemstatlearn/.

[17] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2002.

[18] S. Hettich and S. Bay. The UCI KDD archive http://kdd.ics.uci.edu, 1999.

[19] G. Hulten and P. Domingos. Mining complex models from arbitrarily large databases in constant time. In *KDD*, pages 525–531, 2002.

[20] M. Jordan and R. Jacobs. Hierarchical Mixtures of Experts and the EM Algorithm. *Graphical Models: Foundations of Neural Computation*, 2001.

[21] P. Kranen, D. Kensche, S. Kim, N. Zimmermann, E. Muller, C. Quix, X. Li, T. Gries, T. Seidl, M. Jarke, and S. Leonhardt. Mobile mining and information management in healthnet scenarios. In *MDM*, 2008.

[22] C.-L. Liu and M. P. Wellman. On state-space abstraction for anytime evaluation of bayesian networks. *SIGART Bulletin*, 7(2):50–57, 1996.

[23] K. Myers, M. J. Kearns, S. P. Singh, and M. A. Walker. A boosting approach to topic spotting on subdialogues. In *ICML*, pages 655–662, 2000.

[24] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992.

[25] T. Seidl. *Nearest Neighbor Classification / Liu L., Özsu M. T. (eds.): Encyclopedia of Database Systems. (to appear)*. Springer, 2009.

[26] T. Seidl and K. H.-P. Optimal multi-step k-nearest neighbor search. In *Proc. ACM SIGMOD Int. Conf. on Management of Data, Seattle, Washington*, pages 154–165, 1998.

[27] L. Silva, J. M. de Sa, and L. Alexandre. Neural network classification using shannonŠs entropy. In *ESANN*, 2005.

[28] B. Silverman. Density Estimation for Statistics and Data Analysis. 1986.

[29] W. N. Street and Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *KDD*, pages 377–382, 2001.

[30] K. Ueno, X. Xi, E. Keogh, and D. Lee. Anytime Classification Using the Nearest Neighbor Algorithm with Applications to Stream Mining. *ICDM*, pages 623–632, 2006.

[31] W. Wahlster. *Verbmobil: Foundations of Speech-To-Speech Translation*. Springer, 2000.

[32] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *KDD*, pages 226–235, 2003.

[33] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[34] Y. Yang, G. Webb, K. Korb, and K. M. Ting. Classifying under computational resource constraints: anytime classification using probabilistic estimators. *Machine Learning*, 69(1):35–53, 2007.

[35] T. Zhang, R. Ramakrishnan, and M. Livny. Fast density estimation using cf-kernel for very large databases. In *KDD*, pages 312–316, 1999.

[36] S. Zilberstein. Using anytime algorithms in intelligent systems. *The AI magazine*, 17(3):73–83, 1996.