Using Index Structures for Anytime Stream Mining

Philipp Kranen

Supervised by Thomas Seidl
Data Management and Exploration Group
RWTH Aachen University, Germany

{kranen, seidl}@cs.rwth-aachen.de

ABSTRACT

Stream data mining has gained a lot of attention over the last years due to an abundance of streaming data in professional as well as personal applications. Solutions have been proposed for many mining tasks such as clustering, classification, frequent item set mining and aggregation. Stream mining is especially challenging due to the large (usually endless) amount of data and the time constraints posed by the stream's arrival rate. We recently presented an index-based solution for anytime stream classification that handles both large amounts of data and arbitrary arrival times. In this paper we present our ongoing work, wherein we investigate bulk loading strategies to improve the classification accuracy w.r.t. anytime constraints. We show promising results and discuss future challenges related to index-based classification on data streams. Furthermore we discuss extensions of our technique to other data mining tasks.

1. INTRODUCTION

Data streams are ubiquitous. An abundance of streaming data emerges from numerous applications such as machine monitoring, health monitoring, sensor networks, speech recognition, network protocols, customer transaction data, and so forth. The nature of the streams differs and can generally be divided into constant streams, where the time between two consecutive stream data items is constant, and varying streams, where the amount of data per time unit is varying.

Mining on data streams has been extensively researched over the past years. Algorithms have been developed or adapted in many mining areas such as clustering [1, 2], classification [3], frequent item set mining [15] and aggregation [17]. New research areas such as concept drift detection evolved from stream data mining due to the additional time component. The time however poses another challenge to stream mining algorithms since the processing time is limited by the interval between two consecutive stream data items. This problem has been tackled by two different approaches: budget algorithms and anytime algorithms.

Budget algorithms use a fixed (budgeted) time limit for their calculation. They can neither provide a result in less time nor exploit additional time to improve their result. Anytime algorithms provide a result at any time of interruption and improve its quality with more time allowance. Anytime algorithms are an active field of research, e.g. in clustering [22], top k processing [4] and anytime learning of classifiers [23]. Many anytime classifiers have been proposed, e.g. for support vector machines [7], decision trees [9], nearest neighbor classifiers [20] and Bayes classifiers [24].

A second issue when dealing with classification on data streams is related to the training of a classifier. In many applications like machine monitoring or health monitoring there is constantly new training data available. This data may result from supervised hospital situations or monitoring applications where an expert sporadically classifies the current status manually. To apply a classifier consistently in a stream scenario it is therefore important to be able to learn from new training data incrementally and in an online fashion. Especially in the light of evolving data the model of a classifier has to be updated using new training data.

Recently we proposed a novel anytime classifier that we called Bayes tree [16]. It is capable of classifying objects with arbitrary time allowance and can learn from new training data incrementally and online. The Bayes tree is essentially an index structure that stores a hierarchy of Gaussian mixture models to enable anytime Bayesian classification. We showed its performance in various stream scenarios on benchmark data sets in [16] and applied our technique in a health monitoring application [13].

To improve the anytime classification performance of the Bayes tree we investigate bulk loading techniques from different perspectives. Since our classifier is based on the statistical Bayes approach, we investigate statistical methods to approximate a given Gaussian mixture model with a coarser mixture representation. Furthermore we evaluate machine learning concepts based on clustering methods and compare those techniques against traditional bulk loading algorithms from the literature. We achieved promising results that we will present in this paper.

Our goal in [16] was to enable anytime Bayesian classification by the means of hierarchical index structures. We recapitulate the Bayes tree in the next section. Our next goal is the improvement of its performance. We present ongoing work and results in Section 3 and describe further steps in Section 4.1. The third goal is the extension of our technique to other data mining tasks, ideas and approaches are discussed in Section 4.2.

2. THE BAYES TREE

We first briefly review Bayesian classification and density estimation in Section 2.1 and then describe the structure and working of the Bayes tree.

2.1 Preliminaries

Given a set $\mathbf{C}$ of classes, 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. 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, i.e.

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

Note that $p(\mathbf{x})$ is left out in the last term, because it does not affect the *argmax* evaluation. With $\mathbf{D}_{c_i} = \{(\mathbf{x}_j, y_j) \in \mathbf{D} \,|\, c_i = y_j\}$ 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 set $\mathbf{D}$ as the relative frequency of each class $P(c_i) = \frac{|\mathbf{D}_{c_i}|}{|\mathbf{D}|}$. Since $\mathbf{x}$ is typically multidimensional, the task of estimating the class-conditional density $p(\mathbf{x}|c_i)$ is not trivial.

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 $p(\mathbf{x}|c_i) = \sum_{j=1}^{k} w_j \cdot g(\mathbf{x}, \mu_j, \sigma_j)$, where μ_j is the mean of the j-th Gaussian component, w_j its weight and σ_j its variance vector.

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. 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. 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. 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. To set the bandwidth h_i for our d-dimensional kernel estimators we use a common data independent method according to [18].

2.2 Structure, descent and query processing

Our goal in [16] was to enable anytime kernel density estimation for efficient and interruptible classification. 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. 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. The Bayes tree solves this problem by storing aggregated statistical information in its inner nodes.

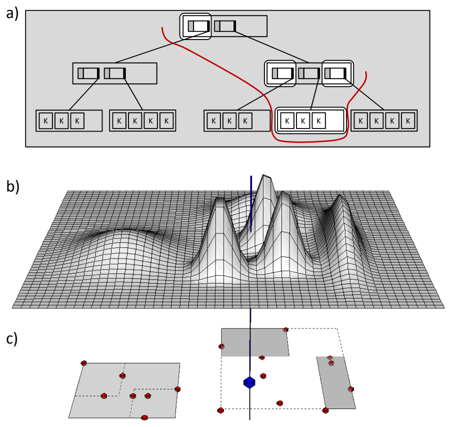


Figure 1: a) Bayes tree and frontier. b) The resulting mixture model. c) The underlying R-tree structure.

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 [11] is used to store the kernels at leaf level. The directory 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. To derive the mixture models we store the necessary information to compute parameters of the mixture densities, i.e. the mean and variance of the Gaussians, in the entries.

Definition 1. ***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 **cluster feature** $CF = (n_s, LS, SS)$ of the objects in $\mathbf{T_s}$ containing the number n_s of objects, their linear sum LS and their squared sum SS*

Please note that all objects stored in the leaves of the Bayes tree are d-dimensional kernels. 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 $\mu_s = LS/n_s$ and $\sigma_s^2 = SS/n_s - (LS/n_s)^2$. Our Bayes tree extends the R*-tree to store model specific information in the following manner:

Definition 2. ***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. Each inner node $node_s$ contains between

$\mathbf{m}$ and $\mathbf{M}$ entries (see Def. 1). The root has at least a single entry and 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).

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 1b. 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. Figure 1 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 dot and the vertical line represent the query object from which the above frontier originated.

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, which can then be used to answer a probability density query.

Definition 3. ***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 as described above. For a leaf entry a kernel estimator as discussed in Section 2.1 is used and obviously μ_{e_s} is the object itself.

From time step t to $t+1$, the set of entries in the frontier changes by adding all entries in the child node $node_s$ of one frontier entry $e_s \in E$. If $node_s$ has ν_s entries, then the frontier's entry set E_t changes to E_{t+1} by

$$E_{t+1} = (E_t \setminus \{e_s\}) \cup \{e_{s \circ 1}, \ldots, e_{s \circ \nu_s}\}$$

i.e. e_s is replaced by its children. The probability density for $\mathbf{x}$ in time step $t+1$ is calculated taking the probability density for $\mathbf{x}$ in time step t, subtracting the contribution of the refined entry's Gaussian and adding the contributions of its children's Gaussians. 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.

For tree traversal we evaluated three basic descent strategies: breadth first (*bft*), depth first (*dft*) and global best descent (*glo*), which orders nodes globally with respect to a priority measure and refines nodes in this ordering. For the priority measure we tested a geometric measure, i.e. the distance from the query object to the MBR, and a probabilistic measure, i.e. the weighted probability density for the query object w.r.t. the Gaussian component of each entry.

One Bayes tree is built per class, therefore we proposed several improvement strategies to decide which tree has the right to refine its model in the next time step. Through extensive experiments we found that refining the k most probable classes (*qbk*) in turns yielded the best results throughout. $k = \min\{2, \lfloor \log(m) \rfloor\}$, where m is the number of classes, showed the best performance on all tested data sets. For more details please refer to [16].

3. BULK LOADING THE BAYES TREE

Our goal in ongoing work is to improve the performance of the Bayes tree. We therefore investigate bulk loading approaches, which we describe in the next Section. Preliminary results are presented in Section 3.2.

3.1 Approaches

Since the Bayes tree is a statistical approach to classification we looked for statistical methods to create a smaller mixture model from a given mixture model. Starting bottom up with a mixture model that contains a kernel estimator for each training set item we create successively coarser models that represent good approximations. We adapted two approaches, a virtual sampling approach described in [21] and a second approach described in [10]. We will describe our approach based on [10], called Goldberger in the following, since it outperformed the first approach.

The Goldberger approach assumes two initial mixture models f and g to be given, where f is the finer model with r components and g an approximation with s components, hence $r > s$. Each component is assigned a weight and is specified by its mean and covariance matrix. To measure the quality of the approximation [10] defines the distance between two mixture densities as follows:

Definition 4. *Let $f = \sum_{i=1}^{r} \alpha_i f_i$ and $g = \sum_{j=1}^{s} \beta_j g_j$ be two mixture densities containing r and s Gaussian components f_i and g_j with their respective weights α_i and β_j. The distance between f and g is then defined using the Kullback-Leibler divergence KL [6] as follows*

$$d(f, g) = \sum_{i=1}^{r} \alpha_i \cdot \min_{j=1}^{s}\{KL(f_i, g_j)\}$$

The optimal mixture model $\hat{g}$ reducing f to s components is $\hat{g} = \arg\min_g(d(f, g))$. Since there is no closed form to compute $\hat{g}$, a local optimum is computed iterating the following two steps until the distance $d(f, g)$ does no longer decrease. Therein $\pi(i) : \{1 \ldots r\} \to \{1 \ldots s\}$ is a mapping function that assigns each component in f to a component in g.

1. **regroup** - update π: $\pi(i) = \arg\min_{j=1}^{s}\{KL(f_i, g_j)\}$
2. **refit** - for each component g_j recompute weight β_j, mean μ_j and covariance matrix Σ_j as follows
   - $\beta_j = \sum_{i,\pi(i)=j} \alpha_i$
   - $\mu_j = \frac{1}{\beta_j} \sum_{i,\pi(i)=j} \alpha_i \mu_i$
   - $\Sigma_j = \frac{1}{\beta_j} \sum_{i,\pi(i)=j} \alpha_i \left(\Sigma_j + (\mu_i - \mu_j)^2\right)$

We devise a bulk loading technique based on [10] as follows. To initialize the mixture g we compute a first mapping π_0 by assigning $0.75 \cdot M$ components from f to one component in g according to the z-curve order of their mean values. M is given through the fanout, which in turn is dictated by the page size.

The components g_j are converted to Bayes tree nodes containing the entries f_i with $\pi(i) = j$. Since the final π might map more than M components from f to a single component in g, we investigated several strategies to restrict the fanout to the given boundaries. First we reformulated the regroup step into an integer linear program with constraints regarding the resulting fanout. However, for realistic problem sizes, this approach took way too long to compute a complete bulk loading. Hence, we decided for a post processing after the mapping π was computed, which splits the nodes that contain too many entries. Therefore two representatives are computed by moving the mean along the dimension with the highest variance by an ϵ in both direction. A Gaussian is placed over the two representatives and the mapping of the entries to the representatives is computed as in the regroup step. If a node contains too few entries it is merged with the node closest to it in term of the Kullback-Leibler divergence.

Besides the above mentioned bottom up approaches we implemented a top down approach that recursively splits the training set into several clusters. In contrast to the previous approach, where Gaussian components were merged and mapped, we now operate solely on the data objects. More precisely, we start by applying the EM [8] algorithm to the complete training set. The desired number M of resulting clusters is always set to the fanout which is again given through the page size. If the EM returns less than m clusters, the biggest resulting cluster is split again such that the total number of resulting clusters is at most M. In the rare case that the EM returns a single cluster, this cluster is split by picking the two farthest elements and assigning the remaining elements to the closest of the two. Finally, if a resulting cluster contains more than L objects (the capacity of a leaf node), the cluster is recursively split using the procedure described above. Otherwise the items contained in that cluster are stored in a leaf node, its corresponding entry is calculated and returned to build the Bayes tree. The EM approach may result in an unbalanced tree, which differs from the primary Bayes tree idea. However, as we will see in the next section, the results show that this is not a drawback but even leads to better anytime classification performance.

Finally we employed traditional R-tree bulk loading algorithms, i.e. we implemented space filling curves like Hilbert curve or z-curve and other partitioning approaches, e.g. sort-tile-recursive [14]. We briefly describe the Hilbert curve approach since we will present its results in the next section. The bulk loading according to the Hilbert curve is a bottom up approach where in the first step the Hilbert value for each training set item is calculated. Next the items are ordered according to their Hilbert value and put into leaf nodes w.r.t. the page size. After that the corresponding entry for each resulting node is created, i.e. MBR, cluster features (CF) and the pointer. These steps are repeated using the mean vectors as representatives until all entries fit into one node, the root node.

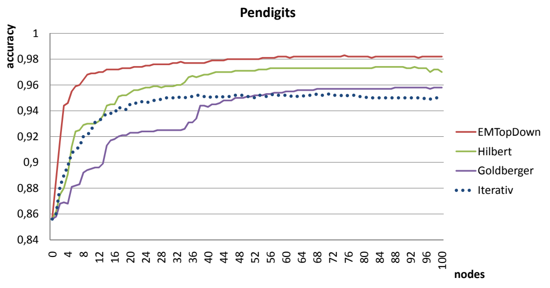


Figure 2: Anytime classification accuracy on pendigits using different bulk loading approaches.

3.2 Results

The data sets that were used in the evaluation are summarized in Table 1. We performed 4-fold cross validation and show the classification accuracy after each node averaged over the four folds. We used global best descent and the *qbk* improvement strategy as they showed the best results in [16] (cf. Section 2.2). The three proposed bulk loading techniques are compared to the previous results from [16] (called *Iterativ* in the graphs).

Figure 2 shows the results for the pendigits data set. The Goldberger approach fails to improve the accuracy over the iterative insertion for the first 50 nodes. After that it performs slightly better, but cannot increase the accuracy more than 1%. The curve corresponding to the Hilbert bulkload shows a steep increase similar to the iterative insertion and shows better performance in most cases. The EMTopDown bulkload outperforms all other approaches and improves the accuracy over the iterative insertion constantly by 3% or more on this data set.

The performance of the Goldberger bulkload stayed below the iterative insertion in the majority of our experiments. Just on the Letter data set it improved the accuracy for larger time allowances. Figure 3 shows the results on Letter. For the first 40 nodes Goldberger and Iterativ perform equally well, after that the accuracy of Iterativ stays behind that of Goldberger. While the Hilbert bulkload shows similar performance to Iterativ, the EMTopDown again constantly yields the best accuracy up to 13% better than the iterative insertion.

Figure 4 shows the results for the gender and covertype data sets. For readability the results for the Goldberger approach are left out. As stated above, its performance was below that of the iterative insertion. For both data sets $k = 2$ for the *qbk* improvement strategy (cf. Section 2.2). The graphs for EMTopDown and Hilbert using the global best descent (*glo*) show an oscillating behavior on both data sets. For comparison we recapitulated the breadth first traversal (*bft*), the results are plotted as well. As was

name	size	classes	features	ref.
Pendigits	10,992	10	16	[12]
Letter	20,000	26	16	[12]
Gender	189,961	2	9	[19]
Covertype	581,012	7	10	[12]

Table 1: Data sets used in our experiments.

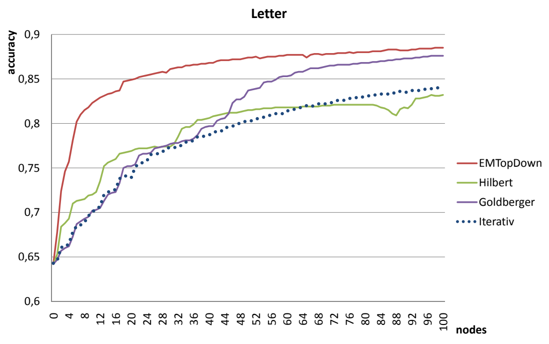


Figure 3: Anytime classification accuracy on letter.

found in [16], the global best descent performs better than breadth first traversal. However, the graphs for bft do not show the oscillating behavior mentioned above. While the reason for the oscillation has to be investigated in further research it does not affect the superiority of the bulk loading over the iterative insertion.

In general the EMTopDown shows the best results in terms of anytime classification accuracy on all tested data sets and continuously improves the accuracy over that of the previous results in [16] up to 13%. This proves the effectiveness of bulk loading for our hierarchical anytime classifier.

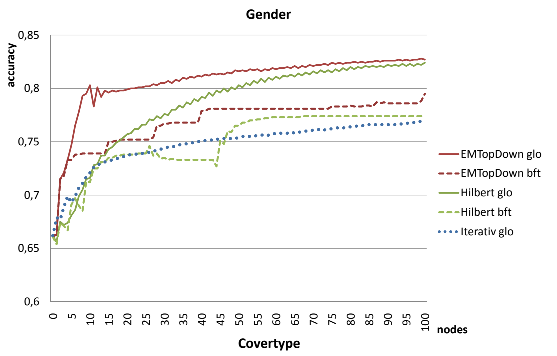
4. FUTURE WORK

The general performance of our anytime classification approach from [16] and the promising results presented in the last section motivate further research and application of our technique. Our goals are first to further explore and improve anytime classification using index structures (cf. Sec. 4.1) and second to extend our approach and exploit index structures for other data mining tasks (cf. Sec. 4.2).

4.1 Next challenges in anytime classification

For the Bayes tree in its current version there is a number of options we plan to investigate. One option is the use of different kernels, e.g. Epanechnikov kernels instead of Gaussian kernels, to test their performance but also the robustness of our general approach. The benefit of employing covariances, i.e. skipping the independence assumption, will be evaluated as well. Moreover, a modification of the Bayes tree to enable its application on data sets containing (or consisting of) categorical data is a further challenge.

A structural modification we are currently investigating regards the separation of the training data according to the class label. So far we build one Bayes tree per class. Now we investigate storing the complete training data in a single Bayes tree and modifying the entry structure such that information about the individual classes can still be obtained. Note that the number of classes does not impose an overhead (cf. *qbk* strategy), since each read node yields an improvement independent of the number of trees. However, the modification yields a parallel refinement of several classes in a single descent and therefore can speed up the improvement. Combining multiple classes in a tree poses several interesting questions: is the descent decision still based entirely on the probability density or is it favorable to include the class distribution into the decision, e.g. through the entropy of the current entries? How is the *qbk* strategy best



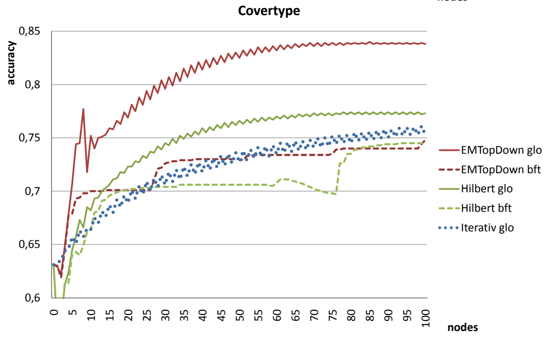


Figure 4: Anytime classification accuracy on gender (top) and covertype (bottom).

adapted to the new structure, i.e. are the k best entries chosen w.r.t. to the posterior probability or are the frequency counts for the different classes taken into account? What is the best trade-off between entry size and classification accuracy, i.e. do we store variances/covariances per class in each node or is variance pooling also a good option? This structural modification also motivates research in the direction of multi-label classification and semi-supervised learning.

Finally we want to investigate further application scenarios. We applied the Bayes tree in health monitoring [13], where we exploited its hierarchical structure to perform multi-step classification. More precisely, we used the upper levels of the trained Bayes trees for pre-classification on mobile devices with restricted resources. Based on the pre-classification the mobile devices sent more or less data to a central server, yielding a varying data stream. On the server the unrestricted Bayes tree was used for full classification and triggered an optional detail request to the mobile devices. We plan to extend this research in multi-step classification, but also elaborate the usage of our approach to other streaming applications, e.g. in sensor networks.

4.2 Extension to other data mining tasks

Besides further research on stream data classification we plan to extend our Bayes tree approach and use index structures for other stream data mining tasks, especially with respect to anytime constraints. A major research direction will be unsupervised learning (stream clustering) in the light of evolving data distributions.

We plan to use our Bayes tree to generate a hierarchical clustering on data streams. In particular the nice properties of the stored cluster features (CF) allow several approaches to modeling and tracking evolving data distributions. Exploiting their temporal multiplicity [2] for example we can decrease the influence of older data in the current representation by an exponential decay function. Moreover, this allows to reuse node entries if their contribution is too insignificant due to their age. As a consequence, we can maintain an up-to-date view on the data distribution in constant space. The additivity property of the CF [2] allows the comparison of data distributions from arbitrary points in time. Applying a pyramidal time frame as in [1] guarantees a moderate memory consumption even for long running applications.

Through the hierarchical nature of the Bayes tree as an index structure we can insert new objects in logarithmic time and hence can maintain a finer cluster representation than previous approaches in the same time. Moreover, using these fine grained CF representation we can find clusters of arbitrary shape by using density based clustering in an offline component as in [5].

A promising research direction in using index structures for anytime stream mining is the extension of the Bayes tree to enable anytime clustering. This can be achieved by modifying the entry structure such that we can "park" insertion objects in inner nodes and take them along in a later descent. Another great benefit of this modification is the property of *self-adaptation*. More precisely, the size of the tree will automatically adapt itself to the stream speed since insertion objects will descent as far as time permits, be parked there and hence no further splits occur.

Further topics include the detection of outliers, handling of missing values and the investigation of a subspace variant of the Bayes tree. In general we believe that the effective usage of index structures for stream data mining as described in [16, 13] and Section 3 can be successfully extended.

5. CONCLUSION

We have presented and applied a novel index-based anytime classifier (Bayes tree) in recent work constituting our first goal in enabling anytime Bayesian classification. Our next goal is the improvement of its performance. To this end we presented ongoing work in this paper wherein we improved the anytime classification accuracy up to 13% through different bulk loading approaches. Moreover, we have laid out various further research aspects regarding index-based stream classification. The third goal is the extension of our technique to other data mining tasks. Hereto we identified approaches to use index structures for modeling evolving data streams or even for anytime clustering.

Acknowledgments

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

6. REFERENCES

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *29th VLDB*, 2003.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for projected clustering of high dimensional data streams. In *30th VLDB*, 2004.

[3] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. On demand classification of data streams. In *10th ACM KDD*, pages 503–508, 2004.

[4] B. Arai, G. Das, D. Gunopulos, and N. Koudas. Anytime measures for top-k algorithms. In *33rd VLDB*, 2007.

[5] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SDM*, 2006.

[6] J.-Y. Chen, J. Hershey, P. Olsen, and E. Yashchin. Accelerated monte carlo for kullback-leibler divergence between gaussian mixture models. In *ICASSP*, 2008.

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

[8] A. P. Dempster, N. M. L. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[9] S. Esmeir and S. Markovitch. Anytime induction of decision trees: An iterative improvement approach. In *21st AAAI*, 2006.

[10] J. Goldberger and S. T. Roweis. Hierarchical clustering of a mixture model. In *NIPS*, 2004.

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

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

[13] P. Kranen, D. Kensche, S. Kim, N. Zimmermann, E. Müller, C. Quix, X. Li, T. Gries, T. Seidl, M. Jarke, and S. Leonhardt. Mobile mining and information management in healthnet scenarios. In *9th IEEE MDM*, 2008.

[14] S. T. Leutenegger, J. M. Edgington, and M. A. Lopez. Str: A simple and efficient algorithm for r-tree packing. In *ICDE*, pages 497–506, 1997.

[15] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *28th VLDB*, 2002.

[16] T. Seidl, I. Assent, P. Kranen, R. Krieger, and J. Herrmann. Indexing density models for incremental learning and anytime classification on data streams. In *12th EDBT/ICDT*, 2009.

[17] A. Silberstein, A. Gelfand, K. Munagala, G. Puggioni, and J. Yang. Suppressions and failures in sensor data: A bayesian approach. In *33rd VLDB*, 2007.

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

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

[20] K. Ueno, X. Xi, E. J. Keogh, and D.-J. Lee. Anytime classification using the nearest neighbor algorithm with applications to stream mining. In *ICDM*, 2006.

[21] N. Vasconcelos and A. Lippman. Learning mixture hierarchies. In *NIPS*, pages 606–612, 1998.

[22] M. Vlachos, J. Lin, E. J. Keogh, and D. Gunopulos. A wavelet-based anytime algorithm for k-means clustering of time series. In *Workshop on Clustering High Dimensionality Data and Its Applications*, 2003.

[23] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *9th ACM KDD*, 2003.

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