Learning Probabilistic Transfer Functions: A Comparative Study of Classifiers

K. P. Soundararajan and T. Schultz
University of Bonn, Germany

Abstract

Complex volume rendering tasks require high-dimensional transfer functions, which are notoriously difficult to design. One solution to this is to learn transfer functions from scribbles that the user places in the volumetric domain in an intuitive and natural manner. In this paper, we explicitly model and visualize the uncertainty in the resulting classification. To this end, we extend a previous intelligent system approach to volume rendering, and we systematically compare five supervised classification techniques – Gaussian Naive Bayes, k Nearest Neighbor, Support Vector Machines, Neural Networks, and Random Forests – with respect to probabilistic classification, support for multiple materials, interactive performance, robustness to unreliable input, and easy parameter tuning, which we identify as key requirements for the successful use in this application. Based on theoretical considerations, as well as quantitative and visual results on volume datasets from different sources and modalities, we conclude that, while no single classifier can be expected to outperform all others under all circumstances, random forests are a useful off-the-shelf technique that provides fast, easy, robust and accurate results in many scenarios.

Categories and Subject Descriptors (according to ACM CCS): I.4.6 [Image Processing and Computer Vision]: Segmentation—Pixel classification

1. Introduction

Interaction techniques and user interfaces for intuitive transfer functions design continue to pose challenges to direct volume rendering research [MJW’13, IVJ12]. Reliable classification in complex datasets frequently requires high-dimensional transfer functions, which might include information from spatial derivatives [KKH02], about the spatial position and intensity values in spatial neighborhoods [TLM05], size [CM08], or texture [CR08].

Traditional transfer function design is performed directly in the abstract transfer function domain, which is difficult to grasp especially for inexperienced users. Dual-domain interaction [KKH02] can be a valuable aid, showing the user which locations in transfer function space correspond to specific spatial structures and vice versa, and allowing him or her to incrementally build a multidimensional transfer function by combined use of interaction widgets in the spatial and the transfer function domains. However, even when such
additional tools are available, it is very difficult to form a mental image of feature space, and to explore it in a systematic manner, as soon as it has more than three dimensions.

One solution to this is to shift interaction completely to the spatial domain, which is most familiar and intuitive to the user. Our work follows such an approach, in which users can brush examples of the different materials they would like to classify directly in the volume, and supervised machine learning is applied to infer a probabilistic multi-material transfer function from that input. A particular focus of our paper is on choosing a classification technique that is most suitable for this particular task.

After reviewing related work in Section 2, Section 3 establishes the requirements that probabilistic transfer functions impose on the classifier. Section 4 reviews five popular techniques from the machine learning literature and explains to which extent they meet those requirements. Section 5 reports the results of three case studies, each representing one difficulty frequently met in real-world volume rendering applications, in which the five classifiers have been compared to each other. The paper concludes in Section 7 with a clear recommendation and an outlook on the perspectives opened up by learning-based transfer function design.

2. Related Work

The painting metaphor proposed by Tzeng et al. [TLM05], which has been highlighted as “an excellent example of [...] transfer-function research” by others [EHK’06], has inspired our work. However, our approach differs from the one by Tzeng et al. in several important aspects, including explicit probabilistic modeling to enable uncertainty visualization and the use of multi-material classifiers.

A key contribution of our work is a systematic comparison of five supervised classification techniques. Unlike a recent comparison from the machine learning literature [FDCBA14], which has focused on accuracy, our evaluation accounts for the specific requirements of learning transfer functions, which includes factors such as computational speed, ease of parameter selection, and robustness to unreliable user input. In a similar spirit, Möbrets et al. [MVvW05] have compared clustering methods with respect to their applicability in diffusion tensor imaging.

Other visualization systems that have shifted transfer function design to the volumetric domain include one by Teistler et al. [TBL⁺07], which is based on a simple region growing approach; one by Ropinski et al. [RPSH08], which generates one-dimensional transfer functions through histogram analysis; and the WYSIWYG Volume Visualization tool by Guo et al. [GMY11], which allows the user to place scribbles directly onto the volume rendered image.

Related less directly, but still relevant are approaches that use unsupervised learning. They include an earlier work for high-dimensional transfer function design by Tzeng et al. [TM04], systems by Šereda et al. [ŠVG06] and Nguyen et al. [NTCO12], which cluster in the two-dimensional low-high (LH) space, and a system by Ip et al. [IVJ12], which focuses on hierarchical exploration of the two-dimensional intensity-gradient space.

Finally, a different type of learning-based transfer function design has been proposed by Rezk Salama et al. [RSKK06], who create a statistical model from a set of transfer functions for similar datasets, in order to facilitate rapid adaptation to new data.

3. Analysis of Requirements

In this section, we argue that classifiers for learning transfer functions should allow for probabilistic multi-material classification, that they should not require time-consuming parameter selection, and that they should be efficient and accurate, even when provided with unreliable input.

3.1. Probabilistic Classification

The original intelligent system approach by Tzeng et al. [TLM05] defines transfer functions as mapping a vector \( \mathbf{x} \), which contains all relevant data attributes, directly to optical properties such as color or opacity. Even though this agrees with standard references on volume rendering [EHK’06], the direct volume rendering paper by Drebin et al. [DCH88] takes an alternative, explicitly probabilistic view, in which the transfer function is defined as a composition of two functions: The first maps data attributes to material probabilities \( P(y|x) \), where the values of \( y \) are labels from a discrete set \( \mathcal{L} \), one per material. In a second step, each material is assigned a color and opacity, and a weighted average of those is computed based on the material probabilities.

We prefer this latter view in our own work, since the need for visualizations that reflect uncertainty is increasingly being recognized in our community [JS03, BHJ⁺14], and explicitly probabilistic transfer functions are required for uncertainty visualization approaches such as the ones by Kniss et al. [KUS’05] or Lundström et al. [LLPY07], as well as for automatic identification of locations that would benefit most from the user’s attention [PRH10].

An analogous dichotomy exists between supervised classification approaches: Often, classification is defined as learning a function \( f : \mathbb{R}^d \rightarrow \mathcal{L} \) that maps the \( d \)-dimensional feature vector \( \mathbf{x} \in \mathbb{R}^d \) to a single material label from \( \mathcal{L} \). Probabilistic classification instead learns a probability vector \( f : \mathbb{R}^d \rightarrow \mathbb{R}^{\left|\mathcal{L}\right|} \), so that each material is assigned a non-negative probability \( P(y|x) \), and probabilities of all materials – including a null or background material – sum to one.

This establishes the first requirement for our theoretical analysis in Section 4: We are looking for supervised machine learning techniques that permit probabilistic classification.
3.2. Multi-Material Classification

Most real-world volume rendering tasks require distinguishing between several different materials. However, some widely used supervised classification techniques can only distinguish between two classes (i.e., background and a single material). Tzeng et al. [TLM05] work around this by training a separate classifier for each material, and rendering each one in a separate rendering pass.

Unfortunately, multi-pass rendering does not easily allow for correct rendering of semi-transparent materials, and is incompatible with our probabilistic approach, in which probabilities for all materials are needed simultaneously for the second stage of the transfer function evaluation. This establishes the second requirement for our theoretical analysis: We are looking for classifiers that support multiple classes.

3.3. Easy Parameter Tuning

Most supervised classification techniques have parameters that need to be tuned to the data in order to obtain fast and accurate results. This has not been mentioned by Tzeng et al. [TLM05], who may have found it less crucial for creating an initial proof-of-concept implementation.

However, in practice, it will be important to ensure that classification accuracy does not suddenly drop, or computational time increase drastically, due to parameter settings that are far from optimal for the given data. This establishes the first criterion for our experimental analysis in Section 5: We are looking for classifiers that perform well over a wide range of parameter values.

3.4. Accuracy, Robustness, and Speed

Finally, transfer functions should of course classify materials correctly, even if they have been learned from few scribbles, or if the user has not placed them carefully and some of them extend into different materials. Another important point for experimental investigation is speed: Interactive response times are particularly important since we would like to allow the user to incrementally update the initial transfer function, in order to correct its results where needed.

Parameter tuning is interwoven with speed and accuracy: Cross-validation is an established method for automatic parameter selection [HTF11], but its computational cost can be prohibitive. Therefore, our experimental comparison proceeds in two steps: In Section 5.1, we first study how speed and accuracy of the different classifiers vary with parameter settings for three different volume datasets. Based on this, we develop custom strategies for parameter selection, which we use in the final comparison in Section 5.2.

4. Theoretical Analysis of Classifiers

We now apply the requirements from the previous section in a theoretical comparison of five popular supervised classification techniques, specific to the needs of learning transfer functions for direct volume rendering.

4.1. Naive Bayesian Classification

The most direct way to obtain material probabilities $P(y|x)$, as required in Section 3.1, is to use Bayes’ rule: Based on $P(x|y)$, the conditional likelihood of the data given the label, as well as the prior probability $P(y)$ of the label, we obtain

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$  \hspace{1cm} (1)

The main difficulty in using Eq. (1) is in estimating $P(x|y)$, the joint probability of all data attributes given the material. With increasing dimension $d$, a given set of training data $x_i \in \mathbb{R}^d$ quickly becomes sparse, making it impossible to reliably estimate $P(x|y)$; this is known as the curse of dimensionality [HTF11]. Naive Bayes classifiers avoid this problem by making the “naive” assumption that all components $x_j$ of $x$ are distributed independently. Naive Bayes classifiers have been found to perform well in practice even in many cases where this assumption is violated [Zha04].

In our comparison, we include a Gaussian naive Bayes classifier: It makes the additional assumption that all $x_j$ are Gaussian distributed. This simple model intrinsically supports multi-material classification and is free from parameters. Finally, training and classification are fast. Training simply amounts to computing the means $\mu_{lj}$ and variances $\sigma_{lj}$ per label $l$ and data attribute $j$, as well as the relative frequency of each material, which defines its prior $P(y)$; classification is done by computing $P(y|x)$ in closed form.

4.2. k Nearest Neighbor Classifier

The $k$ Nearest Neighbor ($k$-NN) classifier [HTF11] is a non-parametric method which lifts the assumptions of Gaussianity and independence, but is still quite simple: In order to classify a vector $x$, it finds, among all training data $x_i$, the $k$ samples closest to $x$ and lets them vote on the label. A straightforward probabilistic variant reports the relative frequency of each label among the $k$ nearest neighbors.

$k$-NN intrinsically supports multiple classes and has a single parameter $k$, which provides a tradeoff between reducing noise (large $k$) and the ability to adapt to small structures in the input (small $k$). In order to avoid discontinuous probability estimates — with standard $k$-NN, they can only take on discrete values $n/k$ to $n$ — we weight each vote by the inverse distance of the corresponding sample $x_i$.

Unlike all other methods covered here, $k$-NN does not compute an abstraction from the training data. Therefore, it does not strictly require a training phase. Our implementation still builds a k-d tree during training, in order to find the $k$ nearest neighbors more quickly later on.
4.3. Support Vector Machines

The basic idea of support vector machines (SVMs) is to find a hyperplane that separates samples from two classes such that the distance between the hyperplane and the closest sample is maximized, thus minimizing the risk of misclassifying new samples when using the hyperplane as a classification boundary. For practical use, this idea is extended in two ways: First, a tunable degree of misclassification in the training data (“slack”) is permitted, in order to avoid adapting the classification boundary to errors in the input (“overfitting”). Second, in many cases, the decision boundary cannot be expected to be a hyperplane. This is dealt with by an implicit nonlinear mapping of the input to a higher-dimensional space in which it becomes possible to use a simple hyperplane (“kernel trick”). For details, we refer to [SS02].

Benefits of SVMs include the fact that they can be formulated as quadratic programs with a unique solution; the fact that the resulting classifiers only depend on a sparse subset of the training data (the “support vectors”), which makes them cheaper to evaluate; and the fact that they have proven to produce state-of-the-art accuracy in a range of applications. For these reasons, SVMs were previously considered for learning transfer functions [TLM05].

In their original formulation, SVMs only predict a single class label. To learn probabilistic transfer functions, we employ an established extension that permits estimation of class probabilities, at the expense of a more costly training phase [WLW04]. Similarly, standard SVMs only deal with two classes. Several strategies exist for applying SVMs to multi-class data. Among those, we have chosen “one-against-one”, which combines votes from binary classifiers trained for each pair of classes. For $k$ classes, this requires $k(k - 1)/2$ classifiers. A comparison to “one-against-rest”, which only requires $k$ classifiers, has shown that “one-against-one” is still faster overall in practice, at a similar accuracy [HL02]. This can be explained by the fact that its individual training sets are much smaller.

Kernel SVMs have at least two parameters: The first is a regularization parameter, commonly denoted $C$, steering the degree of misclassification which is permitted during training. The level of noise in the input affects its optimal value. The second is the choice of kernel which maps the input to a higher-dimensional space. In our experiments, we use the popular radial basis function $K(x, x') = \exp(-\gamma ||x - x'||^2)$. It introduces another parameter $\gamma$, whose optimal value depends on the scale of the structures formed by the samples.

4.4. Neural Networks

Single hidden layer perceptrons, a specific type of neural network, are the second classifier that was employed to learn transfer functions by Tzeng et al. [TLM05]; therefore, we also include them in our comparison. Perceptrons consist of $h$ hidden nodes, which combine the input components $x_j$ into features $z_k$ by taking a weighted sum with weight vector $a_k$, adding a bias term $a_k$, and taking the sigmoid function $\sigma(v) = 1/(1 + \exp(-v))$ of the result,

$$z_k(x) = \sigma(a_k + a_k^T x).$$

(2)

For binary classification, a single output node combines the $z_k$ in an analogous fashion,

$$f(x) = \sigma(b_0 + b^T z(x)).$$

(3)

where $f(x)$ can be interpreted as a single class probability. Perceptrons are typically trained by back-propagation: Weights are initialized randomly, and $f(x_i)$ is computed iteratively for the training samples $x_i$, each time updating the weights $a_k$ and $b_k$ to reduce deviation from the true label $y_i$.

Perceptrons generalize to multi-class data by replacing the single output node with a separate node per label, and having each of them compute the corresponding class probability $f_j(x)$ using the softmax function [HTF11]:

$$f_j(x) = \frac{e^{s_j}}{\sum_{l=1}^C e^{s_l}} \text{ with } s_j(x) = b_{0j} + b_j^T z(x).$$

(4)

The main parameters of perceptrons are the number $h$ of hidden nodes, which controls their complexity, and the choice of nonlinearity. Like Tzeng et al. [TLM05], we select the sigmoid.

4.5. Random Forests

Binary decision trees are a popular yet simple classifier: In order to classify a sample $x$, starting at the root, each node selects a component $x_j$ and compares it to a threshold $\theta$. If $x_j \leq \theta$, $x$ is passed down the left sub-tree; otherwise, it is passed down the right sub-tree. Each leaf node carries a set of class probabilities that is assigned to vectors reaching it.

Decision trees are constructed by recursively splitting a set of training data so that the distribution of labels in the two resulting sets is purer than in the original set. To quantify this, our work uses Gini impurity $I_G$. Given a set in which a fraction $F_i$ of all samples has label $i$, $I_G$ is given as

$$I_G = \sum_{i=1}^C F_i(1 - F_i) = 1 - \sum_{i=1}^C F_i^2.$$

(5)

Each node selects a variable $x_j$ and a threshold $\theta$ so that Gini impurity of its children is minimized.

Averaging over the decisions of many trees boosts the accuracy of the overall result if each tree has at least weak predictive power and there is little correlation between them. A forest of trees with these properties can be grown by randomly sampling – with replacement – the training data used while constructing each tree, and by only considering a random subset of components $x_j$ as candidates for thresholding at each node. This is the idea behind random forests [Bre01].
We include random forests in our comparison since they easily support probabilistic multi-material classification, they have previously been found to provide high accuracy [FDCEA14], even in the presence of label noise, and they can be constructed and applied quickly and in parallel, which is appealing for interactive use in visualization.

The two parameters of random forests that are usually cited to be the most relevant, and will be included in our experimental comparison, are the number of trees, and the number of features that are considered when constructing a node. However, we found that another parameter, which is often fixed to a heuristic default [HTF11], has an important effect on the quality of the resulting visualization.

For classification, it is common to continue splitting impure training sets until they only contain a single class. We found that this leads to overly confident predictions. Fig. 2 (a) shows an example of a discontinuity artifact that results from this, at a point where the classifier abruptly switches from one class to the other, rather than providing gradually varying class probabilities. As shown in Fig. 2 (b), disallowing splits that would lead to children with fewer than a minimum number of samples, which is common practice when using random forests to predict continuous variables [Bre01], helps avoid this problem. All our experiments impose a minimum number of eight samples per leaf.

5. Experimental Comparison of Classifiers

Our experimental comparison proceeds in three steps: First, we examine how speed and accuracy of individual classifiers depend on their parameters in order to develop strategies for parameter selection. The results of this investigation are reported in Section 5.1. We then use the developed strategies within a quantitative comparison of the classifiers (Section 5.2) and a visual comparison of the results (Section 5.3).

We selected four datasets that exemplify different problems frequently met in practice: The well-known bonsai CT dataset (256 × 256 × 256 voxels) with four material classes air, leaves, wood, and bowl has been selected because it suffers from reconstruction artifacts that should be suppressed in the visualization. A T1 weighted head MRI scan (256 × 256 × 220) with five material classes air, skin, bone, as well as gray and white matter of the brain is included because it is notoriously difficult to reliably differentiate between brain tissue and the surrounding structures. A low-dose chest CT (256 × 256 × 138) that simultaneously suffers from severe noise, reconstruction artifacts, and overlapping intensity ranges for the classes background, bone, lung, and other tissues, was used to test the limits of our learning-based approach. Finally, a CT scan of a tooth (103 × 94 × 161) with classes air, dentin, enamel and pulp has been included, because it has exemplified multi-material classification in many previous publications [KKH02, TM04, IVJ12]. Since this dataset is relatively clean, we use it as a test case for imperfect labels, mimicking a sloppy user who sometimes places scribbles so that they extend into another material. An example of this is shown in Fig. 6 (a).

Training data was generated by placing scribbles for each material on slice views. The exact numbers of training samples are given in the first column of Table 1. Each voxel was represented by an 11-dimensional feature vector \( \mathbf{x} \) that contained local intensity and gradient magnitude, intensities in six face-connected neighbors, as well as the voxel coordinates [TLM05]. We subtracted the mean from all features and re-scaled them to unit variance; this is a widely used type of normalization, since features that vary over a large numerical range otherwise dominate the results of many classifiers.

We used existing and well-tested implementations of all classifiers: Naive Bayes, \( k \) Nearest Neighbors, SVM, and Random Forests have all been taken from the widely used scikit-learn toolbox [PVG+11]. Since it does not support Neural Networks, we used PyBrain [SBW10] for those.

5.1. Impact of Parameters

We repeatedly trained all classifiers with a large range of possible parameter values, and monitored their accuracy. It is important not to evaluate classifiers on their training data, since this might mask the effects of overfitting and lead to unrealistically optimistic performance estimates. Therefore, we generated independent test sets, each containing at least 10,000 samples, by placing scribbles on slices different from the ones used for training. We only included voxels in the test data for which it was obvious to a human observer which material they represent. For a classifier to be useful in our context, we consider it necessary that it performs well on those selected "obvious" cases; for a full evaluation, quantitative results will be complemented by a visual inspection on the whole datasets in Section 5.3.

Quality is measured in terms of precision and recall: For a given material \( A \), precision \( P \) is defined as the fraction of voxels classified as \( A \) that are truly \( A \). In terms of True and False Positives, \( P = TP/(TP + FP) \). Recall \( R \) is the fraction of voxels of material \( A \) that were correctly classified as such. In terms of True Positives and False Negatives, \( R = TP/(TP + FN) \). Precision and recall are both important in our context, but plotting both would lead to visual clutter.
Therefore, we combine them into the established F score

$$F = 2 \times \frac{P \times R}{P + R}.$$  \hspace{1cm} (6)

If precision and recall are both perfect, $F = 1$; if one of them is zero, $F = 0$. $F$ is computed separately for each material. Fig. 3 plots the mean of $F$ over all materials, with error bars indicating one standard deviation. Fig. 4 additionally plots the effects of selected parameters on training and classification time.

Fig. 3 (a) shows that it depends on the dataset what number $k$ in the $k$ nearest neighbors classifier works best: In case of the tooth, a larger $k$ averages out noise in the data; for the brain, increasing $k$ reduces accuracy since it starts to include many samples from other materials. Therefore, we decide to use five-fold cross-validation on the training data to tune the value of $k$ to the specific dataset. Since Fig. 4 (a) shows that computational effort increases noticeably with large values of $k$, we limit ourselves to the values $k \in \{4, 16\}$.

Accuracy of the Support Vector Machine depends strongly on the regularization parameter $C$ in Fig. 3 (b): Strong regularization through smaller $C$ works better in the presence of label noise in the tooth dataset, while it oversmoothes the complex brain data. Due to feature normalization, the default bandwidth parameter $\gamma = 1/d$ proposed by scikit-learn provides reasonable results. Note that in Fig. 3 (c), $C = 1$ is fixed; while the brain dataset benefits from a $\gamma$ that is larger than the default in this plot, even better results are obtained using the default $\gamma$ at higher $C$. The training cost of SVM makes joint cross-validation of $C$ and $\gamma$ prohibitive in interactive applications. Therefore, we decide to stick to the default $\gamma$ and only use two-fold cross-validation to choose between $C \in \{1, 1000\}$. 

---

**Figure 3:** Classification accuracy as a function of the number $k$ of nearest neighbors (a); regularization parameter $C$ (b) and bandwidth parameter $\gamma$ (c) of a support vector machine; the number of hidden nodes in a single layer perceptron (d); and the number of trees (e) and features per node (f) in a random forest.

**Figure 4:** Time needed for training and classification as a function of the number $k$ of nearest neighbors (a); the number of hidden nodes in a single layer perceptron (b); and the number of trees in a random forest (c).
Table 1: Accuracy achieved by all classifiers after training on example sets of the given sizes. Specified values are mean and standard deviation of F scores over all classes. Best values and ones within a margin of 0.01 of it are highlighted in boldface.

<table>
<thead>
<tr>
<th></th>
<th>GNB</th>
<th>k-NN</th>
<th>SVM</th>
<th>SLP</th>
<th>RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bonsai (32910 samples)</td>
<td>0.83 ± 0.24</td>
<td><strong>0.99 ± 0.01</strong></td>
<td><strong>0.99 ± 0.01</strong></td>
<td>0.87 ± 0.21</td>
<td><strong>0.99 ± 0.01</strong></td>
</tr>
<tr>
<td>Brain (6285 samples)</td>
<td>0.83 ± 0.13</td>
<td><strong>0.94 ± 0.05</strong></td>
<td><strong>0.93 ± 0.07</strong></td>
<td>0.83 ± 0.13</td>
<td><strong>0.93 ± 0.05</strong></td>
</tr>
<tr>
<td>Chest (71138 samples)</td>
<td>0.69 ± 0.40</td>
<td><strong>0.94 ± 0.05</strong></td>
<td>0.91 ± 0.07</td>
<td>0.73 ± 0.37</td>
<td><strong>0.99 ± 0.01</strong></td>
</tr>
<tr>
<td>Tooth (13243 samples)</td>
<td>0.90 ± 0.14</td>
<td><strong>0.97 ± 0.03</strong></td>
<td>0.94 ± 0.03</td>
<td>0.96 ± 0.06</td>
<td><strong>0.98 ± 0.01</strong></td>
</tr>
</tbody>
</table>

Table 2: Time (in seconds) it took to train the classifier and to apply it to the full volume dataset.

<table>
<thead>
<tr>
<th></th>
<th>GNB</th>
<th>k-NN</th>
<th>SVM</th>
<th>SLP</th>
<th>RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bonsai (32910 samples)</td>
<td>&lt; 0.1 + 4.7</td>
<td>3.5 + 101.7</td>
<td>6.0 + 72.9</td>
<td>16.8 + 276.9</td>
<td>0.7 + 7.4</td>
</tr>
<tr>
<td>Brain (6285 samples)</td>
<td>&lt; 0.1 + 5.1</td>
<td>0.6 + 37.6</td>
<td>0.9 + 30.3</td>
<td>3.3 + 233.0</td>
<td>0.1 + 6.9</td>
</tr>
<tr>
<td>Chest (71138 samples)</td>
<td>&lt; 0.1 + 2.3</td>
<td>13.2 + 87.5</td>
<td>45.1 + 166.6</td>
<td>36.6 + 145.8</td>
<td>2.5 + 5.0</td>
</tr>
<tr>
<td>Tooth (13243 samples)</td>
<td>&lt; 0.1 + 0.3</td>
<td>1.2 + 6.8</td>
<td>1.2 + 1.9</td>
<td>6.7 + 25.3</td>
<td>0.3 + 0.8</td>
</tr>
</tbody>
</table>

Fig. 3 (d) shows that single layer perceptrons require a certain complexity, corresponding to a minimum number \( h \) of hidden nodes, to perform good classification. However, they saturate at some point. We fix \( h = 32 \), which worked well in all cases and was not slower than networks with smaller \( h \) (Fig. 4 (b)).

Finally, Fig. 3 (e) shows how performance of the Random Forest classifier depends on the number of trees. Again, a minimum number of trees is required for good accuracy. However, the additional benefit quickly becomes marginal compared to the much larger computational expense of using many trees, shown in Fig. 4 (c). This led us to limit ourselves to 32 trees. Since the number of features considered at each node did not greatly affect accuracy (Fig. 3 (f)), we stick to the default in scikit-learn, \( \sqrt{d} \).

### 5.2. Accuracy and Speed

Accuracies of all five classifiers with the parameter selection schemes derived in the previous section are reported in Table 1; timings for training and classification are in Table 2.

- The Gaussian Naive Bayesian (GNB) classifier is fast, but lacks a mechanism to reduce the impact of outliers in the tooth example, and its oversimplifying assumptions limit performance on the more complex datasets.
- \( k \)-Nearest Neighbor (k-NN) classification provides good accuracy, but its computational cost makes it less attractive for large volumes. In particular, the lack of abstraction from the training data makes classification slow.
- Due to the complexity of the optimization problem that is solved when training Support Vector Machines (SVM), training is slow on large inputs. Classification is sometimes faster than in k-NN, depending on the level of sparsity achieved during training. Overall, SVMs provide reasonable accuracy in all cases.
- The Single Layer Perceptron (SLP) was usually the slowest technique, even though its complexity scales more favorably with the size of the training data. At the same time, its accuracy was not competitive.

- In contrast, Random Forests (RF) produced competitive results in all cases, and were always faster than all methods except the overly simplistic Naive Bayesian approach.

Regarding the timings in Table 2, we note that there was significant overhead from communicating with the scikit-learn and PyBrain frameworks, and from their limited ability for parallel processing. We expect that, by directly interfacing to a GPU-based implementation of random forests [Sha08], interactive performance could be achieved. However, optimizing computational performance of any particular classifier is outside the scope of our current work, which focuses on a comparative analysis.

### 5.3. Visual Comparison

Visualizing probabilistic classification results requires mapping material probabilities to color and opacity. The most traditional approach to this involves probability-based averaging [DCH88]. In Fig. 5 (a), this method illustrates that, even though the reconstruction artifact was fully suppressed by the random forest classifier in terms of the most likely
material (c), it continues to be a source of uncertainty. Risk-based volume rendering (b) [KUS05] creates a middle ground between these two extremes, and adds flexibility by defining importance weights for all materials. Compared to the risk-based approach with equal weights, Fig. 1 (e) removes the remaining traces of the artifact in Fig. 5 (b) by increasing the importance of the background. Risk-based rendering was also used to create all other figures.

It is clear from Fig. 1 (a) that Gaussian Naive Bayes correctly captures the intensity ranges of the different materials, but the oversimplifying Gaussianity and independence assumptions prevent it from successfully using the spatial information to suppress the reconstruction artifact.

The probabilities estimated by the Single Layer Perceptron in Fig. 1 (d) are too fuzzy to be useful for risk-based rendering. The fact that neural networks were previously successfully used for volume rendering can be explained by the fact that Tzeng et al. [TLM05] only rendered the most likely material. We expect that sharper results can be achieved by repeatedly passing the same samples through the network during training. However, given the high computational cost of SLP even without this, we did not pursue this idea further.

The remaining classifiers produce reasonable results, even though the k nearest neighbors classifier fails to suppress some artifacts in proximity of the crown of the tree. We did not mark the ground as a separate material, so the fact that it gets classified differently does not speak in favor of any particular method. Due to the unfavorable results of GNB and SLP, which is evident also for the other datasets from Table 1, we only present visual results for k-NN, SVM and RF in the other three datasets.

Fig. 6 (a) shows a slice of the tooth data, with the scribbles that served as training samples superimposed in the same colors that are later used for rendering. These annotations imitate an imperfect user, who starts by carefully annotating the pulp (red), but then grows impatient and puts overly rough sketches on the remaining materials, leading – among other inaccuracies – to parts of the tooth being labeled as air and vice versa. Fig. 6 (b)–(d) map classification results of three classifiers on the same slice, with probabilistic averaging of material colors.

The volume renderings in Fig. 6 (e)–(g) confirm that SVM is most severely affected by the sloppy scribbles. This problem could be removed by correcting them, or by adding a large enough number of accurate new scribbles. In k-NN, a small artifact appears on the surface of the enamel, and some of the air above the tooth is misclassified as pulp (blue arrows). Even though the latter problem also appears in the slice of the random forest result (d), it is weak enough to be suppressed by increasing the importance weight of the background in the risk-based rendering (g). Identical importance weights have been used with all classifiers.

Fig. 7 confirms that, on the brain MRI, all three classifiers successfully used spatial information to mask out the overlapping intensities of bone and surrounding tissues, and give a clear view on the white matter (rendered in red). Scribbles were only placed at the center of the brain, and all classifiers reasonably extrapolate to other locations. However, it is evident from the sharp creases in the rendering of the white matter surface in Fig. 7 (b) that some of the outermost parts of the white matter are missing in the SVM result.
Finally, Fig. 8 shows that, even with transfer functions generated from a large number of training samples (~0.8% of the volume) using state-of-the-art learning techniques, the high levels of noise and reconstruction artifacts continue to pose a serious challenge for volume rendering raw data from low-dose chest CT. Even though the random forest (c) provides a reasonable overview of the lungs and bones, the segmentation is incomplete, and the quality of the other classifiers suffers even more drastically. We expect that fully satisfactory results on such data can only be achieved by additional pre-processing, such as anisotropic filtering or spatially regularized volume segmentation. However, tailoring such techniques to this particular case is outside the scope of our current comparison.

6. Discussion

From our comparison of five widely used machine learning techniques with respect to the requirements of learning transfer functions, we draw the following conclusions:

- Due to its restrictive assumptions of Gaussianity and statistical independence of individual features, the Gaussian Naive Bayes classifier is not flexible enough to adapt to complex spatial structures (Fig. 1 (a)).
- Its long training times, relatively low accuracy, and fuzzy class probabilities that prevent clean renderings make Single Layer Perceptrons unattractive for learning transfer functions (Fig. 1 (d) and Tables 1 and 2).
- Given a significant number of scribbles, Support Vector Machines do not allow for interactive visualization, since their training scales unfavorably with the size of the training data (Table 2). Moreover, SVMs are quite sensitive to their tuning parameters (Fig. 3), and visual performance in Fig. 6 and Fig. 8 was clearly worse than for the remaining two classifiers.

Both remaining classifiers produce useful results. Certainly, we cannot expect any single method to always outperform all others. However, when given the choice between two classifiers that achieve acceptable accuracy, we prefer the one that allows for faster visual feedback, making it easier for the user to interactively correct any potential remaining misclassifications.

We found that random forests are faster than the $k$ nearest neighbor classifier, sometimes by an order of magnitude or more (Table 2). A further speedup can be achieved by building the randomized decision trees in parallel. Random forests are also easier to use since, in all our examples, they have led to useful results with default settings, while $k$-NN requires data-driven tuning of the number of neighbors $k$ for optimal performance (Fig. 3 (a)). A limitation of random forest lies in the fact that, due to thresholding at each node, their estimates are discontinuous. However, our results indicate that visual artifacts from this can be minimized by imposing a minimum number of samples per leaf (Fig. 2).

7. Conclusion and Future Work

In this work, we have proposed to explicitly model the uncertainty in transfer functions that have been learned from user-provided scribbles in the volumetric domain. We have also compared five supervised classification techniques with respect to their suitability for this task. Based on our results, we recommend random forests as a fast, accurate, robust, and easy-to-use technique. Even though its properties make it attractive for use in an interactive context, this classifier has not been used widely in the visualization community.

Learning probabilistic transfer functions connects volume visualization to many current trends in the machine learning community: In the future, we plan to integrate approaches from active learning, in which the computer directs the user’s attention to regions in which more training data is required [PRH10]; from online learning, where new training data is integrated without having to start training from scratch [SLS’09]; and from transfer learning, which will allow us to use scribbles from different data sets to learn new transfer functions with less interaction. We consider our present work as another step towards more sophisticated visualization systems that will establish intuitive and smart example-based user interfaces [Ma07].

Acknowledgements

We would like to thank Max Hermann (University of Bonn) for providing a volume rendering framework; Jürgen Gall (University of Bonn) for a discussion on random forests; Tobias Schmidt-Wilcke (University Hospital Bergmannsheil, Bochum, Germany) for providing the head MRI dataset; S. Roettger (University of Stuttgart, Germany) for making the Bonsai dataset available via volvis.org; the SCI Institute (University of Utah, USA) for the tooth dataset; Cornell Medical Center for the chest CT image (taken from the VIA/I-ELCAP public lung database).

References


