Visualizing class probability estimators

Eibe Frank and Mark Hall

Department of Computer Science
University of Waikato
Hamilton, New Zealand
{eibe, mhall}@cs.waikato.ac.nz

Abstract. Inducing classifiers that make accurate predictions on future data is a driving force for research in inductive learning. However, also of importance to the users is how to gain information from the models produced. Unfortunately, some of the most powerful inductive learning algorithms generate "black boxes"—that is, the representation of the model makes it virtually impossible to gain any insight into what has been learned. This paper presents a technique that can help the user understand why a classifier makes the predictions that it does by providing a two-dimensional visualization of its class probability estimates. It requires the classifier to generate class probabilities but most practical algorithms are able to do so (or can be modified to this end).

1 Introduction

Visualization techniques are frequently used to analyze the input to a machine learning algorithm. This paper presents a generic method for visualizing the output of classification models that produce class probability estimates. The method has previously been investigated in conjunction with Bayesian network classifiers [5]. Here we provide details on how it can be applied to other types of classification models.

There are two potential applications for this technique. First, it can help the user understand what kind of information an algorithm extracts from the input data. Methods that learn decision trees and sets of rules are popular because they represent the extracted information in intelligible form. This is not the case for many of the more powerful classification algorithms. Second, it can help machine learning researchers understand the behavior of an algorithm by analyzing the output that it generates. Standard methods of assessing model quality—for example, receiver operating characteristic (ROC) curves [4]—provide information about a model's predictive performance, but fail to provide any insight into why a classifier makes a particular prediction.

Most existing methods for visualizing classification models are restricted to particular concept classes. Decision trees can be easily visualized, as can decision tables and naive Bayes classifiers [8]. In this paper we discuss a general visualization technique that can be applied in conjunction with any learning algorithm for classification models as long as these models estimate class probabilities. Most learning algorithms fall into this category.[1]

The underlying idea is very simple. Ideally we would like to know the class probability estimate for each class for every point in instance space. This would give us a complete picture of the information contained in a classification model. When there are two attributes it is possible to plot this information with arbitrary precision for each class in a two-dimensional plot, where the color of the point encodes the class probability (e.g. black corresponds to probability zero and white corresponds to probability one). This can easily be extended to three dimensions. However, it is not possible to use this simple visualization technique if there are more than three attributes.

This paper presents a data-driven approach for visualizing a classifier regardless of the dimension of the instance space. This is accomplished by projecting its class probability estimates into two dimensions. It is inevitable that some information will be lost in this transformation but we believe that the resulting plotting technique is a useful tool for data analysts as well as machine learning researchers. The method is soundly based in probability theory and aside from the classifier itself only requires an estimate of the attributes' joint density (e.g. provided by a kernel density estimator).

The structure of the paper is as follows. Section 2 describes the visualization technique in more detail. Section 3 contains some experimental results. Section 4 discusses related work, and Section 5 summarizes the paper.

2 Visualizing expected class probabilities

The basic idea is to visualize the information contained in a classification model by plotting its class probability estimates as a function of two of the attributes in the data. The two attributes are user specified and make up the x and y axes of the visualization. In this paper we only consider domains where all the attributes are numeric. We discretize the two attributes so that the instance space is split into disjoint rectangular regions and each region corresponds to one pixel on the screen. The resulting rectangles are open-sided along all other attributes. Then we estimate the expected class probabilities in each region by sampling points from the region, obtaining class probability estimates for each point from the classification model, and averaging the results. The details of this method are explained below.

The probability estimates for each region are color coded. We first assign a color to each class. Each of these colors corresponds to a particular combination of RGB values. Let (r_k, g_k, b_k) be the RGB values for class k, i.e. if class k gets probability one in a given region, this region is colored using those RGB values. If no class receives probability one, the resulting color is computed as a linear combination of all the classes' RGB values. Let $\hat{e}_k$ be the estimated expected

[1] Note that the technique can also be used in conjunction with clustering algorithms that produce cluster membership probabilities.

probability for class k in a particular region. Then the resulting RGB values are computed as follows:

$$r = \sum_k \hat{e}_k \times r_k \qquad g = \sum_k \hat{e}_k \times g_k \qquad b = \sum_k \hat{e}_k \times b_k \tag{1}$$

This method smoothly interpolates between pure regions—regions where one class obtains all the probability mass.

The class colors can be chosen by the user based on a standard color chooser dialog. For example, when there are only two classes, the user might choose black for one class, and white for the other, resulting in a grey-scale image. Note that the colors will not necessarily uniquely identify a probability vector. In a four-class problem setting the corresponding colors to $(1,0,0), (0,1,0), (0,0,1)$, and $(0,0,0)$ will result in a one-to-one mapping. However, with other color settings and/or more classes there may be clashes. To alleviate this problem our implementation shows the user the probability vector corresponding to a certain pixel on mouse over. It also allows the user to change the colors at any time, so that the situation in ambiguous regions can be clarified.

We now discuss how we estimate the expected class probabilities for each pixel (i.e. each rectangular region in instance space). If the region is small enough we can assume that the density is approximately uniform within the region. In this case we can simply sample points uniformly from the region, obtain class probability estimates for each point from the model, and average the results. However, if the uniformity assumption does not hold we need an estimate $\hat{f}$ of the density function—for example, provided by a kernel density estimator [2]—and sample or weight points according to this estimate. Using the density is crucial when the method is applied to instance spaces with more than two dimensions (i.e. two predictor attributes) because then the uniformity assumption is usually severely violated.

Given a kernel density estimator $\hat{f}$ we can estimate the expected class probabilities by sampling instances from a region using a uniform distribution and weighting their predicted class probabilities $\hat{p}_k$ according to $\hat{f}$. Let $S = (\mathbf{x}_1, ..., \mathbf{x}_l)$ be our set of l uniformly distributed samples from a region. Then we can estimate the expected class probability $\hat{e}_k$ of class k for that region as follows:

$$\hat{e}_k = \frac{\sum_{\mathbf{x} \in S} \hat{f}(\mathbf{x}) \hat{p}_k(\mathbf{x})}{\sum_{\mathbf{x} \in S} \hat{f}(\mathbf{x})}. \tag{2}$$

If there are only two dimensions this method is quite efficient. The number of samples required for an accurate estimate could be determined automatically by computing a confidence interval for it, but the particular choice of l is not critical if the screen resolution is high enough (considering the limited resolution of the color space and the sensitivity of the human eye to local changes in color).

Unfortunately this estimation procedure becomes very inefficient in higher-dimensional instance spaces because most of the instances in S will receive a very low value from the density function: most of the density will be concentrated in specific regions of the space. Obtaining an accurate estimate would require a very large number of samples. However, it turns out that there is a more efficient sampling strategy for estimating $\hat{e}_k$. This strategy is based on the kernel density estimator that we use to represent $\hat{f}$.

A kernel density estimator combines local estimates of the density based on each instance in a dataset. Assuming that there are n instances $\mathbf{x}_i$ it consists of n kernel functions:

$$\hat{f}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} k_i(\mathbf{x}) \tag{3}$$

where k_i is the kernel function based on instance $\mathbf{x}_i$:

$$k_i(\mathbf{x}) = \prod_{j=1}^{m} k_{ij}(x_j). \tag{4}$$

This is a product of m component functions, one for each dimension. We use a Gaussian kernel, for which the component functions are defined as:

$$k_{ij}(x_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp\left(-\frac{(x_j - x_{ij})^2}{2\sigma_{ij}^2}\right). \tag{5}$$

Each k_{ij} is the density of a normal distribution centered on attribute value j of instance $\mathbf{x}_i$. The parameter σ_{ij} determines the width of the kernel along dimension j. In our implementation we use $\sigma_{ij} = (max_j - min_j) \times d_i$, where max_j and min_j are the maximum and minimum value of attribute j, and d_i is the Euclidean distance to the k-th neighbor of $\mathbf{x}_i$ after all the attributes' values have been normalized to lie between zero and one. The value of the parameter k is user specified. Alternatively it could be determined by maximizing the cross-validated likelihood of the data [7].

Based on the kernel density estimator we can devise a sampling strategy that produces a set of instances Q by sampling a fixed number of instances from each kernel function. This can be done by sampling from the kernel's normal distributions to obtain the attribute values for each instance. The result is that the instances in Q are likely to be in the populated areas of the instance space. Given Q we can estimate the expected class probability for a region R as follows:

$$\hat{e}_k = \frac{1}{|\mathbf{x} \in R \wedge \mathbf{x} \in Q|} \sum_{\mathbf{x} \in R \wedge \mathbf{x} \in Q} \hat{p}_k(\mathbf{x}). \tag{6}$$

Unfortunately this is not the ideal solution: for our visualization we want accurate estimates for every pixel, not only the ones corresponding to populated parts of the instance space. Most regions R will not receive any samples. The solution is to split the set of attributes into two subsets: the first set containing the two attributes our visualization is based on, and the second set containing the remainder. Then we can fix the values of the attributes in the first set so that we are guaranteed to get an instance in the area that we are interested in (corresponding to the current pixel), sample values for the other attributes from a kernel, and use the fixed attributes to weight the resulting instance according to the density function.

Let us make this more precise by assuming that our visualization is based on the first two attributes in the dataset. For these two attributes we fix values x_1 and x_2 in the region corresponding to the pixel that we want to plot. Then we obtain an instance $\mathbf{x}_i$ from kernel k_i by sampling from the kernel's normal distributions to obtain attribute values $x_{i3}, ..., x_{im}$ and setting $x_{i1} = x_1$ and $x_{i2} = x_2$. We can then estimate the class probability $p_k(x_1, x_2)$ for location x_1, x_2 as follows:

$$\hat{p}_k(x_1, x_2) = \frac{\sum_{i=1}^{n} \hat{p}_k(\mathbf{x}_i) k_{i1}(x_1) k_{i2}(x_2)}{\sum_{i=1}^{n} k_{i1}(x_1) k_{i2}(x_2)}. \tag{7}$$

This is essentially the likelihood weighting method used to perform probabilistic inference in Bayesian networks [6]. The $p_k(\mathbf{x}_i)$ are weighted by the $k_{i1}(x_1)k_{i2}(x_2)$ to take the effect of the kernel on the two fixed dimensions into account. The result of this process is that we have marginalized out all dimensions apart from the two that we are interested in.

One sample per kernel is usually not sufficient to obtain an accurate representation of the density and thus an accurate estimate of $p_k(x_1, x_2)$, especially in higher-dimensional spaces. In our implementation we repeat the sampling process r^{m-2} times, where r is a user-specified parameter, evaluate Equation 7 for each resulting set of instances, and take the overall average as an estimate of $p_k(x_1, x_2)$. A more sophisticated approach would be to compute a confidence interval for the estimated probability and to stop the sampling process when a certain precision has been attained.

Note that the running time can be decreased by first sorting the $\mathbf{x}_i$ according to their weights $k_{i1}(x_{i1})k_{i2}(x_{i2})$ and then sampling from the corresponding kernels in decreasing order until the cumulative weight exceeds a certain percentage of the total weight (e.g. 99%). Usually only a small fraction of the kernels need to be sampled from as a result of this filtering process.

To obtain the expected class probability $\hat{e}_k$ for a region corresponding to a particular pixel we need to repeat this estimation process for different locations x_{l1}, x_{l2} within the pixel and compute a weighted average of the resulting probability estimates based on the density function:

$$\hat{e}_k = \frac{\sum_l \hat{f}(x_{l1}, x_{l2}) \hat{p}_k(x_{l1}, x_{l2})}{\sum_l \hat{f}(x_{l1}, x_{l2})}, \tag{8}$$

where

$$\hat{f}(x_{l1}, x_{l2}) = \frac{1}{n} \sum_{i=1}^{n} k_{i1}(x_{l1}) k_{i2}(x_{l2}). \tag{9}$$

This weighted average is then plugged into Equation 1 to compute the RGB values for the pixel.

3 Some example visualizations

In the following we visualize class probability estimators on three example domains. We restrict ourselves to two-class domains so that all probability vectors can be represented by shades of grey. Some color visualizations are available online at `http://www.cs.waikato.ac.nz/ml/weka/bvis`. Note that our implementation is included in the Java package `weka.gui.boundaryvisualizer` as part of the Weka machine learning workbench [9] Version 3.3.6 and later.[2]

For each result we used two locations per pixel to compute the expected probability, set the parameter r to two (i.e. generating four samples per kernel for a problem with four attributes), and used the third neighbor ($k = 3$) for computing the kernel width.

The first domain is an artificial domain with four numeric attributes. Two of the attributes are relevant (x_1 and x_2) and the remaining ones (x_3 and x_4) are irrelevant. The attribute values were generated by sampling from normal distributions with unit variance. For the irrelevant attributes the distributions were centered at zero for both classes. For the relevant ones they were centered at -1 for class one and $+1$ for class two. We generated a dataset containing 100 instances from each class.

We first built a logistic regression model from this data. The resulting model is shown in Figure 1. Note that the two irrelevant attributes have fairly small coefficients, as expected. Figure 2 shows the results of the visualization procedure for three different pairs of attributes based on this model. The points superimposed on the plot correspond to the actual attribute values of the training instances in the two dimensions visualized. The color (black or white) of each point indicates the class value of the corresponding instance.

Figure 2a is based on the two relevant attributes (x_1 on the x axis and x_2 on the y axis). The linear class boundary is clearly defined because the two visualization attributes are the only relevant attributes in the dataset. The lower triangle represents class one and the upper triangle class two. Figure 2b shows the result for x_1 on the x axis, and x_3 on the y axis. It demonstrates visually that x_1 is relevant while x_3 is not. Figure 2c displays a visualization based on the two irrelevant attributes. It shows no apparent structure—as expected for two completely irrelevant attributes.

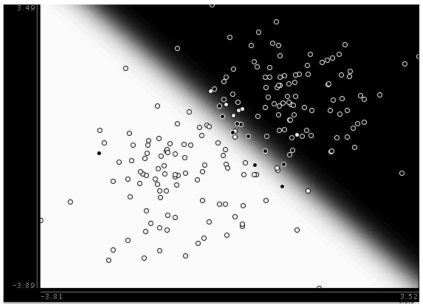
Figure 4 shows visualizations based on the same pairs of attributes for the decision tree from Figure 3. The tree is based exclusively on the two relevant attributes, and this fact is reflected in Figure 4a: the area is divided into rectangular regions that are uniformly colored (because the probability vectors are constant within each region). Note that the black region corresponds to three separate leafs and that one of them is not pure. The difference in "blackness" is not discernible.

Figure 4b shows the situation for attributes x_1 (relevant) and x_3 (irrelevant). Attribute x_1 is used twice in the tree, resulting in three distinct bands. Note that

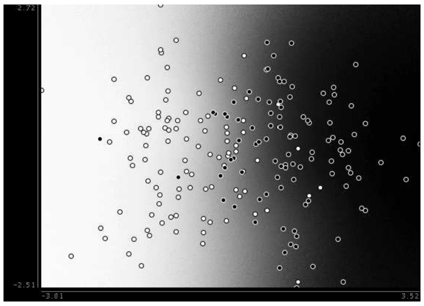
[2] Available from http://www.cs.waikato.ac.nz/ml/weka.

$$p(class = one|\mathbf{x}) = \frac{1}{1+e^{4.77x_1+4.21x_2-0.15x_3-0.14x_4-1.05}}$$

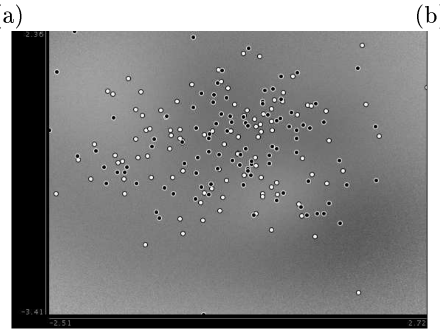
Fig. 1. The logistic regression model for the artificial dataset.



(a)



(b)



(c)

Fig. 2. Visualizing logistic regression for the artificial data using (a) the two relevant attributes, (b) one relevant and one irrelevant attribute, and (c) the two irrelevant attributes.

the three bands are (almost) uniformly colored, indicating that the attribute on the y axis (x_3) is irrelevant.

Figure 4c is based on the two irrelevant attributes. The visualization shows no structure and is nearly identical to the corresponding result for the logistic regression model shown in Figure 2c. Minor differences in shading compared to Figure 2c are due to differences in the class probability estimates that are caused by the two relevant attributes (i.e a result of the differences between Figures 4a and 2a).

For illustrative purposes Figure 6 shows a visualization for a two-class version of the `iris` data (using the 100 instances pertaining to classes `iris-virginica` and `iris-versicolor`) based on the decision tree in Figure 5. The `iris` data can be obtained from the UCI repository [1]. In Figure 6a the `petallength` attribute is shown on the x axis and the `petalwidth` attribute on the y axis. There are four uniformly colored regions corresponding to the four leaves of the tree. In Figure 6b `petallength` is on the x axis and `sepallength` on the y axis. The influence of `sepallength` is clearly visible in the white area despite

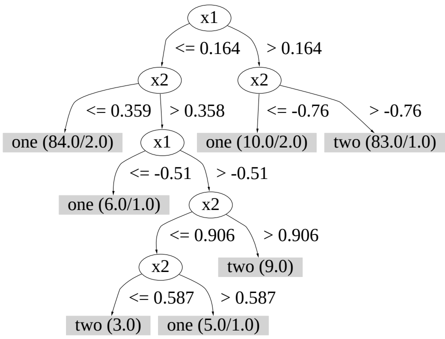
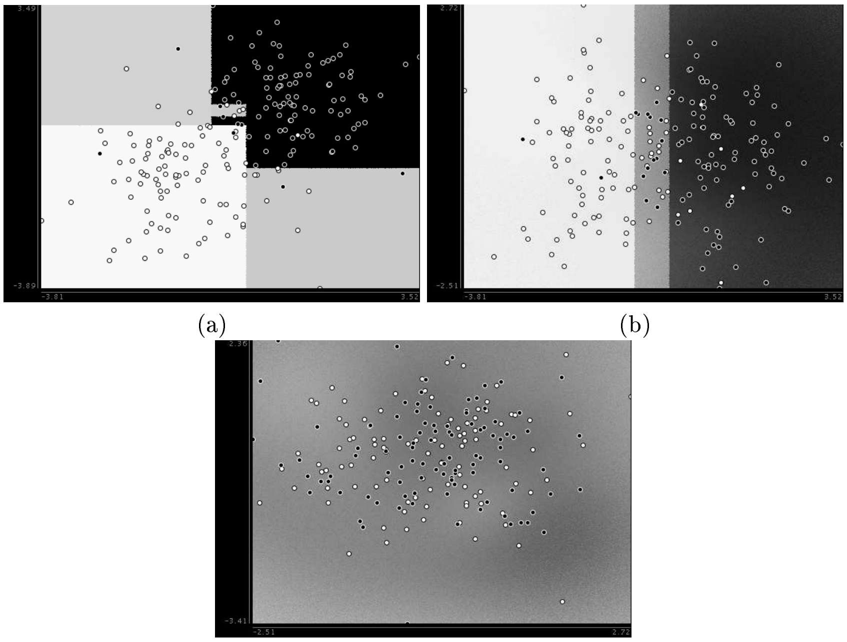


Fig. 3. The decision tree for the artificial dataset.



(a) (b)

(c)

Fig. 4. Visualizing the decision tree for the artificial data using (a) the two relevant attributes, (b) one relevant and one irrelevant attribute, and (c) the two irrelevant attributes.

this attribute not being used in the tree. Figure 6c is based on `sepallength` (x) and `sepalwidth` (y). Although these attributes are not used in the tree the visualization shows a clear trend going from the lower left to the upper right, and a good correlation of the probability estimates with the actual class values of the training instances. This is a result of correlations that exist between `sepallength` and `sepalwidth` and the two attributes used in the tree.

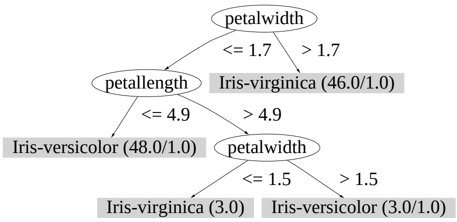
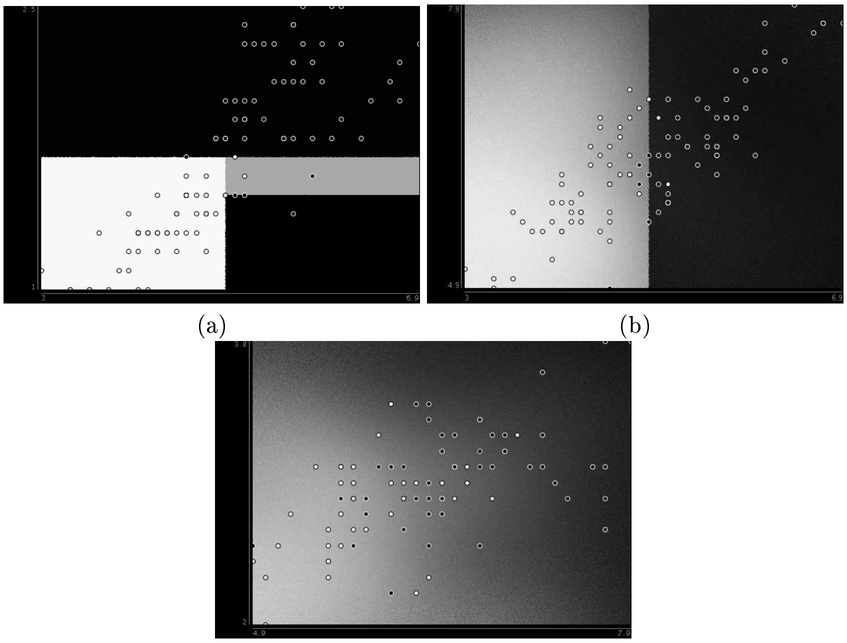


Fig. 5. The decision tree for the two-class iris dataset.



(a) (b)

(c)

Fig. 6. Visualizing the decision tree for the two-class iris data using (a) `petallength` and `petalwidth`, (b) `petallength` and `sepallength`, and (c) `sepallength` and `sepalwidth` (with the first attribute on the x axis and the second one on the y axis).

This particular example shows that the pixel-based visualization technique can provide additional information about the structure in the data even when used in conjunction with an interpretable model like a decision tree. In this case it shows that the decision tree implicitly contains much of the information provided by the `sepallength` and `sepalwidth` attributes (although they are not explicitly represented in the classifier).

To provide a more realistic example Figure 8 shows four visualizations for pairs of attributes from the `pima-indians diabetes` dataset [1]. This dataset has eight attributes and 768 instances (500 belonging to class `tested_negative`

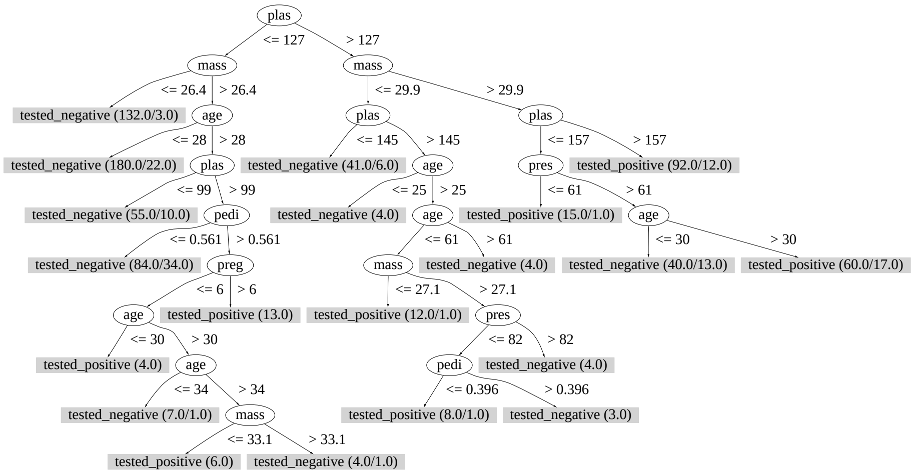


Fig. 7. The decision tree for the diabetes dataset.

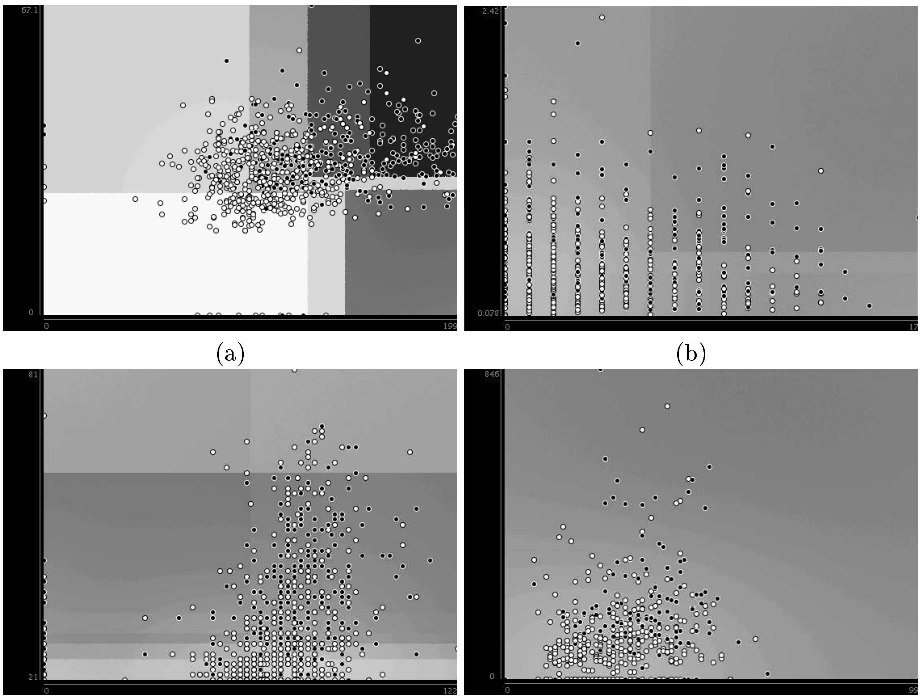


Fig. 8. Visualizing the decision tree for the diabetes data using (a) `plas` and `mass`, (b) `preg` and `pedi`, (c) `pres` and `age`, and (d) `skin` and `insu` (with the first attribute on the x axis and the second one on the y axis).

and 268 to class `tested_positive`). The decision tree for this problem is shown in Figure 7.

Figure 8a is based on the `plas` (x axis) and `mass` (y axis) attributes, which are tested at the root and the first level of the decision tree respectively (as well as elsewhere in the tree). This makes them likely to be the most predictive attributes in the data. There are eight nodes in the tree where either one of these attributes is tested. This results in nine distinct rectangular regions, of which eight are visible in the visualization. The missing region is a result of the last split on the `mass` attribute at the bottom of the left subtree and hidden by the points in the middle of the plot. There are two regions where the classifier is certain about the class membership: a white region in the lower left corner, and a black region in the upper right one. These correspond to the left-most and right-most paths in the tree respectively, where only the two visualization attributes are tested. All other regions involve tests on other attributes and are therefore associated with greater uncertainty in the class membership.

Figure 8b is based on `preg` (x) and `pedi` (y). They are tested at three nodes in the tree—all below level four—resulting in four rectangular regions. Only three of these are discernible in the plot; the fourth one corresponds to the split on `pedi` at the bottom of the right subtree (and is very faintly visible on screen but not in the printed version).

In Figure 8c the visualization is based on `pres` (x) and `age` (y), tested eight times in total. Note that some of the rectangular regions are the consequence of overlapping regions originating from splits in different subtrees because the subtrees arise by partitioning on non-visualized attributes.

Figure 8d visualizes the model using the only two attributes that do not occur in the decision tree: `skin` (x) and `insu` (y). Again, like in the iris data (Figure 6b), there is some correlation between the actual class labels and the attributes not explicitly taken into account by the classifier. However, in this case the correlation is very weak.

4 Related Work

There appears to have been relatively little attention devoted to general techniques for the visualization of machine learning models. Methods for particular types of classifiers have been developed, for example, for decision trees, decision tables, and naive Bayesian classifiers [8], but in these cases the visualization procedure follows naturally from the model structure.

The structure of Bayesian networks can be visualized as a directed graph. However, the graph-based visualization is limited because it does not provide any visualization of the probability estimates generated by a network. Rheingans and desJardins [5] apply the basic pixel-based visualization technique discussed in this paper to visualize these estimates. They indicate that the technique can be used in conjunction with other types of class probability estimators but do not provide details on how this can be done. Inference methods for Bayesian networks directly provide estimates of conditional probabilities based on evidence variables (in this case the two attributes used in the visualization), and this means a separate density estimator and sampling strategy is not required. Rheingans and desJardins also investigate a visualization technique that maps the instance space into two dimensions using a self-organizing map (SOM) [3]. However, this makes it difficult to relate a pixel in the visualization to a point in the original instance space.

5 Conclusions

This paper has presented a generic visualization technique for class probability estimators. The basic method is not new and has been investigated in the context of Bayesian network classifiers before. Our contribution is that we have provided details on how to generalize it to arbitrary classification models that produce class probability estimates. We have provided some example visualizations based on logistic regression and decision trees that demonstrate the usefulness of this method as a general tool for analyzing the output of learning algorithms. Potential applications are two fold: practitioners can use this tool to gain insight into the data even if a learning scheme does not provide an interpretable model, and machine learning researchers can use it to explore the behavior of a particular learning technique.

Acknowledgments

Many thanks to Len Trigg for pointing out that the method is applicable to probabilistic clustering algorithms, and to Geoff Holmes and Bernhard Pfahringer for their comments. This research was supported by Marsden Grant 01-UOW-019.

References

1. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. [www.ics.uci.edu/~mlearn/MLRepository.html].
2. Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer-Verlag, 2001.
3. T. Kohonen. *Self-Organizing Maps.* Springer-Verlag, 1997.
4. Foster J. Provost and Tom Fawcett. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. *Knowledge Discovery and Data Mining*, pages 43–48, 1997.
5. Penny Rheingans and Marie desJardins. Visualizing high-dimensional predictive model quality. In *Proceedings of IEEE Visualization 2000*, pages 493–496, 2000.
6. Stuart Russell and Peter Norvig. *Artificial Intelligence.* Prentice-Hall, 1995.
7. P. Smyth. Model selection for probabilistic clustering using cross-validated likelihood. *Statistics and Computing*, pages 63–72, 2000.
8. Kurt Thearling, Barry Becker, Dennis DeCoste, Bill Mawby, Michel Pilote, and Dan Sommerfield. *Information Visualization in Data Mining and Knowledge Discovery*, chapter Visualizing Data Mining Models. Morgan Kaufmann, 2001.
9. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, 2000.