Prediction Cubes

Bee-Chung Chen, Lei Chen, Yi Lin, Raghu Ramakrishnan

University of Wisconsin, Madison, WI 53706, USA
{beechung, chenl, raghu}@cs.wisc.edu, yilin@stat.wisc.edu

Abstract

In this paper, we introduce a new family of tools for exploratory data analysis, called *prediction cubes*. As in standard OLAP data cubes, each cell in a prediction cube contains a value that summarizes the data belonging to that cell, and the granularity of cells can be changed via operations such as roll-up and drill-down. In contrast to data cubes, in which each cell value is computed by an aggregate function, e.g., SUM or AVG, each cell value in a prediction cube summarizes a predictive model trained on the data corresponding to that cell, and characterizes its decision behavior or predictiveness. In this paper, we propose and motivate prediction cubes, and show that they can be efficiently computed by exploiting the idea of *model decomposition*.

1. Introduction

It is widely recognized that exploratory data analysis is an iterative process, and that the bulk of the time is spent on understanding the structure and patterns suggested by applying one or more data mining algorithms on different subsets or differently conditioned versions of the data. Yet, almost all research has concentrated on either improving the quality or efficiency of mining algorithms, and has ignored the bottleneck of the human in the loop.

In this paper, we directly address the question of how we can assist the analyst in identifying subsets of data that are "interesting" in light of a given predictive model; the underlying idea can be generalized to support other kinds of exploratory analysis settings as well.

Our basic proposal is simple yet powerful—OLAP is now a well-understood, powerful tool for systematically exploring trends in aggregation queries across subsets of data. We adapt it naturally to support exploration of trends in the decision behavior of a given predictive model across the same family of data subsets, allowing for ready integration into existing OLAP interfaces. Repeated similar analysis of different subsets of data is avoided by a single high-level cube construction query, followed by OLAP-style exploration of hierarchically organized results.

1.1 Contributions and Future Directions

In this paper, we: (1) introduce *prediction cubes,* (2) develop a general computational technique, called *scoring function decomposition*, to improve the efficiency of prediction cube materialization, (3) show how to apply the proposed technique to build prediction cubes for several commonly used machine learning algorithms, and finally (4) present a series of experiments that empirically evaluate the accuracy and efficiency of cube construction.

This paper is a first step, and opens a number of interesting directions for future research. Beyond possible improvements to the algorithms we propose, constructing prediction cubes for other predictive models is an important challenge. If we view parameters of learning algorithms as dimensions of the cube, this opens the door to a significantly more general use of prediction cubes; beyond exploring data subsets, the paradigm can be used to explore alternatives in conditioning the data (e.g., choices for scaling different data axes) or tuning the learning algorithm (e.g., choices for various "magic thresholds"). Efficient cube computation for these generalizations is wide open.

1.2 Motivating Example

Consider a nationwide bank whose managers want to analyze the bank's loan approval process with respect to two dimensions, *Location* and *Time* (illustrated in Figure 1, 3 pages after). They are interested in questions like:

1. Given a set of sensitive attributes (e.g., race and sex), are there locations and times during which approvals depended highly on those attributes?
2. Are there locations and times when the decision making was similar to that in 1950s Alabama?

When predictive models built by training a machine learning algorithm are used to assist in such approval decisions, the questions essentially have to do with predictiveness of certain attributes and similarity of models trained on different subsets of data. Social scientists have raised concerns that the use of data mining introduces the risk of discrimination that is hard to identify, because it is latent in sophisticated models [6].

Proceedings of the 31st VLDB Conference,
Trondheim, Norway, 2005

These questions are also complicated by the fact that the candidate answers are *subsets* of data, partitioned by location and time values; clearly, there are a large number of candidates. Although the *Location* and *Time* hierarchies are known, the right level (granularity) for the analysis is unclear; e.g., is performing analysis using State-Month better than using City-Year? Thus, it is desirable to have a tool that allows the bank's analysts to navigate through different levels by rolling-up and/or drilling-down along the hierarchies. We propose a new kind of data mining tool, called *prediction cubes*, to support such analysis.

Figure 3 (2 pages after) shows an example of a 2-dimensional prediction cube for answering the first question. In Figure 3 (c), each cell is indexed by a [*State, Year*] pair. Each cell value is the *predictiveness* of the sensitive attributes, calculated by evaluating two models trained on the subset of data in that cell. (In Section 2, we discuss how to measure predictiveness.) We call this kind of prediction cube a *predictiveness cube*. Prediction cubes support navigation via roll-up (e.g., from [State, Year] to [State, All]) and drill-down (e.g., from [State, Year] to [State, Month]). By navigating through this cube, we can quickly check whether the models trained on different subsets of data reflect, e.g., racial discrimination.

A naïve approach, called the *exhaustive method*, to roll-up or drill-down in a prediction cube is to exhaustively build a model from scratch and evaluate it for each cell. Given that building a model is generally costly, this is likely to be prohibitively expensive. Instead of exhaustively building a model from scratch each time, we propose to generate the model of a higher-level (coarse-grained) cell by combining the models of those lower-level (finer-grained) cells that fall in the higher-level cell, e.g., to build the model for [WI, 1986] by combining the models of [WI, Jan 86], …, [WI, Dec 86].

2. Predictive Models

Predictive models are the central objects in prediction cubes. We first present the basic concepts and notation, and then describe standard machine learning techniques for measuring the mode accuracy, the similarity between models, and attribute predictiveness.

2.1 Basics

Let **D** denote a table of data of schema [$\boldsymbol{X}$, Y], where $\boldsymbol{X} = \{X_1, \ldots, X_m\}$ is a set of predictor attributes and Y is the class label (i.e., the dependent attribute). Each row in **D** is called an example. A predictive model $h(\boldsymbol{X}; \mathbf{D})$ is a model trained on **D** using learning algorithm h that can predict the class label y of a new example $\boldsymbol{x}$. For ease of expression, if the training dataset is not important or can be inferred from the context, we just use $h(\boldsymbol{X})$ to denote a predictive model. Also, we use $h(\boldsymbol{x}; \mathbf{D})$ to denote the function that outputs the prediction of $h(\boldsymbol{X}; \mathbf{D})$ on input $\boldsymbol{x}$. For example, **D** is a table of loan application data, with schema [*Age*, *Gender*, *Race*, *Approval*], where $\boldsymbol{X}$={*Age*, *Gender*, *Race*} denotes the predictor attributes and *Y*=*Approval* is the class label. The predictive model *decision_tree*($\boldsymbol{X}$; **D**) is the decision tree trained on **D**, to predict whether a person's loan application would be approved based on his/her age, gender and race.

In Machine Learning and Statistics, **D** is usually assumed to be a random sample drawn independently from an underlying probability distribution $p^*(\boldsymbol{X}, Y)$. Since different datasets come from different distributions, we use $p^*(\boldsymbol{X}, Y \mid \mathbf{D})$ to denote the distribution for dataset **D**. Given this distribution, the "best" class label for input $\boldsymbol{x}$ is the class label that maximizes the conditional class-probability $p^*(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, \mathbf{D})$, for all class labels y; i.e.,

$$\text{best\_class}(\boldsymbol{x} \mid \mathbf{D}) = \text{argmax}_{\,y}\, p^*(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, \mathbf{D}).$$

From this probabilistic point of view, a predictive model $h(\boldsymbol{X}; \mathbf{D})$ is *optimal* if for any input $\boldsymbol{x}$, $h(\boldsymbol{x}; \mathbf{D})$ always outputs the "best" class label of $\boldsymbol{x}$; i.e.,

$$h(\boldsymbol{x}; \mathbf{D}) = \text{argmax}_{\,y}\, p^*(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, \mathbf{D}).$$

Thus, $h(\boldsymbol{X}; \mathbf{D})$ can be thought of as an approximation of $p^*(Y \mid \boldsymbol{X}, \mathbf{D})$. Further, it is intuitive to imagine that, during training, $h(\boldsymbol{X}; \mathbf{D})$ constructs an internal probability distribution $p_h(Y \mid \boldsymbol{X}, \mathbf{D})$ that approximates $p^*(Y \mid \boldsymbol{X}, \mathbf{D})$. Thus, the prediction that $h(\boldsymbol{X}; \mathbf{D})$ makes on $\boldsymbol{x}$ is the class label that maximizes $p_h(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, \mathbf{D})$, for all y; i.e.,

$$h(\boldsymbol{x}; \mathbf{D}) = \text{argmax}_{\,y}\, p_h(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, \mathbf{D}).$$

In fact, many machine learning algorithms either do have such internal probability distributions, or have some scoring components that have a similar probabilistic meaning, though the scores are not actually probabilities.

2.2 Model Accuracy

How well a predictive model performs is generally measured by its accuracy. Theoretically, the accuracy of $h(\boldsymbol{X}; \mathbf{D})$ is defined by how often we expect it to be correct:

$$E_{x,y}[\, I(\, h(\boldsymbol{x}; \mathbf{D}) = y\,)\,],$$

where ($\boldsymbol{x}$, y) is drawn from $p^*(\boldsymbol{X}, Y \mid \mathbf{D})$, and I is the indicator function. If statement Ψ is true, $I(\Psi) = 1$, else $I(\Psi) = 0$. Since $p^*(\boldsymbol{X}, Y \mid \mathbf{D})$ is an unknown distribution, in practice, the accuracy of $h(\boldsymbol{X}; \mathbf{D})$ is measured empirically by using additional test set $\boldsymbol{\Delta}$ that is not used to train $h(\boldsymbol{X}; \mathbf{D})$, and is assumed to be drawn from $p^*(\boldsymbol{X}, Y \mid \mathbf{D})$.

Definition 1: Test-set accuracy. *Given a set-aside test set* $\boldsymbol{\Delta}$ *of schema* [$\boldsymbol{X}$, Y], *the test-set accuracy of* $h(\boldsymbol{X}; \mathbf{D})$ *is*

$$\frac{1}{|\boldsymbol{\Delta}|}\sum_{(x,y)\in\boldsymbol{\Delta}} I(h(\boldsymbol{x};\mathbf{D}) = y),$$

where $|\boldsymbol{\Delta}|$ *is the size of* $\boldsymbol{\Delta}$. *We use accuracy*($h(\boldsymbol{x}; \mathbf{D}) \mid \boldsymbol{\Delta}$) *to denote the test-set accuracy of* $h(\boldsymbol{X}; \mathbf{D})$ *based on test set* $\boldsymbol{\Delta}$.

Note that if we do not have a set-aside test set, a commonly used method is n-fold cross validation. We first randomly divide **D** into n non-overlapping partitions $\mathbf{D}_1, \ldots, \mathbf{D}_n$. Then, for i = 1 to n, we use $\cup_{j\neq i} \mathbf{D}_j$ as the training data to build a model, and then use $\mathbf{D}_i$ as the test data to measure the model's test-set accuracy. Then, the cross-validation accuracy is the average of the above n accuracies. A common choice of n is 10.

2.3 Model Similarity

The notion of similarity (or difference) between models is important in prediction behavior analysis. Let $h_1(\boldsymbol{X})$ and $h_2(\boldsymbol{X})$ be two predictive models. One simple method of measuring the similarity between $h_1(\boldsymbol{X})$ and $h_2(\boldsymbol{X})$ is to test whether these two models predict same class labels for most examples in a test-set.

Definition 2: Prediction similarity and distance. *The (test-set-based) prediction similarity between two models, $h_1(\boldsymbol{X})$ and $h_2(\boldsymbol{X})$, on test set $\boldsymbol{\Delta}$ is*

$$\frac{1}{|\boldsymbol{\Delta}|}\sum_{x\in\boldsymbol{\Delta}} I(h_1(\boldsymbol{x}) = h_2(\boldsymbol{x})).$$

We use similarity($h_1(\boldsymbol{X})$, $h_2(\boldsymbol{X})$) to denote model similarity between $h_1(\boldsymbol{X})$ and $h_2(\boldsymbol{X})$. The prediction distance between $h_1(\boldsymbol{X})$ and $h_2(\boldsymbol{X})$ is $1 -$ similarity($h_1(\boldsymbol{X})$, $h_2(\boldsymbol{X})$).

Note that the test set $\boldsymbol{\Delta}$ used here need not have class labels. It is used to provide the desired distribution of $\boldsymbol{X}$. Usually, $\boldsymbol{\Delta}$ is generated according to the true underlying distribution $p^*(\boldsymbol{X})$. However, we can also control the test data; i.e., by using different test sets, we can compare models based on different regions of the feature space. For example, by using a test set of information about rich people, we can focus the comparison on how similarly two models treat rich people. Since $\boldsymbol{\Delta}$ does not need to have class labels, generating it is much easier than generating the test set for accuracy measurement.

From the probabilistic point of view, if the models $h_1(\boldsymbol{X})$ and $h_2(\boldsymbol{X})$ can also estimate the conditional class-probabilities, i.e., $p_{h_1}(Y \mid \boldsymbol{X})$ and $p_{h_2}(Y \mid \boldsymbol{X})$, then we can measure the similarity between $h_1(\boldsymbol{X})$ and $h_2(\boldsymbol{X})$ more precisely by using the Kullback-Leibler (KL) divergence between $p_{h_1}(Y \mid \boldsymbol{X})$ and $p_{h_2}(Y \mid \boldsymbol{X})$.

Definition 3: KL-distance [14]**.** *The (test-set-based) KL-distance between models, $h_1(\boldsymbol{X})$ and $h_2(\boldsymbol{X})$, on test set $\boldsymbol{\Delta}$ is*

$$\frac{1}{|\boldsymbol{\Delta}|}\sum_{x\in\boldsymbol{\Delta}}\sum_{y} p_{h_1}(y\mid x)\log\frac{p_{h_1}(y\mid x)}{p_{h_2}(y\mid x)}.$$

We use KL_distance($h_1(\boldsymbol{X})$, $h_2(\boldsymbol{X})$) to denote the KL-distance between $h_1(\boldsymbol{X})$ and $h_2(\boldsymbol{X})$.

Note that *KL_distance*(h_1, h_2) $\neq$ *KL_distance*(h_2, h_1), in general. A commonly used trick to make the distance symmetric is to use the sum of the two as the distance.

2.4 Attribute Predictiveness

Predictive models can be used to measure whether a set of attributes $\boldsymbol{V} \subseteq \boldsymbol{X}$ is predictive with respect to Y on a dataset $\mathbf{D}$. The intuition is that $\boldsymbol{V}$ is not predictive if and only if $\boldsymbol{V}$ is independent of Y given the other attributes $\boldsymbol{X} - \boldsymbol{V}$; i.e., $p^*(Y \mid \boldsymbol{X} - \boldsymbol{V}, \mathbf{D}) = p^*(Y \mid \boldsymbol{X}, \mathbf{D})$. Thus, the similarity between these two probabilities is a good measure of the predictiveness of $\boldsymbol{V}$. Since p^* is unknown in practice, we use the difference (prediction- or KL-distance) between $h(\boldsymbol{X}; \mathbf{D})$ and $h(\boldsymbol{X} - \boldsymbol{V}; \mathbf{D})$ as the measure of predictiveness.

Note that there is another way to measure the predictiveness of $\boldsymbol{V}$, based on the intuition that $\boldsymbol{V}$ is predictive if and only if the model using $\boldsymbol{V}$ is more accurate than the model not using $\boldsymbol{V}$; i.e., $h(\boldsymbol{X}; \mathbf{D})$ is more accurate than $h(\boldsymbol{X} - \boldsymbol{V}; \mathbf{D})$. Cross validation can be used to estimate the accuracies of $h(\boldsymbol{X}; \mathbf{D})$ and $h(\boldsymbol{X} - \boldsymbol{V}; \mathbf{D})$. In the interest of space, we do not discuss this alternative further.

3. Prediction Cubes

In this section, we define prediction cubes formally. We first introduce the kinds of analysis for which prediction cubes are designed, and then formally define prediction cubes and consider their materialization.

3.1 Model-based Subset Analysis

We are interested in model-based data analysis. More specifically, given a data table $\mathbf{D}$ of schema $[\boldsymbol{X}, Y]$, we want to understand the relationship between $\boldsymbol{X}$ and Y (i.e., $p^*(Y \mid \boldsymbol{X}, \mathbf{D})$) by building a model (i.e., $h(\boldsymbol{X}; \mathbf{D})$) that captures this relationship. Subsets $\sigma(\mathbf{D})$ are defined by relational selections, and we use models $h(\boldsymbol{X}; \sigma(\mathbf{D}))$ to approximate true distributions $p^*(Y \mid \boldsymbol{X}, \sigma(\mathbf{D}))$. The model characteristics we are interested in are:

- *Test-set behavior*: Given a test set $\boldsymbol{\Delta}$ of schema $[\boldsymbol{X}, Y]$, we want to know whether the models built on different subsets of $\mathbf{D}$ behave like the underlying distribution that generates $\boldsymbol{\Delta}$. For example, $\boldsymbol{\Delta}$ can be a list of loan applications that have been unfairly treated. We want to understand which branch or region at what time would treat those applications similarly unfairly. This can be estimated by using test-set accuracy.
- *Model-based data similarity*: Given a dataset $\mathbf{D}_0$, which can itself be a subset of $\mathbf{D}$, we want to know how similar $\mathbf{D}_0$ is to different subsets of $\mathbf{D}$. This comparison can be done by measuring the model similarity or distance between the model built on $\mathbf{D}_0$ and the models built on different subsets of $\mathbf{D}$.
- *Attribute predictiveness*: Given a set $\boldsymbol{V} \subseteq \boldsymbol{X}$ of attributes, e.g., sensitive attributes like race and sex, we want to know whether $\boldsymbol{V}$ is predictive with respect to Y on different subsets of $\mathbf{D}$. This is the predictiveness notion defined in Section 2.4.

Note that, for the above discussion to be valid, we have made the following fundamental assumption:

> *Predictive models built on datasets $\sigma(\mathbf{D})$ and $\mathbf{D}$ are good approximations to the true distribution, i.e., $p^*(Y \mid \boldsymbol{X}, \sigma(\mathbf{D}))$ and $p^*(Y \mid \boldsymbol{X}, \mathbf{D})$.*

As long as the accuracies of the predictive models are reasonably high, this assumption is generally accepted in Machine Learning and Statistics. In practice, we can try several different learning algorithms, and get a good sense about the prediction or decision characteristics. However, the number of all possible subsets of $\mathbf{D}$ is too large and not every subset of $\mathbf{D}$ is of interest. Thus, we borrow the idea of multidimensional and hierarchical data grouping from OLAP, and constrain the subsets that we consider to the ones defined by valid multidimensional hierarchical groupings.

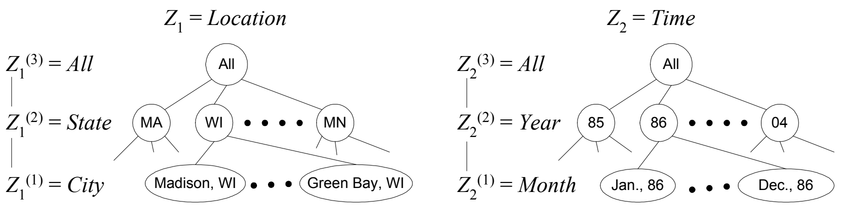


Figure 1. An example of dimensional hierarchies

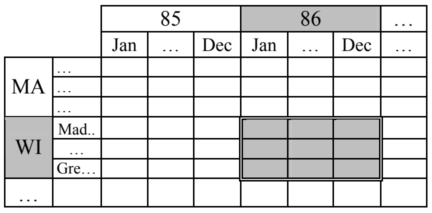


Figure 2. Visualization of $\sigma_{[WI,\, 86]}(\mathbf{D})$

		85			86			…
		Jan	…	Dec	Jan	…	Dec	…
MA	…	0.4	0.8	0.9	0.6	0.8	…	…
	…	0.2	0.3	0.5	0.3	…	…	…
	…	0.4	0.4	0.4	0.2	…	…	…
WI	Mad..	0.8	0.9	0.7	…	…	…	…
	…	0.3	0.8	0.8	…	…	…	
	Gre…	0.8	0.2	0.8	…	…	…	…
…	…	…	…	…	…	…	…	…

(a) The cube at level [1, 1]

		85			86			…
		Jan	…	Dec	Jan	…	Dec	…
MA	…	0.7			0.4			
	…	0.4			0.5			
	…	0.2			0.8			
WI	Mad..	0.8			0.9			
	…	0.4			…			
	Gre…	0.8			…			
…	…	…			…			

(b) The cube at level [1, 2]

		85			86			…
		Jan	…	Dec	Jan	…	Dec	…
MA	…	0.6			0.7			
	…							
	…							
WI	Mad..	0.7			0.9			
	…							
	Gre…							
…								

(c) The cube at level [2, 2]

Figure 3. An example of a cube at different levels

3.2 From Data Cubes to Prediction Cubes

OLAP is an environment that supports multidimensional and hierarchical data analysis. Data is stored in a fact table **D** with a set $\boldsymbol{Z} = \{Z_1, \ldots, Z_d\}$ of dimension attributes and a measure attribute Y, where each dimension Z_i has a hierarchical domain, e.g., Figure 1. A data cube is a d-dimensional array where the value in each cell is an aggregate value, e.g., sum or average, which summarizes the subset of data falling into that cell. Figure 3 (c) is an example. Formally, the value in the cell indexed by $[z_1, \ldots, z_d]$ is defined by a query of the following form.

SELECT agg(Y) FROM **D** WHERE Z_1=z_1 AND … AND Z_d=z_d;

where z_i's are values in domain hierarchies and agg(·) is an aggregate function, e.g., sum or average. For example, in Figure 3(c), the cell indexed by [WI, 86] is 0.9.

While data cubes are useful tools for understanding characteristics about data subsets, they provide little knowledge about prediction or decision characteristics. Thus, we extend the concept of a data cube as follows:

- Use the same OLAP mechanism to partition data into subsets and the same OLAP user interface to choose which subsets to see, e.g., roll-up and drill-down.
- Introduce new kinds of aggregate functions that capture prediction or decision behavior of data. Instead of simple aggregation, e.g. sum and average, the value in each cell is computed by evaluating a model built on the data subset defined by the cell.

We call these new kinds of cubes *prediction cubes*. The computational complexity of operating prediction cubes is much higher than that of data cubes. The main contributions of this paper are introducing the concept of prediction cubes, and developing efficient methods to compute them.

3.3 Dimensions and Hierarchies

We first redefine the schema of **D** to be $[\boldsymbol{Z}, \boldsymbol{X}, Y]$, where $\boldsymbol{Z}=\{Z_1, \ldots, Z_d\}$ is a set of dimension attributes, d is the number of dimensions, $\boldsymbol{X}$ is a set of predictor attributes and Y is the class label. In the motivating example, $\boldsymbol{Z} = \{Location, Time\}$. Along each dimension Z_i, there is a hierarchy. For simplicity of exposition, we assume the hierarchy of Z_i is "linear": $<Z_i^{(1)}, \ldots, Z_i^{(k)}>$, for some k, where $Z_i^{(t)}$ is a more "general" domain than $Z_i^{(t-1)}$. Thus, $Z_i^{(1)}$ is called the *least general domain*, and $Z_i^{(k)}$ is called the *most general domain*. We say that $Z_i^{(a)}$ is more "general" than $Z_i^{(b)}$ if each value in domain $Z_i^{(b)}$ is a child of exactly one value in domain $Z_i^{(a)}$ in the hierarchy. We call $Z_i^{(t)}$ the domain at *level t*. For example, as shown in Figure 1, the domain hierarchy of *Location* is <*City*, *State*, *All*>, where *City* is at level 1 and is the least general domain; *All* is at level 3 and is the most general domain. In this hierarchy, every city is a child of exactly one state, and every state is a child of "All" in the location hierarchy.

We use $v \in Z_i^{(t)}$ to denote that a value v is from domain $Z_i^{(t)}$. Without loss of generality, we assume that for any dimension Z_i, the domains $Z_i^{(1)}, \ldots, Z_i^{(k)}$ have different sets of values; i.e., there is no value v such that $v \in Z_i^{(a)}$ and $v \in Z_i^{(b)}$, for any i, a and b. E.g., there are different values in domain *Month* for the same month of different years.

Similar to the fact table in OLAP, we assume that the values in the dimension attributes of the data table **D** come from the least general domains, i.e., the $Z_i^{(1)}$s. A valid *multidimensional hierarchical subset* at level $[l_1, \ldots, l_d]$, denoted by $\sigma_{[v_1,\ldots,v_d]}(\mathbf{D})$ where $v_i \in Z_i^{(l_i)}$, is defined by the following query:

SELECT * FROM **D**
WHERE Z_1 IN $desc(v_1)$ AND … AND Z_d IN $desc(v_{|d|})$;

where $desc(v_i)$ denotes the set of values that are the descendants of v_i in the hierarchy of Z_i and v_i. For example, $\sigma_{[WI,\, 86]}(\mathbf{D})$ is the subset of data with location in *WI* and time in *86*. Note that the level of this subset is [2,2]. We can visualize a multidimensional hierarchical subset by plotting each example of **D** as a point in a d dimensional space based on their values on dimension attributes. Then, a multidimensional hierarchical subset

$\sigma_{[v_1,\ldots,v_d]}(\mathbf{D})$ is the set of examples (data points) falling in the box defined by $v_1, \ldots, v_d$. E.g., $\sigma_{[WI,\,86]}(\mathbf{D})$ is the set of examples falling in the shaded rectangle in Figure 2.

A *cube at level* $[l_1, \ldots, l_d]$ is a *d*-dimensional array, where each cell is indexed by $[v_1, \ldots, v_d]$, $v_i \in Z_i^{(l_i)}$, and the value in the cell is a number that summarizes the data subsets $\sigma_{[v_1,\ldots,v_d]}(\mathbf{D})$. We say that $\sigma_{[v_1,\ldots,v_d]}(\mathbf{D})$ is *the subset defined by cell* $[v_1, \ldots, v_d]$. Figure 3 shows an example of a cube at different levels. For example, in the cube of level [2, 2] each cell of the array is indexed by a state name and a year name. The value in cell [*WI*, *86*] is a number that summarize $\sigma_{[WI,\,86]}(\boldsymbol{D})$ (we will define the meaning of the values in prediction cubes later). *Roll-up* is the operator that changes a prediction cube from level $[l_1, \ldots, l_i, \ldots, l_d]$ to level $[l_1, \ldots, l_i^*, \ldots, l_d]$, where $l_i^* > l_i$, for some dimension *i*. *Drill-down* is the operator that changes a prediction cube from level $[l_1, \ldots, l_i, \ldots, l_d]$ to level $[l_1, \ldots, l_i^*, \ldots, l_d]$, where $l_i^* < l_i$, for some dimension *i*.

3.4 Prediction Cubes

We now formally define three types of test-set based (TS) prediction cubes, and then explain how to use them to perform model-based subset analysis. For all of the TS-prediction cubes, the user specifies: (1) a data table **D**, of schema $[\boldsymbol{Z}, \boldsymbol{X}, Y]$, together with the hierarchies associated with $\boldsymbol{Z}$, (2) a learning algorithm *h*, and (3) a test dataset Δ of schema $[\boldsymbol{X}, Y]$ (for TS-accuracy cubes; but $[\boldsymbol{X}]$ for the other two types of cubes).

Note that the test set Δ is a user-specified parameter. That means the user can choose the test set based on his/her desired data distribution.

Definition 4: TS-accuracy cube. *The TS-accuracy cube at level* $[l_1, \ldots, l_d]$ *is a d-dimensional array, in which the value in each cell is the test-set accuracy of* $h(\boldsymbol{X}; \sigma(\mathbf{D}))$ *based on test set* Δ, *where* $\sigma(\mathbf{D})$ *is the subset defined by that cell.*

Definition 5: Model-similarity (or distance) cube. *Given another user-specified model* $h_0(\boldsymbol{X})$, *the prediction-similarity cube (or KL-distance cube) at level* $[l_1, \ldots, l_d]$ *is a d-dimensional array, in which the value in each cell is the prediction similarity (or KL-distance) between* $h_0(\boldsymbol{X})$ *and* $h(\boldsymbol{X}; \sigma(\mathbf{D}))$ *based on unlabeled test set* Δ, *where* $\sigma(\mathbf{D})$ *is the subset defined by that cell.*

Definition 6: Predictiveness cube. *Given a set* $\boldsymbol{V} \subseteq \boldsymbol{X}$ *of attributes, the PD- (or KL-)* predictiveness *cube at level* $[l_1, \ldots, l_d]$ *is a d-dimensional array, in which the value in each cell is the prediction* (or *KL-*) *distance between* $h(\boldsymbol{X}–\boldsymbol{V}; \sigma(\mathbf{D}))$ *and* $h(\boldsymbol{X}; \sigma(\mathbf{D}))$ *measured by unlabeled test set* Δ, *where* $\sigma(\mathbf{D})$ *is the subset defined by that cell.*

Note that the operators on prediction cubes are the same as for data cube, e.g., roll-up and drill-down. In the following, we explain how to use prediction cubes to perform model-based subset analysis.

- *Test-set behavior*: We can use the TS-accuracy cube to analyze the test-set (Δ) behavior on different subsets.
- *Model-based data similarity*: Given a dataset $\mathbf{D}_0$, which can itself be a subset of **D**, we can first build a model h_0 on $\mathbf{D}_0$, and then measure the model-based similarity of $\mathbf{D}_0$ to different subsets of **D** using the model-similarity (or distance) cubes by providing h_0 as one of the input parameters.
- *Attribute predictiveness*: Given a set $\boldsymbol{V} \subseteq \boldsymbol{X}$ of attributes, we can check the predictiveness of $\boldsymbol{V}$ w.r.t. *Y* on different subsets using the predictiveness cubes.

Generalizing from the above cubes, if the user provides an evaluation function $Eval(h, \sigma(\mathbf{D}) \mid \Delta, \Theta)$ that evaluates the model behavior of $\sigma(\mathbf{D})$ using learning algorithm *h* based on test set Δ and some optional parameters Θ, then the general TS-prediction cube (general test-set-based prediction cube) can be defined as follows.

Definition 7: General TS-prediction cube. *Given an evaluation function Eval and an optional parameter set* Θ, *the general TS-prediction cube at level* $[l_1, \ldots, l_d]$ *is a d-dimensional array, where the value in each cell is* $Eval(h, \sigma(\mathbf{D}) \mid \Delta, \Theta)$, *and* $\sigma(\mathbf{D})$ *is the subset defined by that cell.*

Note that for TS-accuracy cubes, $Eval(h, \sigma(\mathbf{D}) \mid \Delta, \Theta)$ is the test-set accuracy of $h(\boldsymbol{X}; \sigma(\mathbf{D}))$ using Δ with Θ being empty. For model-similarity (or distance) cubes, Θ is h_0 and $Eval(h, \sigma(\mathbf{D}) \mid \Delta, \Theta)$ is the similarity (or distance) between $h(\boldsymbol{X}; \sigma(\mathbf{D}))$ and $h_0(\boldsymbol{X})$ based on Δ. For predictiveness cubes, $Eval(h, \sigma(\mathbf{D}) \mid \Delta, \Theta)$ is the similarity (or distance) between $h(\boldsymbol{X}; \sigma(\mathbf{D}))$ and $h(\boldsymbol{X}–\boldsymbol{V}; \sigma(\mathbf{D}))$ based on Δ with Θ being $\boldsymbol{V}$. Also note that we can define prediction cubes based on cross validation. However, in the interest of space, we do not discuss this variation.

3.5 Prediction Cube Materialization

Although the concept of prediction cubes is intuitive, supporting prediction cube navigation is very computationally costly. Thus, to achieve acceptable interactive responses, materializing the cell values at different levels is generally necessary. For simplicity, in this paper, we only consider *full materialization*, i.e., materializing all the cell values for all possible levels. Partial materialization with budget constraints can be done by extending the full materialization techniques developed in this paper using the partial materialization techniques developed for data cubes, e.g., [11].

Definition 8: Full materialization table. *The full materialization table of a prediction cube is a table of schema* $[Z_1, \ldots, Z_d, M]$ *that contains all the cell values of the cube at all possible levels. That is, the table contains a tuple* $[v_1, \ldots, v_d, m(v_1, \ldots, v_d)]$, *where* $m(v_1, \ldots, v_d)$ *is the value in the cube cell* $[v_1, \ldots, v_d]$, *for each* $v_i \in Z_i^{(l)}$, *for each i and l.*

Note that values of Z_i in the given data table **D** are from domain $Z_i^{(1)}$, the least general domain. However, the values of attribute Z_j in the full materialization table are from the union of all the domains on that dimension, i.e., $\cup_l Z_i^{(l)}$.

A brute-force way to generate the full materialization table for a prediction cube is to exhaustively build a model and evaluate it for each cell, for each level. That means we need to build $(\sum_l |Z_1^{(l)}|)\times\cdots\times(\sum_l |Z_d^{(l)}|)$ models. We call this method the *exhaustive method*. Note that the sizes of the training data for those models are dramatically different. To one extreme, consider the cells in the cube at the lowest level [1, …, 1]. The size of the data falling into each of such cells is small. That means building a model for such a cell is relatively less expensive. To another extreme, consider the cell in the cube at the most general level. In this case, the training data for that cell is the whole dataset **D**. That means building a model for that cell requires extremely large resources. Further, it is very likely that building the single model of the most general cell is much more expensive than building the models for all the cells at the lowest level. This observation points out a great computational challenge in prediction cube materialization. If we do not adapt machine learning algorithms for data cubes, repeated model construction for $(\sum_l |Z_1^{(l)}|)\times\cdots\times(\sum_l |Z_d^{(l)}|)$ times seems to be unavoidable, and the large resource requirements for cells at high levels make the situation even worse. Thus, we propose to compose models, rather than building models from scratch repeatedly.

4. General Computational Techniques

The key idea of our model composition technique is scoring function decomposition. In this section, we define two kinds of decomposable scoring functions and show that decomposable scoring functions allow data cube computation techniques to be applied to prediction cube computation. Then, in the next section, we develop decomposable scoring functions for several commonly used machine learning models.

4.1 Base Subsets

Intuitively, we only build predictive models for the subsets at the lowest level, i.e., at level [1, …, 1]. Then, to compute the cube cell values of higher levels, we combine the results generated by the models of the lowest-level subsets. We call these lowest-level subsets the *base subsets*, denoted by $b_1(\mathbf{D})$, …, $b_B(\mathbf{D})$. Note that the number B of base subsets is the product of the size of the least general domains, i.e., $|Z_1^{(1)}|\times\cdots\times|Z_d^{(1)}|$.

It can be easily seen that every multidimensional hierarchical subset $\sigma_{[v_1, \dots v_d]}(\mathbf{D})$ can be represented as the union of some base subsets of **D**. Thus, for ease of expression, we use $\sigma_{\boldsymbol{S}}(\mathbf{D}) = \cup_{i\in \boldsymbol{S}}\, b_i(\mathbf{D})$, where $\boldsymbol{S} \subseteq \{1, \dots, B\}$, to denote a multidimensional hierarchical subset. Note that not every $\boldsymbol{S} \subseteq \{1, \dots, B\}$ gives a valid hierarchical subset. However, every hierarchical subset can be represented as $\sigma_{\boldsymbol{S}}(\mathbf{D})$, for some $\boldsymbol{S}$.

For simplicity, we assume that each base subset contains a significant amount of data such that learning from that subset is reasonable. In reality, this may not be the case. One can generalize prediction cubes to provide confidence intervals as a significance judgement.

4.2 Decomposable Scoring Functions

Our model composition technique is based on the following two observations.

1. Suppose we compute the cell values of a prediction cube from the lowest level to the highest level. At the time we are building a model for subset $\sigma_{\boldsymbol{S}}(\mathbf{D})$, we have already built a model for each base subset $b_i(\mathbf{D})$, for $i \in \boldsymbol{S}$. If we can save some useful intermediate results when building the models for $b_i(\mathbf{D})$'s, we might be able to build the model for $\sigma_{\boldsymbol{S}}(\mathbf{D})$ based only on the saved intermediate results without actually accessing the data.
2. Although prediction cubes are much more complex than data cubes, the cell values in prediction cubes can still be thought of as the results of a kind of aggregation. That means we have the opportunity to apply fast data cube computation techniques to prediction cube computation.

To leverage the above two observations, we introduce the concept of decomposable scoring functions (*scoring fn* for short). For many machine learning algorithms, the prediction of a model can be modelled as finding a class label that maximizes a scoring function. Formally, consider a predictive model $h(\boldsymbol{X};\ \sigma_{\boldsymbol{S}}(\mathbf{D}))$. The prediction of $h(\boldsymbol{X};\ \sigma_{\boldsymbol{S}}(\mathbf{D}))$ on input tuple $\boldsymbol{x}$ can be modelled as maximizing a scoring function $Score(y \mid \boldsymbol{x};\ \sigma_{\boldsymbol{S}}(\mathbf{D}))$; i.e.,

$$h(\boldsymbol{x};\ \sigma_{\boldsymbol{S}}(\mathbf{D})) = \operatorname{argmax}_y Score(y \mid \boldsymbol{x};\ \sigma_{\boldsymbol{S}}(\mathbf{D})).$$

Usually, $Score(y \mid \boldsymbol{x};\ \sigma_{\boldsymbol{S}}(\mathbf{D}))$ has probabilistic meaning; i.e., we expect that, fixing $\boldsymbol{x}$, $Score(y \mid \boldsymbol{x};\ \sigma_{\boldsymbol{S}}(\mathbf{D}))$ has the same maximum as $p(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x},\ \sigma_{\boldsymbol{S}}(\mathbf{D}))$.

Definition 9: Distributive decomposability. *A scoring fn* $Score(y \mid \boldsymbol{x};\ \sigma_{\boldsymbol{S}}(\mathbf{D}))$ *is distributively decomposable if*

$$Score(y \mid \boldsymbol{x};\ \sigma_{\boldsymbol{S}}(\mathbf{D})) = F(\ \{Score(y \mid \boldsymbol{x};\ b_i(\mathbf{D})) : i \in \boldsymbol{S}\}\ ),$$

where F is a distributive aggregate function as defined in [10], *e.g. SUM. A predictive model based on a distributively decomposable scoring function is called a distributively decomposable model.*

Definition 10: Algebraic decomposability. *A scoring fn* $Score(y \mid \boldsymbol{x};\ \sigma_{\boldsymbol{S}}(\mathbf{D}))$ *is algebraically decomposable if*

$$Score(y \mid \boldsymbol{x};\ \sigma_{\boldsymbol{S}}(\mathbf{D})) = F(\ \{G(y, \boldsymbol{x}, b_i(\mathbf{D})) : i \in \boldsymbol{S}\}\ ),$$

where F is an algebraic aggregate function as defined in [10], *e.g. AVERAGE, and G is a function that returns a fixed-length tuple. A predictive model based on an algebraically decomposable scoring function is called an algebraically decomposable model.*

In the next section, we show that several commonly used machine learning models are decomposable, or can be approximately decomposed. Here, we focus on how to apply data cube computation techniques to prediction cube computation.

Theorem 1. *If Score*(y | $\boldsymbol{x}$; $\sigma_{\boldsymbol{S}}(\mathbf{D})$) *is distributively* (*or algebraically*) *decomposable*, *then fixing* y, $\boldsymbol{x}$ *and* $\mathbf{D}$, *Score*(y | $\boldsymbol{x}$; $\sigma_{\boldsymbol{S}}(\mathbf{D})$) *is a distributive* (*or algebraic*) *aggregate function of* $\boldsymbol{S}$. *Further*, *fixing* Δ and h_0, *accuracy*($h(\boldsymbol{X};\sigma_{\boldsymbol{S}}(\mathbf{D}))$ | Δ), *similarity*($h_0(\boldsymbol{X})$, $h(\boldsymbol{X};\sigma_{\boldsymbol{S}}(\mathbf{D}))$ | Δ) *and KL_distance*($h_0(\boldsymbol{X})$, $h(\boldsymbol{X}; \sigma_{\boldsymbol{S}}(\mathbf{D}))$ | Δ) *are algebraic aggregate functions of* $\boldsymbol{S}$, *where* Δ *is the test dataset*, h_0 *is a user-specified model*.

Proof Sketch: The argument that *Score*(y | $\boldsymbol{x}$; $\sigma_{\boldsymbol{S}}(\mathbf{D})$) is a distributive (or algebraic) aggregate function of $\boldsymbol{S}$ directly follows the definition of distributive (or algebraic) aggregate functions. To see *accuracy*($h(\boldsymbol{X}; \sigma_{\boldsymbol{S}}(\mathbf{D}))$ | Δ) *similarity*($h_0(\boldsymbol{X})$, $h(\boldsymbol{X}; \sigma_{\boldsymbol{S}}(\mathbf{D}))$ | Δ) and *KL_distance*($h_0(\boldsymbol{X})$, $h(\boldsymbol{X}; \sigma_{\boldsymbol{S}}(\mathbf{D}))$ | Δ) are algebraic aggregate functions, note that both of them are computed from a fixed-length score array of length $|\text{Dom}(Y)|\times|\Delta|$, in which each element represents a score *Score*(y | $\boldsymbol{x}_i$; $\sigma_{\boldsymbol{S}}(\mathbf{D})$), for class y and $\boldsymbol{x}_i \in \Delta$. Since the scoring function itself is at least algebraic, computing *Score*(y | $\boldsymbol{x}_i;\sigma_{\boldsymbol{S}}(\mathbf{D})$) from lower-level intermediate results requires only fixed-length arrays. Thus, computing *accuracy*($h(\boldsymbol{X};\sigma_{\boldsymbol{S}}(\mathbf{D}))$ | Δ), *similarity*($h_0(\boldsymbol{X})$, $h(\boldsymbol{X};\sigma_{\boldsymbol{S}}(\mathbf{D}))$ | Δ) and *KL_distance*($h_0(\boldsymbol{X})$, $h(\boldsymbol{X}; \sigma_{\boldsymbol{S}}(\mathbf{D}))$ | Δ) from lower-level intermediate results also requires only fixed-length arrays. So, they are algebraic aggregate functions. □

Based on Theorem 1, if the scoring function of a learning algorithm h is decomposable, then, given a dataset $\mathbf{D}$ and a labelled test set Δ, the full materialization table of a TS-accuracy cube for h can be computed by: (1) for each base subset $b_i(\mathbf{D})$ and for each $\boldsymbol{x} \in \Delta$, generating the scores (in the distributive case) or the results of the G function (in the algebraic case), (2) saving them into a DBMS that supports data cube computation and user-defined aggregate functions, and (3) using the DBMS to generate the full materialization table.

To materialize model-similarity (distance) cubes, given h_0, $\mathbf{D}$ and Δ, we first use h_0 to predict the class label for each $\boldsymbol{x} \in \Delta$, and form a labelled test set Δ'. Then, we can use the same computation technique as described above. The materialization of predictiveness cubes can be computed similarly. In the interest of space, we omit it.

Note that, to let this mechanism work, given a test example $\boldsymbol{x}$, we assume that a decomposable model $h(\boldsymbol{X}; \sigma_{\boldsymbol{S}}(\mathbf{D}))$ not only predicts the class label for $\boldsymbol{x}$, but also have the ability to output *Score*(y | $\boldsymbol{x}$; $\sigma_{\boldsymbol{S}}(\mathbf{D})$) for every class label y. Further, if $h(\boldsymbol{X}; \sigma_{\boldsymbol{S}}(\mathbf{D}))$ is algebraically decomposable, it can also output $G(y, \boldsymbol{x}, \sigma_{\boldsymbol{S}}(\mathbf{D}))$.

4.3 Implementation in a DBMS

We now discuss how prediction cube materialization can be implemented in a DBMS. Although the accuracy function, prediction similarity function and KL-distance function can be implemented directly as user-defined algebraic aggregate functions, we do not do so because of their complexity. The following description uses extended SQL `GROUP BY` clauses that support a `CUBE` operator similar to [10]. However, in contrast to [10], instead of using `ROLL UP`, we assume the `CUBE` operator will take care of hierarchical roll-up for each dimension.

In the following, we show the algorithm to compute the full materialization table for a TS-accuracy cube given a dataset $\mathbf{D}$ of schema [$\boldsymbol{Z}$, $\boldsymbol{X}$, Y], a test set Δ and a decomposable machine learning algorithm h. The algorithm for distributive decomposable models is:

1. *Generate the intermediate results for each base subset*: We first create a score table with schema [$\boldsymbol{Z}$, *TID*, Y, *Score*], where *TID* is the test example ID. Then, for each base subset $b_i(\mathbf{D})$, we train a model $h(\boldsymbol{X}; b_i(\mathbf{D}))$, and, for each test example $\boldsymbol{x}_{tid} \in \Delta$ and for each class label y, we use $h(\boldsymbol{X}; b_i(\mathbf{D}))$ to compute *Score*(y | $\boldsymbol{x}_{tid}$; $b_i(\mathbf{D})$) and save the score in ScoreTable.
2. *Materialize the score cube*: Given ScoreTable, we materialize the score cube using the following SQL query, where F is the distributive aggregate function, e.g., SUM, for the scoring function.
```
INSERT INTO ScoreCube
SELECT Z_1, …, Z_|Z|, TID, Y, F(Score)
FROM ScoreTable
GROUP BY TID, Y CUBE Z_1, …, Z_|Z|;
```
3. *Materialize the TS-accuracy cube*: We materialize the prediction cube as follows; *accuracy*(·) is a function that computes the accuracy from the scores:
```
INSERT INTO PredictionCube
SELECT Z_1, …, Z_|Z|, accuracy(TID, Y, Score)
FROM ScoreCube
GROUP BY Z_1, …, Z_|Z|;
```

The algorithm for algebraic decomposable models is similar, and we omit it for lack of space.

Proposition 1. *The algorithms for distributive and algebraic decomposable models correctly generate the full materialization table of a TS-accuracy cube that uses a distributively or algebraically decomposable machine learning algorithm by building only* $|Z_1^{(1)}|\times\cdots\times|Z_d^{(1)}|$ *models*.

4.4 Computational Complexity

We now compare the time complexity of the proposed method with that of the exhaustive method. Since the complexity depends on the chosen base learning algorithm and the sizes of the subsets corresponding to the cube cells, we first introduce some notation:

- $f_{train}(n)$ and $f_{test}(n)$ denotes the training time and testing time of a model on a dataset of size n using the chosen learning algorithm.
- *Levels* denotes the set of all possible levels.
- $n_{[l_1,\ldots,l_d]}$ denotes the (average) size of the subset of data corresponding to a cube cell at level $[l_1, \ldots, l_d] \in$ *Levels*, where l_i is the level of the ith dimension.

The training complexity of the exhaustive method is:

$$\sum\nolimits_{[l_1,\ldots,l_d]\in Levels}\left(|Z_1^{(l_1)}|\times..\times|Z_d^{(l_d)}|\times f_{train}(n_{[l_1,\ldots,l_d]})\right).$$

The training complexity of our decomposable method is:

$$|Z_1^{(1)}| \times .. \times |Z_d^{(1)}| \times f_{train}(n_{[1,...,1]}) .$$

It can be easily seen that our method is much more efficient than the exhaustive method. Further, our method only builds models for base subsets, which usually can fit in memory individually. However, for the exhaustive method, we cannot avoid building models for very large subsets, including the entire (disk-resident) dataset.

The testing complexity of the exhaustive method is:

$$\sum_{[l_1,...,l_d] \in Levels} \left(|Z_1^{(l_1)}| \times .. \times |Z_d^{(l_d)}| \times f_{test}(n_{[l_1,...,l_d]}) \right).$$

The testing complexity of our decomposable method is:

$$|Z_1^{(1)}| \times .. \times |Z_d^{(1)}| \times f_{train}(n_{[1,...,1]}) + \sum_{[l_1,...,l_d] \in (Levels - \{[1,...,1]\})} \left(|Z_1^{(l_1)}| \times .. \times |Z_d^{(l_d)}| \times c \right)$$

where c is the cost depending on the data cube computation technique used and the hierarchy structure. Note that c does not depend on the size of any cell subset. Usually, c is similar to f_{test} if it is not smaller than f_{test}. Thus, the testing complexity of our method is usually at least as good as the exhaustive method.

5. Scoring Function Decomposition

Having presented our general prediction cube computation technique, we now apply the technique to obtain model composition methods for several machine learning algorithms by deriving the decomposable scoring functions for them. The first method is a probability-based ensemble suitable for any machine-learning model that has the ability to output class probabilities. Then, exact model composition methods for Naïve Bayes classifiers and density-estimation-based classifiers are presented. "Exact" means the composed model is exactly the same as the model built directly from the data.

5.1 Probability Based Ensembles

Suppose the machine-learning algorithm h can output class-probabilities. Let $h(y \mid \boldsymbol{x}; \sigma_{\boldsymbol{S}}(\mathbf{D}))$ denote model $h(\boldsymbol{X}; \sigma_{\boldsymbol{S}}(\mathbf{D}))$'s estimate of the conditional class-probability, i.e., $p(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, \sigma_{\boldsymbol{S}}(\mathbf{D}))$. Suppose we have another predictive model $g(\boldsymbol{X})$ that, given an input $\boldsymbol{x}$, predicts from which base subset $b_i(\mathbf{D})$ that $\boldsymbol{x}$ comes. We also assume that $g(\boldsymbol{X})$ can output probabilities. Let $g(b_i \mid \boldsymbol{x})$ denote $g(\boldsymbol{X})$'s estimate of the probability that $\boldsymbol{x}$ comes from $b_i(\mathbf{D})$, i.e., $p(\boldsymbol{X}{\sim}b_i(\mathbf{D}) \mid \boldsymbol{X}{=}\boldsymbol{x})$, where $\boldsymbol{X}{\sim}b_i(\mathbf{D})$ denotes the event that the test data comes from $b_i(\mathbf{D})$. Then, we propose to compose $h(\boldsymbol{X}; \sigma_{\boldsymbol{S}}(\mathbf{D}))$ using $h(\boldsymbol{X}; b_i(\mathbf{D}))$ and $g(\boldsymbol{X})$.

Given a selection $\sigma_{\boldsymbol{S}}$ and a *base learning algorithm h* (e.g., decision tree), the model of the probability-based ensemble $h_{PBE}(\boldsymbol{X}; \sigma_{\boldsymbol{S}}(\mathbf{D}))$ is constructed by combining $h(\boldsymbol{X}; b_i(\mathbf{D}))$ for all $i \in \boldsymbol{S}$, with weights given by $g(\boldsymbol{X})$. Formally, the output of $h_{PBE}(\boldsymbol{X}; \sigma_{\boldsymbol{S}}(\mathbf{D}))$ on input $\boldsymbol{x}$ is defined as:

$$h_{PBE}(\boldsymbol{x}; \sigma_{\boldsymbol{S}}(\mathbf{D})) = \arg\max_y Score_{PBE}(y \mid \boldsymbol{x}; \sigma_{\boldsymbol{S}}(\mathbf{D})) \text{, where}$$

$$Score_{PBE}(y \mid \boldsymbol{x}; \sigma_{\boldsymbol{S}}(\mathbf{D})) = \sum_{i \in \boldsymbol{S}} \left(h(y \mid \boldsymbol{x}; b_i(\mathbf{D})) \cdot g(b_i \mid \boldsymbol{x}) \right).$$

Note that since $Score_{PBE}(\cdot)$ can also be applied to base subsets $b_i(\mathbf{D})$, the scoring function can be written as

$$Score_{PBE}(y \mid \boldsymbol{x}; \sigma_{\boldsymbol{S}}(\mathbf{D})) = \sum_{i \in \boldsymbol{S}} Score_{PBE}(y \mid \boldsymbol{x}; b_i(\mathbf{D})) .$$

Theorem 2. *If the probability estimates of* $h(\boldsymbol{X}; b_i(\mathbf{D}))$ *and* $g(\boldsymbol{X})$ *are correct, i.e.,* $h(y \mid \boldsymbol{x}; b_i(\mathbf{D})) = p(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, b_i(\mathbf{D}))$ *and* $g(b_i \mid \boldsymbol{x}) = p(\boldsymbol{X}{\sim}b_i(\mathbf{D}) \mid \boldsymbol{X}{=}\boldsymbol{x})$, *then the combined model* $h_{PBE}(\boldsymbol{X}; \sigma_{\boldsymbol{S}}(\mathbf{D}))$ *is optimal, i.e.,* $\text{argmax}_y\ Score_{PBE}(y \mid \boldsymbol{x}; \sigma_{\boldsymbol{S}}(\mathbf{D})) = \text{argmax}_y\ p(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, \sigma_{\boldsymbol{S}}(\mathbf{D}))$.

Proof: Note that $\sigma_{\boldsymbol{S}}(\mathbf{D})$ in $p(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, \sigma_{\boldsymbol{S}}(\mathbf{D}))$ can be interpreted as the fact that the test example $\boldsymbol{x}$ comes from $\sigma_{\boldsymbol{S}}(\mathbf{D})$. Thus, we rewrite $p(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, \sigma_{\boldsymbol{S}}(\mathbf{D}))$ as $p(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, \boldsymbol{X}{\sim}\sigma_{\boldsymbol{S}}(\mathbf{D}))$, where $\boldsymbol{X}{\sim}\sigma_{\boldsymbol{S}}(\mathbf{D})$ denotes the event that the test example comes from $\sigma_{\boldsymbol{S}}(\mathbf{D})$. By the definition of conditional probability, we have the following.

$$p(Y = y \mid \boldsymbol{X} = \boldsymbol{x}, \boldsymbol{X} \sim \sigma_{\boldsymbol{S}}(\mathbf{D})) = \frac{p(Y = y, \boldsymbol{X} \sim \sigma_{\boldsymbol{S}}(\mathbf{D}) \mid \boldsymbol{X} = \boldsymbol{x})}{p(\boldsymbol{X} \sim \sigma_{\boldsymbol{S}}(\mathbf{D}) \mid \boldsymbol{X} = \boldsymbol{x})}$$

Since we are only interested in the y with the highest class probability, the denominator $p(\boldsymbol{X}{\sim}\sigma_{\boldsymbol{S}}(\mathbf{D}) \mid \boldsymbol{X} = \boldsymbol{x})$ is a constant and does not affect the choice of y. Let $Z = 1/p(\boldsymbol{X}{\sim}\sigma_{\boldsymbol{S}}(\mathbf{D}) \mid \boldsymbol{X} = \boldsymbol{x})$.

Because $b_1(\mathbf{D}), b_2(\mathbf{D}), \ldots, b_B(\mathbf{D})$ are disjoint,

$$\begin{aligned} & p(Y = y \mid \boldsymbol{X} = \boldsymbol{x}, \boldsymbol{X} \sim \sigma_{\boldsymbol{S}}(\mathbf{D})) \\ &= Z \cdot \sum_{i \in \boldsymbol{S}} p(Y = y, \boldsymbol{X} \sim b_i(\mathbf{D}) \mid \boldsymbol{X} = \boldsymbol{x}) \\ &= Z \cdot \sum_{i \in \boldsymbol{S}} \left(p(Y = y \mid \boldsymbol{X} \sim b_i(\mathbf{D}), \boldsymbol{X} = \boldsymbol{x}) \cdot p(\boldsymbol{X} \sim b_i(\mathbf{D}) \mid \boldsymbol{X} = \boldsymbol{x}) \right) \end{aligned}$$

If $h(y \mid \boldsymbol{x}; b_i(\mathbf{D})) = p(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, \boldsymbol{X}{\sim}b_i(\mathbf{D}))$ and $g(b_i \mid \boldsymbol{x}) = p(\boldsymbol{X}{\sim}b_i(\mathbf{D}) \mid \boldsymbol{X}{=}\boldsymbol{x})$, then $Z \cdot Score_{PBE}(y \mid \boldsymbol{x}; \sigma_{\boldsymbol{S}}(\mathbf{D})) = p(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, \boldsymbol{X}{\sim}\sigma_{\boldsymbol{S}}(\mathbf{D}))$. So, $\text{argmax}_y\ Score_{PBE}(y \mid \boldsymbol{x}; \sigma_{\boldsymbol{S}}(\mathbf{D})) = \text{argmax}_y\ p(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, \boldsymbol{X}{\sim}\sigma_{\boldsymbol{S}}(\mathbf{D}))$. □

Note that, although $h(y \mid \boldsymbol{x}; b_i(\mathbf{D}))$ and $g(b_i \mid \boldsymbol{x})$ may not be good probability estimators, it is still likely that $\text{argmax}_y\ Score_{PF}(y \mid \boldsymbol{x}; \sigma_{\boldsymbol{S}}(\mathbf{D})) \cong \text{argmax}_y\ p(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x}, \sigma_{\boldsymbol{S}}(\mathbf{D}))$. Thus, in practice, $Score_{PF}(y \mid \boldsymbol{x}; \sigma_{\boldsymbol{S}}(\mathbf{D}))$ can be thought of as a good heuristic method to determine the class label of a given input $\boldsymbol{x}$.

Proposition 2. $Score_{PBE}(\cdot)$ *is distributively decomposable with aggregate function*: *SUM*.

5.2 Naïve Bayes Classifiers

Naïve Bayes classifiers are a type of predictive model that is simple but sometimes performs surprisingly well. The key assumption behind Naïve Bayes classifiers is that, given the class label Y, each predictor attribute $X_j \in \boldsymbol{X}$ is independent of the others. The prediction of a Naïve Bayes model on input $\boldsymbol{x}=[x_1, \ldots, x_m]$ is based on the following formula.

$$p(Y = y \mid \boldsymbol{X} = \boldsymbol{x}) = \frac{p(Y = y)}{p(\boldsymbol{X} = \boldsymbol{x})} \cdot \prod_{j=1}^{m} p(X_j = x_j \mid Y = y) \cdot$$

The output is the class y that maximizes the above conditional probability. Note that for any given input $\boldsymbol{x}$, $p(\boldsymbol{X}{=}\boldsymbol{x})$ is the same for each class y. Thus, for prediction purposes, we do not need $p(\boldsymbol{X}{=}\boldsymbol{x})$.

Let $c(y; \sigma_{\boldsymbol{S}}(\mathbf{D})) = |\sigma_{\boldsymbol{S} \wedge Y=y}(\mathbf{D})|$ denote the number of examples in $\sigma_{\boldsymbol{S}}(\mathbf{D})$ whose class label is y, and $c_j(x_j, y; \sigma_{\boldsymbol{S}}(\mathbf{D})) = |\sigma_{\boldsymbol{S} \wedge Y=y \wedge X_j=x_j}(\mathbf{D})|$ denote the number of examples in $\sigma_S(\mathbf{D})$ whose class label is y and the j-th predictor attribute has value x_j. Then, the terms on the right hand side of the Naïve Bayes formula can be estimated as follows.

- $p(Y{=}y) \cong c(y;\ \sigma_S(\mathbf{D})) / |\sigma_S(\mathbf{D})|$.
- $p(X_j{=}x_j \mid Y{=}y) \cong c_j(x_j, y;\ \sigma_S(\mathbf{D})) / c(y;\ \sigma_S(\mathbf{D}))$.

To handle the case in which some counts are zero, a commonly used trick is to initialize each count to one instead of zero [17]. Note that, continuous attributes can be handled by kernel density estimator, which will be discussed in Section 5.3.

Given a selection σ_S, the Naïve Bayes model $h_{NB}(\boldsymbol{X};\ \sigma_S(\mathbf{D}))$ is constructed by combining the counts. Formally, the output of $h_{NB}(\boldsymbol{X};\ \sigma_S(\mathbf{D}))$ on input $\boldsymbol{x}=[x_1, \ldots, x_m]$ is defined as follows.

$$h_{NB}(\boldsymbol{x};\sigma_S(\mathbf{D})) = \arg\max_y \ Score_{NB}(y \mid \boldsymbol{x};\sigma_S(\mathbf{D}))\text{, where}$$

$$Score_{NB}(y \mid \boldsymbol{x};\sigma_S(\mathbf{D})) = \frac{\sum_{i\in S} c(y;b_i(\mathbf{D}))}{\sum_{i\in S} |b_i(\mathbf{D})|} \cdot \prod_{j=1}^{m} \frac{\sum_{i\in S} c_j(x_j, y; b_i(\mathbf{D}))}{\sum_{i\in S} c(y;b_i(\mathbf{D}))}.$$

Proposition 3. *The model $h_{NB}(\boldsymbol{X};\ \sigma_S(\mathbf{D}))$ using the above scoring function is the same Naïve Bayes model of $\sigma_S(\mathbf{D})$.*

Proposition 4. *$Score_{NB}(\cdot)$ is algebraically decomposable.*

5.3 Kernel-density Based Classifiers

By the definition of conditional probability, we have

$$p(Y=y \mid \boldsymbol{X}=\boldsymbol{x}) = \frac{p(\boldsymbol{X}=\boldsymbol{x} \mid Y=y)\cdot p(Y=y)}{p(\boldsymbol{X}=\boldsymbol{x})}.$$

For prediction purposes, given input $\boldsymbol{x}$, $p(\boldsymbol{X}{=}\boldsymbol{x})$ is a constant C for each class y. Thus, estimating $p(Y{=}y \mid \boldsymbol{X}{=}\boldsymbol{x})$ is equivalent to estimating $p(\boldsymbol{X}{=}\boldsymbol{x} \mid Y{=}y)$ and $p(Y{=}y)$. Note that $p(Y{=}y)$ can be easily estimated by the fraction of training data belonging to class y. However, $p(\boldsymbol{X}{=}\boldsymbol{x} \mid Y{=}y)$ cannot generally be estimated by simple counting because the feature space can be much larger than the number of examples. Further, if $\boldsymbol{X}$ contains continuous attributes, $p(\boldsymbol{X}{=}\boldsymbol{x} \mid Y{=}y)$ becomes a probability density function (we still use p to denote probability densities). In many applications, $\boldsymbol{X}$ is usually a mixture of categorical and continuous attributes.

One commonly used method to estimate $p(\boldsymbol{X}{=}\boldsymbol{x} \mid Y{=}y)$ is the kernel density estimation, in which, given a training dataset $\sigma_S(\mathbf{D})$,

$$p(\boldsymbol{X}=\boldsymbol{x} \mid Y=y, \sigma_S(\mathbf{D})) = \frac{1}{|\sigma_{S\wedge Y=y}(\mathbf{D})|}\sum_{[\boldsymbol{x}_k, y_k]\in\sigma_{S\wedge Y=y}(\mathbf{D})} K(\boldsymbol{x}, \boldsymbol{x}_k),$$

where $K(\cdot,\cdot)$ is the kernel function, and $\sigma_{S\wedge Y=y}(\mathbf{D})$ is the set of examples in $\sigma_S(\mathbf{D})$ whose class labels are y. Note that, if we use the counts to estimate $p(Y{=}y \mid \sigma_S(\mathbf{D}))$, i.e.,

$$p(Y=y \mid \sigma_S(\mathbf{D})) = |\sigma_{S\wedge Y=y}(\mathbf{D})| / |\sigma_S(\mathbf{D})|,$$

then, for our prediction purpose, we do not even need to estimate $p(Y{=}y \mid \sigma_S(\mathbf{D}))$ because

$$p(Y=y \mid \boldsymbol{X}=\boldsymbol{x}, \sigma_S(\mathbf{D})) = \frac{1}{C}\frac{1}{|\sigma_S(\mathbf{D})|}\sum_{[\boldsymbol{x}_k, y_k]\in\sigma_{S\wedge Y=y}(\mathbf{D})} K(\boldsymbol{x}, \boldsymbol{x}_k),$$

where $|\sigma_S(\boldsymbol{D})|$ is a constant for any class y. Thus, given input $\boldsymbol{x}$, the kernel-density-based classifier $h_{KDC}(\boldsymbol{x};\sigma_S(\boldsymbol{D}))$ is:

$$h_{KDC}(\boldsymbol{x};\sigma_S(\boldsymbol{D})) = \arg\max_y \ Score_{KDC}(y \mid \boldsymbol{x};\sigma_S(\boldsymbol{D}))\text{, where}$$

$$Score_{KDC}(y \mid \boldsymbol{x};\sigma_S(\boldsymbol{D})) = \sum_{[\boldsymbol{x}_k, y_k]\in\sigma_{S\wedge Y=y}(\boldsymbol{D})} K(\boldsymbol{x}, \boldsymbol{x}_k).$$

If $\boldsymbol{X}$ only contains continuous attributes, the following Gaussian kernel is the most commonly used kernel.

$$K(\boldsymbol{x}, \boldsymbol{x}_k) = \frac{1}{\sqrt{2\pi}s} e^{-\frac{(\boldsymbol{x}-\boldsymbol{x}_k)^2}{2s^2}},$$

where s is the standard deviation. If $\boldsymbol{X}$ is a mixture of categorical and continuous attributes, the following product kernel is a commonly used one (cf. [3, 15]). Let $\boldsymbol{x}[X_j]$ denote the projection of $\boldsymbol{x}$ on $X_j \in \boldsymbol{X}$.

$$K(\boldsymbol{x}, \boldsymbol{x}_k) = \prod_{A_j\in \boldsymbol{P}} W_j(\boldsymbol{x}[X_j], \boldsymbol{x}_k[X_j]),$$

where $W_j(\cdot,\cdot)$ is the kernel function for the j-th attribute. Usually, if X_j is categorical, $W_j(\cdot,\cdot)$ is a smoothed indicator function; i.e. $W_j(\boldsymbol{x}[X_j], \boldsymbol{x}_k[X_j]) = 1-\lambda_j$ if $\boldsymbol{x}[X_j] = \boldsymbol{x}_k[X_j]$; otherwise, $W_j(\boldsymbol{x}[X_j], \boldsymbol{x}_k[X_j]) = \lambda_j$, where λ_j is the smoothing parameter. If X_j is continuous, $W_j(\cdot,\cdot)$ is usually a Gaussian kernel. One can also group a set of k continuous attributes and use a k-dimensional kernel. Note that, when using the product kernel, we have made the *local independence* assumption; i.e., in a small region around a training example $\boldsymbol{x}_k$, $X_1,\ldots,X_m$ are mutually independent given Y.

Proposition 5. *$Score_{KDC}(\cdot)$ is distributively decomposable with aggregate function SUM.*

Note that for Naïve Bayes classifiers, if X_j is a continuous attribute, we can use the kernel density to estimate $p(X_j \mid Y)$. The kernel-based Naïve Bayes model is also algebraically decomposable.

If the size of $\sigma_S(\mathbf{D})$ is too large, making predictions using a kernel-density-based classifier can be very costly. In this case, we can perform clustering first, and use the clusters to estimate the density, where the number of clusters is significantly smaller than the size of $\sigma_S(\mathbf{D})$. Clustering-based density estimation techniques include [20, 15, 3]. Note that we can easily define a distributively decomposable scoring function for such clustering-based classifiers. In the interest of space, we omit this.

6. Experimental Results

In this section, we evaluate the proposed methods. We show that (1) materializing prediction cubes based on decomposable scoring functions is, as expected, much faster than the exhaustive method, (2) our method scales almost linearly in the number of base subsets, and (3) the accuracy of probability-based ensembles is good. We also apply prediction cubes to a real-world dataset to illustrate its use in exploratory analysis. Note that, since the Naïve Bayes and kernel-density-based classifiers produced by our decomposition techniques are exactly the same as the original ones, it is not necessary to evaluate their accuracy.

For brevity, we denote probability-based ensembles as **PBE**, Naïve Bayes classifier as **NB,** and the kernel-density-based classifier as **KDC**. We consider the following base learning algorithms for PBE:

- **J48**: Weka's [19] C4.5 decision trees [18].
- **K2**: A Bayesian Network structure learner with the K2 [5] search algorithm (each node has < 6 parents).
- **RF**: Random Forest [4] with 20 random trees. For each split, log(M+1) random attributes are examined, where M is the number of predictor attributes.

For the following experiments, we used in-memory implementations of cubes and all learning algorithms. Clearly, this does not affect our accuracy results. As for performance, the improvement relative to the exhaustive approach is easily seen from our results, and as we discuss, the improvement will increase significantly with disk-resident data. Although our current implementation is in-memory, in Section 4.3 we showed how cube computation can be scaled for disk-resident data using a DBMS that supports cubes and user-defined aggregate functions. Finally, disk-based learning algorithms (many of which are described in the literature) can be used as base learners, thereby scaling our current implementation.

6.1 Efficiency and Scalability

We use synthetic datasets with schema [Z_1, Z_2, Z_3, X_1, …, X_6, Y], where Z_1, Z_2, Z_3 are the dimensions. The domains of Z_1 and Z_2 contain 10 values in a 3-level hierarchy as shown in Figure 4 (a). Z_3 has a 2-level hierarchy with a variable number n of leaf nodes, as shown in Figure 4 (b). By varying n, we can generate datasets with different numbers of base subsets. X_1, …, X_6 are the predictor attributes. X_1, …, X_4 are numeric attributes drawn uniformly from the range [0,1]. X_5 and X_6 are categorical attributes taking values from 0 to 9. Y, the class label, is a binary attribute generated by the following rules:

Condition	Generation function
When Z_1>1	$Y = I(4X_1+3X_2+2X_3+X_4+0.4X_6 > 7)$
else when Z_3 mod 2 = 0	$Y = I(2X_1+2X_2+3X_3+3X_4+0.4X_6 > 7)$
else	$Y = I(0.1X_5+X_1>1)$

First, we compared our prediction cube computation technique with the exhaustive method. We varied n from 1 to 5 and generated 400 records for each base subset, leading to five datasets with 40K to 200K records. We computed a TS-accuracy cube on a test set of 1000 records for each dataset and for each of the following methods: J48-PBE (J48-based PBE), RF-PBE (RF-based PBE), and the decomposable versions of KDC and NB. Then, the corresponding exhaustive methods, which are denoted by an "ex" suffix, were computed. The result in Figure 5 (a), with y-axis representing elapsed time in seconds, shows that our technique dramatically improves the efficiency of prediction cube computation.

Second, we generated a larger dataset to study scalability. Varying n from 1 to 5 and using 2000 records for each base subset, we obtain five datasets with 200K to

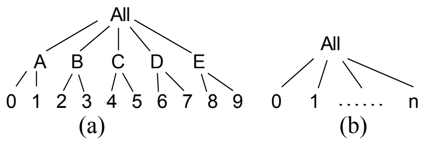


Figure 4. The dimension hierarchies of synthetic datasets

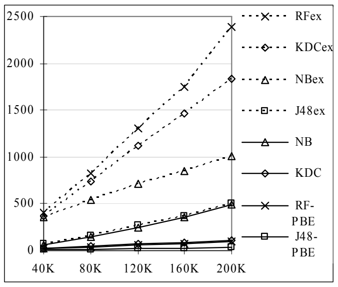
1M records. Figure 5 (b) shows the results, which demonstrate that our methods scale linearly in the number of base datasets. The linear scalability comes from the fact that we do not build or test any model beyond the base subsets. That means no matter what kind of base learning algorithm we use, we still scale linearly in the number of base subsets. In contrast, if we do not use the model composition technique and want to build a model on the union of k base subsets, we depend critically on the scalability properties of the learning algorithm. Of course, our approach also relies on the scalability of the learning algorithm, but to a lesser extent; i.e., the algorithm needs only to scale to the much smaller base subsets.

To understand where the time is spent, we show Figure 6. J48-PBE and RF-PBE spend most of their time on training, while KDC and NB spend most of their time on testing. Because KDC and NB both use kernel density estimators, training can be very fast. However, making a prediction (i.e., testing) requires going through every training example. Also note that NB performs worse than one would expect. This is because the scoring function for NB is algebraically decomposable, which requires more resources to compute than a distributive scoring function.

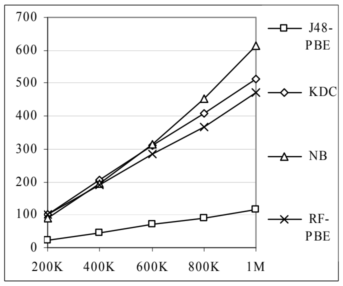
Figure 6 also suggests that if we want to build several prediction cubes on the same data table but using different test sets and/or different measurements, storing the models built on the base subsets can reduce execution time by 60% for J48-PBE and 90% for RF-PBE.

6.2 Evaluation of PBE on UCI Datasets

Although PBE has better computational efficiency, unlike NB and KDC, a PBE of base models trained on base subsets is not identical to the model built using the same base learning algorithm on the union of those base subsets. Thus, we use 8 UCI datasets to evaluate the accuracy of PBE. The results are shown in Table 1. Each row of Table 1 is the result of an experiment of a UCI dataset with K partitions. We first partition the dataset into K disjoint partitions of examples. Each "PBE" column shows the accuracy of PBE using the base learning algorithm indicated by the header. The PBE is an ensemble of K base models, each of which is trained on a single partition. The "Orig.–PBE" columns show the difference in accuracy between "the model using the base learning algorithm trained on the full dataset" and "the PBE of base models trained on partitions." The results are generated by 10-fold cross validation, and the error ranges are 95% confidence intervals using a two-sided t-distribution. A difference is statistically significant if the interval does not include 0.



(a)



(b)

Figure 5. Efficiency and scalability

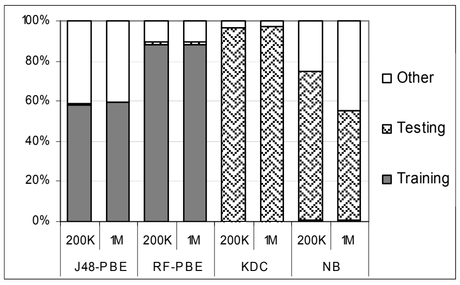


Figure 6. Time spent on training testing or other computation

As expected, when the number of partitions increases, the accuracy of PBE decreases. However, in most cases, the amount of decrease is within 5 percent point.

6.3 Evaluation of PBE in the Prediction Cube Setting

Although accuracy decrease of PBE is small, it is still a concern. Therefore we investigated whether the decrease is bounded and at what level, using synthetic datasets similar to those described in Section 6.1.

We created two datasets with different characteristics, called *deep* and *flat*, containing attributes Z_1, Z_2, Z_3, X_1, … , X_6, Y. Z_1 and Z_2 are dimensions with 2-level hierarchies with three leaves each. For the *deep* dataset, the Z_3 dimension is a binary tree with 7 levels and 64 leaf nodes. For the *flat* dataset, Z_3 is a 2-level hierarchy with 126 leaf nodes. The remaining attributes are as described in Section 6.1, and we produced these datasets by generating 1000 records for each base subset using the decision rule from Section 6.1. We built TS-accuracy cubes using J48-PBE and RF-PBE and compared them with the TS-accuracy cubes built using the decision rule that generates the data.

Figure 7 shows the results. Each point (x, y) in a chart indicates the average absolute error from a cell value generated by a PBE of x base models w.r.t. the same cell value generated by the true decision rule. The x-axes are log-scaled. The main observation is that the error converges to around 0.08 (i.e., 8%) for RF-PBE. Note that the error caused by the base learning algorithm (RF or J48) is the y-value at x=1 for each curve (x=1 means there is only one base model). Thus, the error cased by PBE is reasonably small: less then 0.03 (or 3%) for RF-PBE. However, more research is still needed to reduce the error further and understand the characteristics of PBE better.

6.4 Data Analysis Using Prediction Cubes

We now illustrate the use of prediction cubes to perform real-world analysis using a dataset of population estimates from the American Community Survey [1]. The prediction characteristic we consider is how a person's profile affects his or her income. We chose *Location* and *Age* as dimension attributes. *Location* contains hierarchical levels *State* and *Region*, and *Age* is discretized into 5 groups representing 10-year periods, each of which is further divided into two 5-year periods. Each cell in the cube represents the accuracy of the model (using a person’s profile to predict his/her income) trained on the data in that cell when tested against a test set.

In the first experiment, we selected a test set of 1000 random samples of personal profiles from California with *Age* 30 to 40. The goal is to identify subsets in the data that are similar to the chosen test set. Table 2 shows (part of) the prediction cube. The *All* value for *Location* or *Age* means the attribute has been rolled-up to the coarsest level. As we drill down to the *Region* level, we see that (the model trained using) [Pacific, All] has the highest accuracy. We can drill down further to the [*State*, *10-year-group*] level, which is shown in the third block of the table; [Pacific, [30,40]] is the top cell. As we drill deeper and finally reach the [*State, 5-year-subgroup*] level, the top two cells are exactly those cells included in our sampling. These observations are consistent with the fact that the test set is drawn from California, and illustrate the natural behaviour of prediction cubes.

A more interesting observation arises at the lowest level, where we observe that all subsets containing individuals with *Age* 20 to 25 have significantly low test-accuracy. This is explained by the fact that many people in this age group are still in college, and the correlation of their income with age and location will differ considerably from the test set (a non-college age population). So they will exhibit a dramatically different pattern. We were able to confirm this observation using a cube on predictiveness of income level from education (omitted for lack of space). Note that this analysis focuses on the correlation of income with age and location, and goes well beyond simply looking at average income levels by age and location!

7. Conclusions and Related Work

Prediction cubes and their associated computational challenges are new problems in data mining. In this paper, we motivated these problems and presented some initial results. Our future directions include: (1) developing a mechanism to handle the case where some subsets do not have sufficient data to build a good model, (2) deriving decomposable scoring functions for other predictive models, (3) investigating the problem of how to make the models in prediction cubes interpretable, and (4) extending the definition of dimensions to include parameters of learning algorithms.

Table 1. Evaluation of PBE on UCI datasets

Dataset	*K*	J48		K2		RF	
		PBE	Orig.–PBE	PBE	Orig.–PBE	PBE	Orig.–PBE
adult	2	85.89	0.34 ±0.02	86.27	0.11 ±0.08	85.24	-0.73 ±0.00
	5	85.98	0.25 ±0.07	86.02	0.35 ±0.08	85.57	-1.06 ±0.05
	10	85.70	0.53 ±0.15	85.56	0.81 ±0.02	85.41	-0.90 ±0.15
	15	85.32	0.91 ±0.10	85.50	0.87 ±0.06	85.19	-0.68 ±0.15
	20	85.19	1.04 ±0.05	85.42	0.96 ±0.01	85.15	-0.64 ±0.15
kr-vs-kp	2	99.25	0.19 ±0.04	95.68	0.00 ±0.00	98.72	0.38 ±0.21
	5	97.84	1.60 ±0.31	95.34	0.34 ±0.22	98.22	0.88 ±0.02
	10	97.06	2.38 ±0.04	95.62	0.06 ±0.09	97.56	1.53 ±0.08
	15	96.34	3.10 ±0.22	95.34	0.34 ±0.08	97.06	2.03 ±0.04
	20	95.06	4.38 ±0.45	95.15	0.53 ±0.20	96.62	2.47 ±0.14
nursery	2	95.87	1.18 ±0.21	91.76	1.06 ±0.01	98.19	0.53 ±0.11
	5	94.20	2.85 ±0.26	91.27	1.54 ±0.13	96.59	2.13 ±0.12
	10	91.94	5.11 ±0.10	90.93	1.89 ±0.25	94.65	4.07 ±0.23
	15	91.28	5.77 ±0.05	90.93	1.88 ±0.25	93.60	5.12 ±0.03
	20	91.19	5.86 ±0.11	90.79	2.03 ±0.26	93.02	5.70 ±0.15
pendigits	2	94.53	1.10 ±0.12	95.84	0.59 ±0.25	98.69	0.12 ±0.03
	5	95.39	0.24 ±0.27	95.24	1.19 ±0.18	97.94	0.87 ±0.04
	10	94.95	0.68 ±0.23	94.53	1.89 ±0.13	97.21	1.61 ±0.01
	15	93.93	1.70 ±0.33	94.28	2.15 ±0.12	96.87	1.94 ±0.14
	20	94.00	1.62 ±0.17	93.78	2.64 ±0.20	96.33	2.48 ±0.13
satimage	2	84.83	1.22 ±0.61	88.75	-0.72 ±0.01	90.21	0.68 ±0.05
	5	85.71	0.34 ±0.46	87.91	0.11 ±0.03	88.97	1.92 ±0.08
	10	84.49	1.55 ±0.96	87.08	0.95 ±0.64	88.12	2.77 ±0.18
	15	84.60	1.44 ±1.37	86.81	1.22 ±0.30	87.55	3.34 ±0.10
	20	83.74	2.30 ±1.41	86.09	1.94 ±0.29	87.06	3.83 ±0.38
spambase	2	91.41	1.56 ±0.30	93.54	-0.28 ±0.33	94.63	0.72 ±0.09
	5	89.78	3.19 ±0.04	92.63	0.63 ±0.16	93.91	1.43 ±0.12
	10	87.83	5.15 ±0.59	92.50	0.76 ±0.23	92.70	2.65 ±0.06
	15	87.72	5.26 ±0.66	90.91	2.35 ±0.27	91.70	3.65 ±0.01
	20	86.89	6.09 ±0.57	91.46	1.80 ±0.61	92.13	3.22 ±0.27
waveform	2	75.76	-0.68 ±0.11	82.42	-0.70 ±0.45	84.66	-1.10 ±0.02
	5	78.90	-3.82 ±0.09	83.50	-1.78 ±0.72	84.96	-1.40 ±0.05
	10	81.42	-6.34 ±0.22	84.04	-2.32 ±1.02	85.04	-1.48 ±0.02
	15	81.06	-5.98 ±0.43	83.52	-1.80 ±0.67	85.12	-1.56 ±0.04
	20	81.34	-6.26 ±0.01	83.28	-1.56 ±0.87	85.08	-1.52 ±0.40
letter	2	86.11	1.87 ±0.23	85.00	1.47 ±0.20	95.23	0.42 ±0.03
	5	86.18	1.80 ±0.14	81.19	5.29 ±0.08	93.66	1.99 ±0.09
	10	85.10	2.88 ±0.17	76.31	10.17 ±0.08	91.75	3.90 ±0.24
	15	83.64	4.34 ±0.00	73.79	12.69 ±0.39	89.96	5.69 ±0.20
	20	82.63	5.36 ±0.19	70.96	15.52 ±0.23	88.87	6.77 ±0.03

Table 2. Test-set accuracy of selected cells in the cube

Location	Age	Accuracy
All	All	**0.782**
Pacific	All	**0.789**
East North Central	All	0.779
Mountain	All	0.772
Pacific	(30,40]	**0.835**
East North Central	(30,40]	0.785
East North Central	(40,50]	0.782
……		
CA	(35,40]	0.822
CA	(30,35]	0.817
……		
CA	(20, 25)	0.567

In related work, data cubes have been extended using association rules in [13], but association rules are quite different from the predictive models described in this paper and the particular pruning methods proposed in [13] cannot be applied to prediction cubes. Finding pairs of neighboring cells having characteristics associated with big changes in measure in a data cube was studied in [8]. However, the similarity defined in [8] is very different from the similarity between predictive model behavior. Building models in the OLAP setting was also studied in [2, 16]. [2] considered using statistical log-linear models to approximate dense regions in a data cube, while [16] considered building Bayesian Networks (BN) on data cubes to approximately answer count queries. However, their goal was to use models to compress data cubes, rather than the model-based data analysis proposed in this paper. Note that the BN learning algorithm proposed in [16] can be adapted so that it can be an instance of our decomposable method. In Machine Learning, ensembling [7] is a widely used technique to boost the accuracy of unstable learning algorithms. However, an ensemble typically consists of a set of base classifiers, each trained on a significantly large portion of the full dataset; our use of ensembles does not have this property and has not been carefully studied.

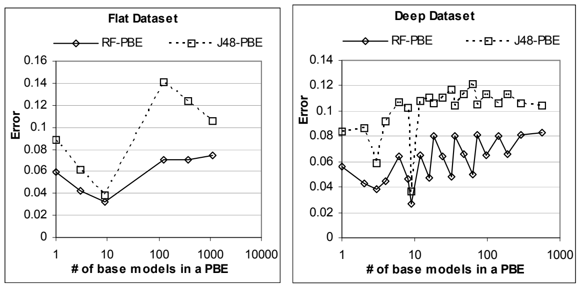


Figure 7. Evaluation of PBE on synthetic datasets

References

[1] ACS Public Use Microdata Sample (PUMS) 2003. <http://factfinder.census.gov/home/en/acs_pums_2003.html>

[2] D. Barbará and X. Wu. Loglinear-Based Quasi Cubes. *J. Intelligent Information System*, 2001.

[3] P.S. Bradley, U.M. Fayyad and C.A. Reina. Scaling EM (Expectation-Maximization) Clustering to Large Databases. *ICML 1998*.

[4] L. Breiman. Random Forests. *Machine Learning*, 2001.

[5] G.F. Cooper, E. Herskovits. A Bayesian Method for the Induction of Probabilistic Networks from Data, *Machine Learning*, 1992.

[6] A. Danna, O. Gandy. All the Glitters is not Gold: Digging Beneath the Surface of Data Mining. *J. Business Ethics, 2002*.

[7] T. G. Dietterich. Ensemble Methods in Machine Learning. *Int. Workshop on Multiple Classifier Systems* (*MCS*), 2000.

[8] G. Dong, J. Han, J. Lam, J. Pei and K. Wang. Mining Multi-dimensional Constrained Gradients in Data Cubes. *VLDB*, 2001.

[9] V. Ganti, J. Gehrke and R. Ramakrishnan. CACTUS—Clustering Categorical Data Using Summaries. *KDD* 1999.

[10] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart and M. Venkatrao. Data Cube: A Relational Aggregate Operator Generalizing Group-By, Cross-Tab, and Sub-Tables.*J. Data Mining and Knowledge Discovery*, 1997.

[11] V. Harinarayan, A. Rajaraman and J.D. Ullman. Implementing Data Cubes Efficiently. *SIGMOD*, 1996.

[12] T. Hastie and R. Tibshirani. Discriminant Analysis by Gaussian Mixtures. *J. Royal Statistical Societ,* 1996.

[13] T. Imielinski, L. Khachiyan and A. Abdulghani. Cubegrades: Generalizing Association Rules. *J. Data Mining and Knowledge Discovery*, 2002.

[14] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Stat.*, 22:79–86, 1951.

[15] Q. Li and J. Racine. Nonparametric Estimation of Distributions with Categorical and Continuous Data. *J. Multivariate Analysis*, 2003.

[16] D. Margaritis, C. Faloutsos and S. Thrun. NetCube: A Scalable Tool for Fast Data Mining and Compression. *VLDB*, 2001.

[17] T. Mitchell. *Machine Learning*, McGraw Hill, 1997.

[18] J.R. Quinlan. *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[19] I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools with Java Implementations*, Morgan Kaufmann, 2000.

[20] T. Zhang, R. Ramakrishnan and M. Livny. Fast density estimation using CF-kernel for large databases. *KDD* 1999.