© 2016 IJSRSET | Volume 2 | Issue 2 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Analytical Research on Decision Tree Algorithm and Naive Bayesian Classification Algorithm for Data Mining

Muhammad Mustaqim Rahman*[1], Tarikuzzaman Emon[2], Zonayed Ahmed[3]
*[1]Département Informatique, Universite de Lorraine, Nancy, France
[2,3] Department of Computer Science and Engineering, Stamford University Bangladesh, Dhaka, Bangladesh

ABSTRACT

The paper presents an extensive modification of ID3 (Iterative Dichotomiser) algorithm and Naïve Bayesian Classification algorithm for data mining. The automated, prospective analyses offered by data mining move beyond the analyses of past events provided by retrospective tools typical of decision support systems. Data mining tools can answer business questions that traditionally were too time consuming to resolve. They scour databases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectations. Most companies already collect and refine massive quantities of data. Data mining techniques can be implemented rapidly on existing software and hardware platforms to enhance the value of existing information resources, and can be integrated with new products and systems as they are brought on-line. This paper provides an introduction to the basic technologies of data mining. Examples of profitable applications illustrate its relevance to today's business environment as well as a basic description of how data warehouse architectures can evolve to deliver the value of data mining to end users.

Keywords: ID3, Naive Bayesian, Algorithm, Data mining, Database.

I. INTRODUCTION

Data mining is the exploration of historical data (usually large in size) in search of a consistent pattern and/or a systematic relationship between variables; it is then used to validate the findings by applying the detected patterns to new subsets of data. The roots of data mining originate in three areas: classical statistics, artificial intelligence (AI) and machine learning. Pregibon et al. [6] described data mining as a blend of statistics, artificial intelligence, and database research, and noted that it was not a field of interest to many until recently.

According to Fayyad et al. [7] data mining can be divided into two tasks: predictive tasks and descriptive tasks. The ultimate aim of data mining is prediction; therefore, predictive data mining is the most common type of data mining and is the one that has the most application to businesses or life concerns. Predictive data mining has three stages. DM starts with the collection and storage of data in the data warehouse. Data collection and warehousing is a whole topic of its own, consisting of identifying relevant features in a business and setting a storage file to document them. It also involves cleaning and securing the data to avoid its corruption. According to Kimball et al. [8], a data warehouse is a copy of transactional or non-transactional data specifically structured for querying, analyzing, and reporting. Data exploration, which follows, may include the preliminary analysis done to data to get it prepared for mining. The next step involves feature selection and reduction.

Mining or model building for prediction is the third main stage, and finally come the data post-processing, interpretation, and deployment. Applications suitable for data mining are vast and are still being explored in many areas of business and life concerns.

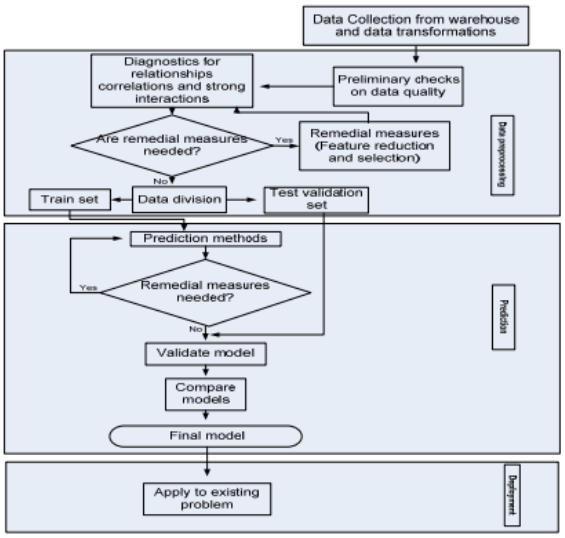


Figure 1. The stages of predictive data mining

This is because, according to Betts et al. [9], data mining yields unexpected nuggets of information that can open a company's eyes to new markets, new ways of reaching customers and new ways of doing business.

II. METHODS AND MATERIAL

To evaluate performance of data mining we are using two predictive algorithms:

A. Predictive Algorithm 1 (Decision Tree Induction)

During the late 1970s and early 1980s, J. Ross Quinlan, a researcher in machine learning, developed a decision tree algorithm known as ID3 (Iterative Dichotomiser). This work expanded on earlier work on concept learning systems, described by E. B. Hunt, J. Marin, and P. T. Stone. Quinlan later presented C4.5 (a successor of ID3), which became a benchmark to which newer supervised learning algorithms are often compared. In 1984, a group of statisticians (L. Breiman, J. Friedman, R. Olshen, and C. Stone) published the book Classification and Regression Trees (CART), which described the generation of binary decision trees. ID3 and CART were invented independently of one another at around the same time, yet follow a similar approach for learning decision trees from training tuples. These two cornerstone algorithms spawned a flurry of work on decision tree induction.

ID3, C4.5, and CART adopt a greedy (i.e., nonbacktracking) approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner. Most algorithms for decision tree induction also follow such a top-down approach, where the Algorithm: Generate decision tree. Generate a decision tree from the training tuples of data partition D.

Input

- Data partition, D, which is a set of training tuples and their associated class labels.
- Attribute list, the set of candidate attributes.
- Attribute selection method, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a slipping_attribute and possibly either a split point or splitting subset.

Output - A decision tree.

Method

- ✓ create a node N
- ✓ if tuples in D are all of the same class, C then
- ✓ return N as a leaf node labelled with the class C
- ✓ if attribute list is empty then
- ✓ return N as a leaf node labelled with the majority class in D; // majority voting
- ✓ apply Attribute selection method(D, attribute list) to find the "best" splitting criterion
- ✓ label node N with splitting criterion;
- ✓ if splitting attribute is discrete-valued and multiway splits allowed then // not restricted to binary trees
- ✓ attribute list attribute list □splitting attribute; // remove splitting attribute
- ✓ for each outcome j of splitting criterion // partition the tuples and grow subtrees for each partition
- ✓ let Dj be the set of data tuples in D satisfying outcome j; // a partition
- ✓ if Dj is empty then
- ✓ attach a leaf labelled with the majority class in D to node N
- ✓ else attach the node returned by Generate decision tree(Dj, attribute list) to node N endfor
- ✓ return N

1) Entropy

- The Formula for entropy is:

$$Entropy(S) = -\left(\frac{p}{p+n}\right) * \log_2\left(\frac{p}{p+n}\right) - \left(\frac{n}{p+n}\right) * \log_2\left(\frac{n}{p+n}\right)$$

- ✓ Where p is the positive samples
- ✓ Where n is the negative samples
- ✓ Where S is the total sample

- The Entropy from the Table 1 is:

$$\textbf{Entropy}\ (\mathbf{S}) = -\left(\frac{9}{14}\right)\log_2\left(\frac{9}{14}\right) - \left(\frac{5}{14}\right)\log_2\left(\frac{5}{14}\right) = 0.940$$

Table 1. Class Labelled Training Tuples From All Electronics Customer Database

RID	Age	Income	Student	Credit_rating	Class:buys_computer
1	Youth	High	No	Fair	No
2	Youth	High	No	Excellent	No
3	Middle_Aged	High	No	Fair	Yes
4	Senior	Medium	No	Fair	Yes
5	Senior	Low	Yes	Fair	Yes
6	Senior	Low	Yes	Excellent	No
7	Middle_Aged	Low	Yes	Excellent	Yes
8	Youth	Medium	No	Fair	No
9	Youth	Low	Yes	Fair	Yes
10	Senior	Medium	Yes	Fair	Yes
11	Youth	Medium	Yes	Excellent	Yes
12	Middle_Aged	Medium	No	Excellent	Yes
13	Middle_Aged	High	Yes	Fair	Yes
14	Senior	Medium	No	Excellent	No

2) Information Gain

ID3 uses information gain as its attribute selection measure. This measure is based on pioneering work by Claude Shannon on information theory, which studied the value or "information content" of messages. Let node N represents or hold the tuples of partition D. The attribute with the highest information gain is chosen as the splitting attribute for node N. This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or "impurity" in these partitions. Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple (but not necessarily the simplest) tree is found. The expected information needed to classify a tuple in D is given by

$$Info(D) = -\sum_{i=1}^{m} Pi \log_2(Pi)$$

Where p_i is the probability that an arbitrary tuple in D belongs to class C_i and is estimated by $jC_i,D_j/jD_j$. A log function to the base 2 is used, because the information is encoded in bits. Info(D) is just the average amount of information needed to identify the class label of a tuple in D. Note that, at this point, the information we have is based solely on the proportions of tuples of each class. Info(D) is also known as the entropy of D.

Now, suppose we were to partition the tuples in D on some attribute A having v distinct values, $fa_1, a_2 \ldots a_v$, as observed from the training data. If A is discrete-valued, these values correspond directly to the v outcomes of a test on A. Attribute A can be used to split D into v partitions or subsets, $fD_1, D_2, : : : , D_v$, where D_j contains those tuples in D that have outcome a_j of A. These partitions would correspond to the branches grown from node N. Ideally, we would like this partitioning to produce an exact classification of the tuples. That is, we would like for each partition to be pure. However, it is quite likely that the partitions will be impure (e.g., where a partition may contain a collection of tuples from different classes rather than from a single class). How much more information would we still need (after the partitioning) in order to arrive at an exact classification? This amount is measured by

$$Info_A(D) = \sum_{j=1}^{y} \frac{|D_j|}{|D|} * Info(D_j)$$

Information gain is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the new requirement (i.e., obtained after partitioning on A). That is,

$$Gain(A) = Info(D) - Info_A(D)$$

In other words, Gain(A) tells us how much would be gained by branching on A. It is the expected reduction in

the information requirement caused by knowing the value of A. The attribute A with the highest information gain, (Gain(A)), is chosen as the splitting attribute at node N. This is equivalent to saying that we want to partition on the attribute A that would do the "best classification," so that the amount of information still required to finish classifying the tuples is minimal (i.e., minimum $Info_A(D)$).

- Calculating the information gain for each of the attributes:
  - ✓ For Age
  - ✓ For Income
  - ✓ For Student
  - ✓ For Credit_rating

For Age

- E = 0.940
- S = (9+, 5-)

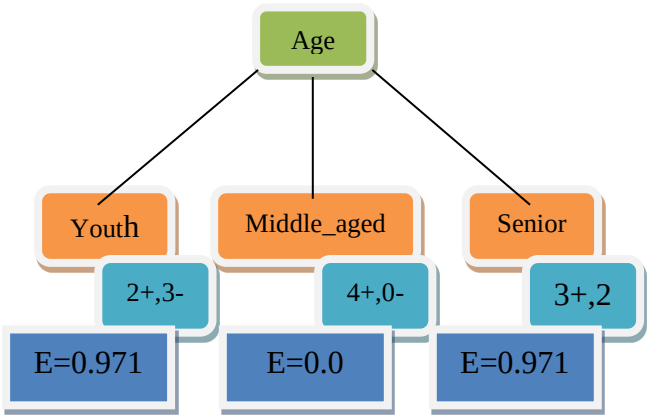


Figure 2. Decision tree construction with the root age

$$Gain(S, Age) = 0.940 - \left(\frac{5}{14}\right) \times 0.971 - \left(\frac{4}{14}\right) \times 0.0 - \left(\frac{5}{14}\right) \times 0.0971 = 0.247$$

For Student

- E = 0.940
- S = (9+, 5-)

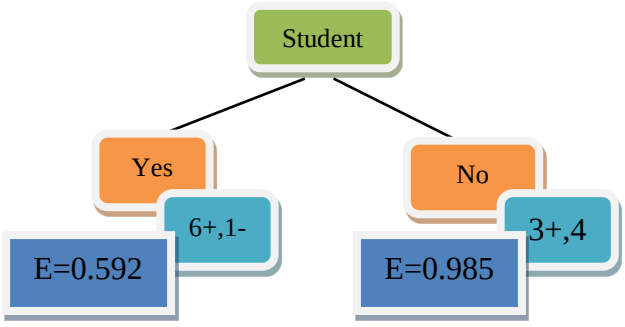


Figure 3. Entropy calculation of the attribute student

$$Gain(S, Student) = 0.940 - \left(\frac{7}{14}\right) \times 0.592 - \left(\frac{7}{14}\right) \times 0.0985 = 0.151$$

For Credit_rating

- E = 0.940
- S = (9+, 5-)

$$Gain(S, Credit\_rating) = 0.940 - \left(\frac{8}{14}\right) \times 0.811 - \left(\frac{6}{14}\right) \times 1.0 = 0.048$$

Now, selecting the root from all information gain:

- Compute information gain for each attribute:
  Gain (credit_rating) = 0.048
  Gain (student) = 0.151
  Gain (income) = 0.029
  Gain (age) = 0.246
- Select attribute with the maximum information gain, which is 'age' for splitting.

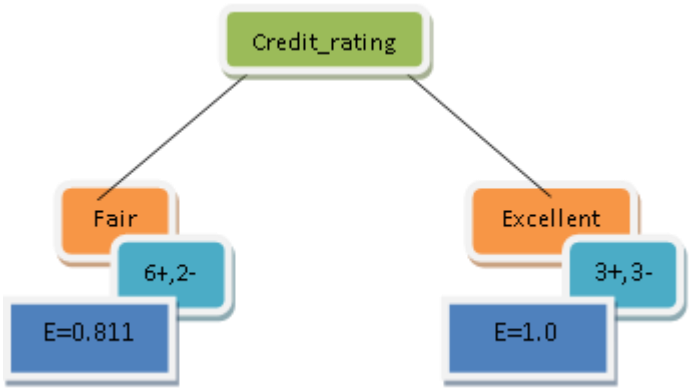
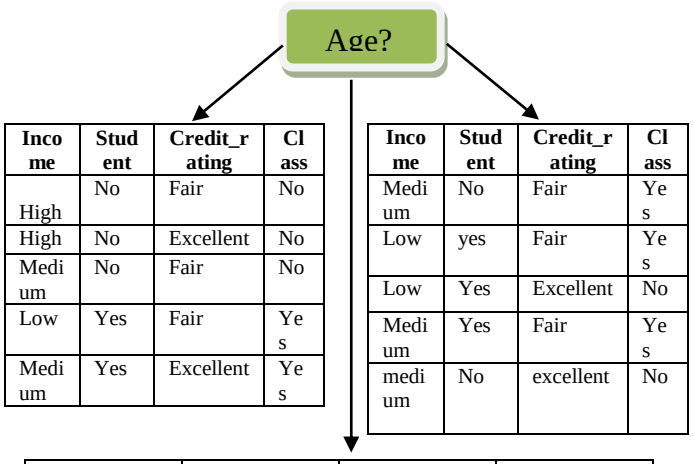


Figure 4. Entropy calculation of the attribute credit rating



Income	Student	Credit_rating	Class
High	No	Fair	No
High	No	Excellent	No
Medium	No	Fair	No
Low	Yes	Fair	Yes
Medium	Yes	Excellent	Yes

Income	Student	Credit_rating	Class
Medium	No	Fair	Yes
Low	yes	Fair	Yes
Low	Yes	Excellent	No
Medium	Yes	Fair	Yes
medium	No	excellent	No

Income	Student	Credit_rating	Class
High	No	Fair	Yes
Low	Yes	Excellent	Yes
Medium	No	Excellent	Yes
High	Yes	Fair	Yes

Figure 5. The attribute age has the highest information gain and therefore becomes the splitting attribute at the root node of the decision tree. Branches are grown for each outcome of age. The tuples are shown partitioned accordingly.

"But how can we compute the information gain of an attribute that is continuous-valued, unlike above?" Suppose, instead, that we have an attribute A that is continuous-valued, rather than discrete-valued. (For example, suppose that instead of the discretized version of age above, we instead have the raw values for this attribute.) For such a scenario, we must determine the "best" split-point for A, where the split-point is a threshold on A.

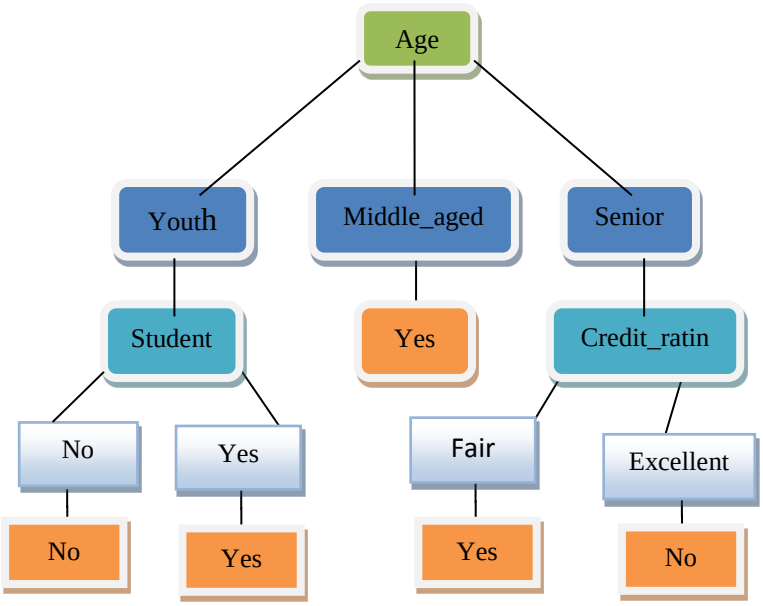


Figure 6. The Complete Decision Tree

We first sort the values of A in increasing order. Typically, the midpoint between each pair of adjacent values is considered as a possible split-point. Therefore, given v values of A, then v+1 possible splits are evaluated. For example, the midpoint between the values ai and ai+1 of A is:

$$\frac{a_i + a_i + 1}{2}$$

If the values of A are sorted in advance, then determining the best split for A requires only one pass through the values. For each possible split-point for A, we evaluate InfoA(D), where the number of partitions is two, that is v = 2 (or j = 1,2). The point with the minimum expected information requirement for A is selected as the split point for A. D1 is the set of tuples in D satisfying A = split point, and D2 is the set of tuples in D satisfying A >split point.

3) Gain Ratio

The information gain measure is biased toward tests with many outcomes. That is, it prefers to select attributes having a large number of values. For example, consider an attribute that acts as a unique identifier, such as product ID. A split on product ID would result in a large number of partitions (as many as there are values), each one containing just one tuple. Because each partition is pure, the information required to classify data set D based on this partitioning would be$Info_{product}ID(D) = 0$. Therefore, the information gained by partitioning on this attribute is maximal. Clearly, such a partitioning is useless for classification.

C4.5, a successor of ID3, uses an extension to information gain known as gain ratio, which attempts to overcome this bias. It applies a kind of normalization to information gain using a "split information" value defined analogously with Info(D) as:

$$SplitInfo_A(D) = -\sum_{j=1}^{y} \frac{|D_j|}{|D|} \log_2\left(\frac{D_j}{D}\right)$$

This value represents the potential information generated by splitting the training data set, *D*, into *v* partitions, corresponding to the *v* outcomes of a test on attribute *A*. Note that, for each outcome, it considers the number of tuples having that outcome with respect to the total number of tuples in *D*. It differs from information gain, which measures the information with respect to classification that is acquired based on the same partitioning. The gain ratio is defined as:

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}$$

The attribute with the maximum gain ratio is selected as the splitting attribute. Note, however, that as the split information approaches 0, the ratio becomes unstable. A constraint is added to avoid this, whereby the information gain of the test selected must be large—at least as great as the average gain over all tests examined.

4) Decision Tree Induction Algorithm

- Read the training data set D from a text file or database
- Calculate the number of positive answers (p) & the negative answers (n) in the class label of data set D
- Calculate the expected information needed to classify a tuple in D [formula: -(p/(p+n))*$\log_2$(p/(p+n))*- (n/(p+n))*)$\log_2$ (n/(p+n)]
- Calculate the expected information requirement for the 1st attribute.
- Calculate the gain in information for the 1st attribute: Gain(attribute_name)= Info(D) – $Info_{attribute\_name}(D)$
- Recursively do the steps 4 & 5 for all the other attributes
- Select the root attribute as the one carrying the maximum gain from all the attributes except the class label
- If any node of the root ends in leaves then put label
- If any node of the root does not end in leaves than go to step 6
- End

5) Implementation of the Algorithm

Begin

Load training sets first, create decision tree root node 'rootNode', add training set D into root node as its subset. For rootNode, we compute Entropy (rootNode.subset) first

$$If\ Entropy\ (rootNode.subset)\ == 0, \text{ then}$$

rootNode.subset consists of records all with the same value for the categorical attribute, return a leaf node with decision attribute:attribute value;

$$If\ Entropy\ (rootNode.subset)!\ = 0, \text{ then}$$

compute information gain for each attribute left (that have not been used in splitting), find attribute A with Maximum Gain(S,A)[where S = Sample and A= Attributes].Create child nodes of this rootNode and add to rootNode in the decision tree. For each child of the rootNode, apply the algorithm from the beginning until node that has entropy=0 or leaf node is reached.
End Algorithm.

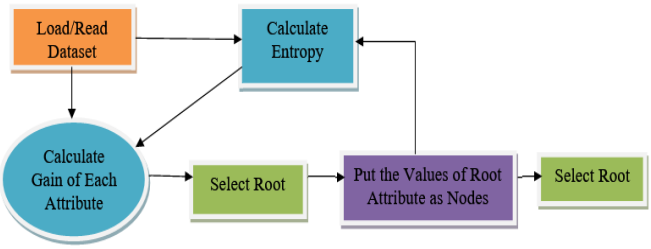


Figure 7. Implementation of the Algorithm

6) Description of Implementation

To describe the operation of decision tree induction algorithm, we use a classic 'buys_computer' example.

The symbolic attribute description:

Attribute	Possible values
Age	Youth, middle_aged, senior
Income	High, medium, low
Student	Yes, no
Credit_rating	Fair, excellent
Buys	Yes, no

After implementing the above algorithm using visual studio 2010 & XAMPP we implemented the algorithm:

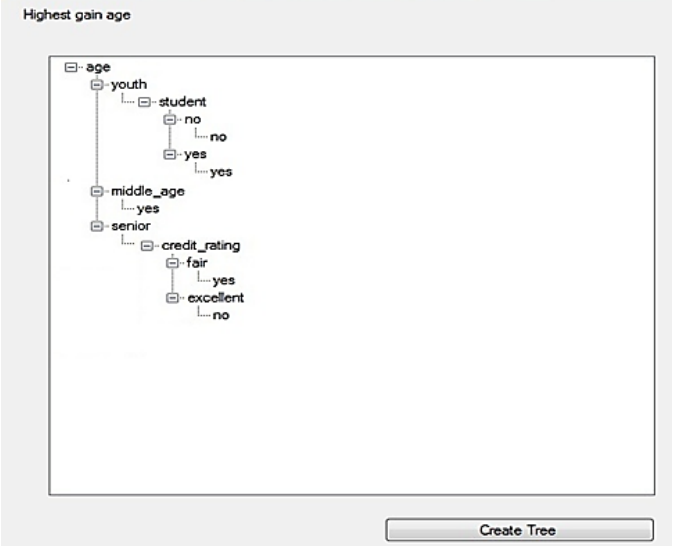


Figure 8. Snapshot of the implemented algorithm

7) Extending Decision Tree Induction Algorithm to real-valued data:

Decision tree induction algorithm is quite efficient in dealing with the target function that has discrete output values. It can easily deal with instance which is assigned to a Boolean decision, such as 'true' and 'false', 'yes (positive)' and 'no (negative)'. It is possible to extend target to real-valued outputs. When we compute information gain in our CODE, we tried to make is as

much dynamic as possible. So we can calculate both discrete values and continuous values by our program. You will see the result as below:

id	age	parents	poverty	Health	Lives	Goes
1	Below 5	yes	Above	Good	Metro pl	Yes
2	Below 5	yes	Below	Bad	Village	No
3	Above 5 Below 10	yes	Above	Avrg	Town	Yes
4	Above 10	no	Above	Avrg	Town	Yes
5	Above 10	yes	Below	Bad	Town	No
6	Above 5 Below 10	no	Above	Bad	Village	Yes
7	Above 5 Below 10	no	Above	Bad	Metro pl	Yes
8	Below 5	yes	Below	Good	Town	No
9	Below 5	no	Below	Bad	Town	No
10	Above 10	yes	Above	Bad	Village	Yes
11	Above 10	no	Below	Good	Metro pl	No
12	Below 5	yes	Above	Avrg	Metro pl	Yes
13	Above 5 Below 10	yes	Below	Good	Village	No
14	Above 5 Below 10	no	Below	Avrg	Town	No
15	Above 10	no	Above	Bad	Metro pl	Yes
16	Below 5	yes	Above	Good	Metro pl	No
17	Above 5 Below 10	yes	Below	Avrg	Village	Yes
18	Above 10	no	Below	Avrg	Metro pl	Yes
19	Below 5	yes	Above	Good	Metro pl	No
20	Above 5 Below 10	yes	Above	Good	Town	Yes
21	Above 10	yes	Below	Good	Town	Yes

Figure 9. Real Valued Dataset

From figure 9, we first calculate the total entropy:

$$\boldsymbol{Entropy}\ (\boldsymbol{S}) = -\left(\frac{12}{21}\right) \times \log_2 \frac{12}{21} - \left(\frac{9}{21}\right) \times \log_2 \frac{9}{21} = 0.985$$

Now calculating information gain for each of the attributes:

- ✓ Age
- ✓ Parents
- ✓ Poverty
- ✓ Health
- ✓ Lives

❖ Age:

- E = 0.985
- S = (12+,9-)

$$\boldsymbol{Gain}(\boldsymbol{S}, \boldsymbol{Age}) = 0.985 - \left(\frac{7}{21}\right) \times 0.863 - \left(\frac{7}{21}\right) \times 0.863 - \left(\frac{7}{21}\right) \times 0.863 = 0.275$$

❖ Parents:

- E = 0.985
- S = (12+, 9-)

$$\boldsymbol{Gain}(\boldsymbol{S}, \boldsymbol{Parents}) = 0.985 - \left(\frac{21}{21}\right) \times 0.958 - \left(\frac{21}{21}\right) \times 0.958 = 0.167$$

Similarly computing the gain of poverty, health & lives we have:

Gain(S, Poverty) = 0.223
Gain(S, Health) = 0.075
Gain(S, Lives) = 0.157

Now, we select attribute with the maximum information gain, which is 'age' for splitting.

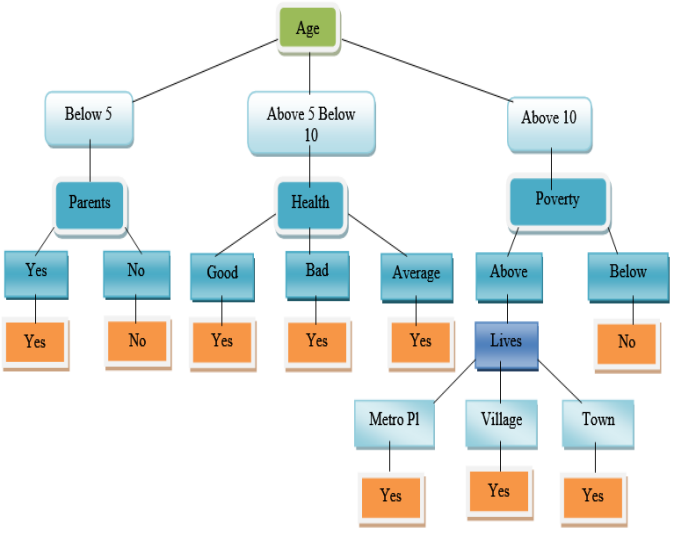


Figure 10. Final Decision Tree of Real Valued Dataset

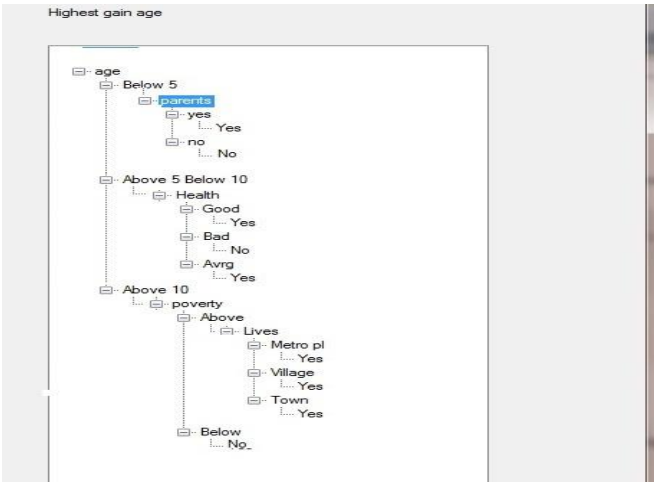


Figure 11. Decision Tree Generated from the Dataset After Implementation.

B. Predictive Algorithm-2 (Naïve Bayesian Classification Algorithm)

It is based on the Bayesian theorem it is particularly suited when the dimensionality of the inputs is high. Parameter estimation for naive Bayes models uses the method of maximum likelihood. In spite over-simplified

assumptions, it often performs better in many complex real-world situations.

The naive Bayesian classifier, or simple Bayesian classifier, works as follows:

1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, X = $(x_1, x_2, \ldots\ldots, x_n)$ depicting n measurements made on the tuple from n attributes, respectively,$A_1, A_2, \ldots\ldots, A_n$

2. Suppose that there are m classes, $C_1, C_2, \ldots\ldots, C_m$. Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X. That is, the naive Bayesian classifier predicts that tuple X belongs to the classC_i; if and only if

$P(C_i|X) > P(C_j|X)$ for $1 \leq j \leq m, j \neq i$.

Thus we maximize $(C_i|X)$. The class C_i for which $P(C_i|X)$is maximized is called the maximum posteriori hypothesis. By Bayes' theorem,

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$$

3. As P(X) is constant for all classes, only $P(X|C_i)P(C_i)$ need be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is, $P(C_1) = P(C_2) = \cdots = P(C_m)$, and we would therefore maximize$P(X|C_i)$. Otherwise, we maximize$P(X|C_i)P(C_i)$. Note that the class prior probabilities maybe estimated by $P(C_i) = |C_{i,d}|/|D|$ where $|C_{i,d}|$is the number of training tuples of class $C_i$in$D$.

4. Given data sets with many attributes, it would be extremely computationally expensive to compute $P(X|C_i)$ .In order to reduce computation in evaluating $P(X|C_i)$, the naive assumption of class conditional independence is made. This presumes that the values of the attributes are conditionally independent of one another, given the class label of the tuple (i.e., that there are no dependence relationships among the attributes). Thus,

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$
$$= P(x_1|C_i) * P(x_2|C_i) * \ldots * P(x_n|C_i)$$

We can easily estimate the probabilities $P(x_1|C_i) * P(x_2|C_i) * \ldots * P(x_n|C_i)$ from the training tuples. Recall that here x_k refers to the value of attribute A_k for tuple X. For each attribute, we look at whether the attribute is categorical or continuous-valued. For instance, to compute $P(X|C_i)$ we consider the following:

(a) If $A_k$is categorical, then $P(x_k|C_i)$is the number of tuples of class C_i in D having the value x_k for A_k, divided by$|C_{i,d}|$, the number of tuples of classC_i in D.
(b) If A_k is continuous-valued, then we need to do a bit more work, but the calculation is pretty straightforward. A continuous-valued attribute is typically assumed to have a Gaussian distribution with a mean μ and standard deviation σ, defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

So that,

$$P(x_k|C_i) = g(x_k, \mu_{c_i}, \sigma_{c_i})$$

These equations may appear daunting, but hold on! We need to compute$\mu_{c_i}$and σ_{c_i} which are the mean (i.e., average) and standard deviation, respectively, of the values of attribute A_k for training tuples of class C_i . We then plug these two quantities into Equation, together with x_k, in order to estimate $P(x_k|C_i)$for example, let X = (35, $40,000), where $A_1$and A_2 are the attributes age and income, respectively. Let the class label attribute be buys -computer. The associated class label for X is yes (i.e., buys_computer = yes). Let's suppose that age has not been discredited and therefore exists as a continuous-valued attribute. Suppose that from the training set, we find that customers in D who buy a computer are 38 ± 12 years of age. In other words, for attribute age and this class, we have μ = 38 years and σ = 12. We can plug these quantities, along with x_1= 35 for our tuple X into Equation in order to estimate P (age = 35|buys_computer = yes).

5. In order to predict the class label of X, $P(X|C_i)P(C_i)$ is evaluated for each class C_i. The classifier predicts that the class label of tuple X is the class C, if and only if

$P(X|C_i)P(C_i) > P(X|C_j)P(C_j)$ for $1 \leq j \leq m, j \neq i$.

In other words, the predicted class label is the class C, for which $P(X|C_i)P(C_i)$ is the maximum.

1) Applying Bayesian Classification on the Data Set-Algorithm

1. Select any tuple 'X' from the dataset.
2. Find the number of P's and N's from the class label where P=Positive Answer and N=Negative Answer.
3. Calculate the probability of P and N where ***P(P) = P/(P+N)*** and ***P(N) = N/(P+N)***
4. Find the probability of all the attributes from the tuple 'X' for P and N where
5. ***P (attribute | class label = P);***
6. ***P (attribute | class label = N);***
7. Multiply all the P (attribute | class label = P)'s as X1 and multiply all the P (attribute | class label = N)'s as X2.
8. Calculate ***R = P (P) * X1*** and ***S = P (N) * X2.***
9. If R>S then answer is P (Positive Answer).
10. Otherwise answer is N (Negative Answer).

2) Implementation of Naïve Bayesian Classification

Begin

Load training datasets first. Calculate the probability of class labels containing the positive answer and the negative answer. Take the values of the attributes from the user as input; we assume that is Tuple X.
Calculate the probability of all the attributes from the tuple X-

Probability of (attribute | class label = Positive Class Label);
Probability of (attribute | class label = Negative Class Label);
Multiply the probability of all the positive answer and negative answer.
If **Probability of (Positive Class Label) >Probability of (Negative Class Label);**
Then the answer is Positive Answer. Otherwise the answer is Negative Answer.

3) Description of Implementation

The implementation of the proposed algorithm can be depicted as follows:

id	age	income	student	credit_rating	Buys
16	youth	high	no	fair	no
17	youth	high	no	excellent	no
18	middle_age	high	no	fair	yes
19	senior	medium	no	fair	yes
20	senior	low	yes	fair	yes
21	senior	low	yes	excellent	no
22	middle_age	low	yes	excellent	yes
23	youth	medium	no	fair	no
24	youth	low	yes	fair	yes
25	senior	medium	yes	fair	yes
26	youth	medium	yes	excellent	yes
27	middle_age	medium	no	excellent	yes
28	middle_age	high	yes	fair	yes
29	senior	medium	no	excellent	no

Figure 12. Training dataset

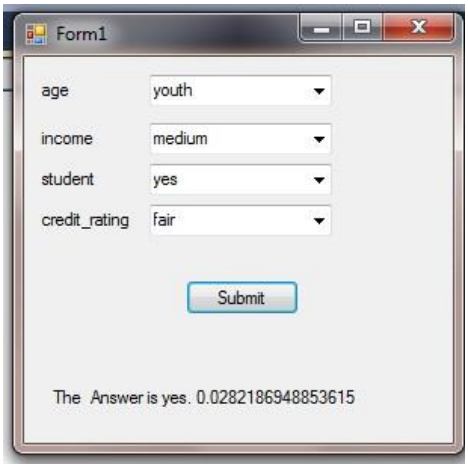


Figure 13. Interface of the program implementing naïve Bayesian algorithm

Our Real Valued Dataset:
The real valued dataset for this paper is constructed as follows in the figure.

id	age	parents	poverty	Health	Lives	Goes
1	Below 5	yes	Above	Good	Metro pl	Yes
2	Below 5	yes	Below	Bad	Village	No
3	Above 5 Below 10	yes	Above	Avrg	Town	Yes
4	Above 10	no	Above	Avrg	Town	Yes
5	Above 10	yes	Below	Bad	Town	No
6	Above 5 Below 10	no	Above	Bad	Village	Yes
7	Above 5 Below 10	no	Above	Bad	Metro pl	Yes
8	Below 5	yes	Below	Good	Town	No
9	Below 5	no	Below	Bad	Town	No
10	Above 10	yes	Above	Bad	Village	Yes
11	Above 10	no	Below	Good	Metro pl	No
12	Below 5	yes	Above	Avrg	Metro pl	Yes
13	Above 5 Below 10	yes	Below	Good	Village	No
14	Above 5 Below 10	no	Below	Avrg	Town	No
15	Above 10	no	Above	Bad	Metro pl	Yes
16	Below 5	yes	Above	Good	Metro pl	No
17	Above 5 Below 10	yes	Below	Avrg	Village	Yes
18	Above 10	no	Below	Avrg	Metro pl	Yes
19	Below 5	yes	Above	Good	Metro pl	No
20	Above 5 Below 10	yes	Above	Good	Town	Yes
21	Above 10	yes	Below	Good	Town	Yes

Figure 14. Real valued dataset

The Results of our Real Valued Dataset:

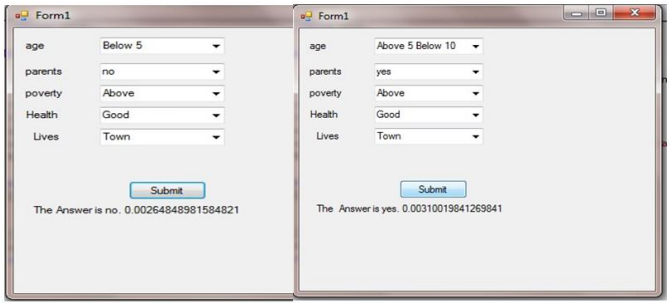
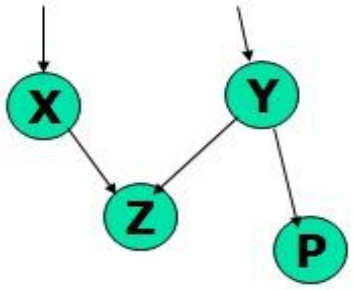


Figure 15. Program running with real valued data

4) Bayesian Networks

- Bayesian belief network allows a *subset* of the variables conditionally independent
- A graphical model of causal relationships
  - Represents dependency among the variables
  - Gives a specification of joint probability distribution



- Nodes: random variables
- Links: dependency
- X,Y are the parents of Z, and Y is the parent of P
- No dependency between Z and P
- Has no loops or cycles

5) Bayesian Belief Network: An Example

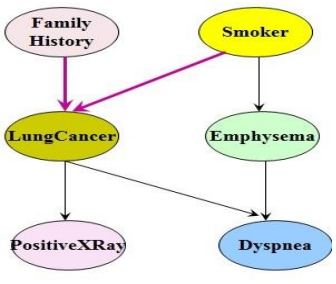


Figure 16. Bayesian Belief Networks

TABLE 2. THE CONDITIONAL PROBABILITY TABLE FOR THE VARIABLE LUNG CANCER

	(FH, S)	(FH, ~S)	(~FH, S)	(~FH, ~S)
LC	0.8	0.5	0.7	0.1
~LC	0.2	0.5	0.3	0.9

The table shows the conditional probability for each possible combination of its parents:

$$P(z1,\ldots,zn) = \prod_{i=1}^{n} P(z_i \mid Parents(Z_i))$$

III. PREDICTION AND ACCURACY

A. Prediction

Regression analysis is a good choice when all of the predictor variables are continuous valued as well. Many problems can be solved by *linear regression*, and even more can be tackled by applying transformations to the variables so that a nonlinear problem can be converted to a linear one. For reasons of space, we cannot give a fully detailed treatment of regression. Several software packages exist to solve regression problems. Examples include SAS(*www.sas.com*), SPSS (*www.spss.com*), and S-Plus (*www.insightful.com*). Another useful resource is the book *Numerical Recipes in C*, by Press, Flannery, Teukolsky, and Vetterling, and its associated source code.

B. Accuracy and error measures

Now that a classifier or predictor has been trained, there may be several questions. For example, suppose one used data from previous sales to train a classifier to predict customer purchasing behaviour. Then an estimate of how accurately the classifier can predict the purchasing behaviour of future customers, that is, future customer data on which the classifier has not been trained can be done. One may even have tried different methods to build more than one classifier (or predictor) and now wish to compare their accuracy. There are various strategies to estimate and increase the accuracy of a learned model. These issues are addressed next.

TABLE 3. CONFUSION MATRIX FOR THE EXAMPLE OF ALL ELECTRONICS

Clsses	buys_computer = yes	Buys_computer = no	Total	Recognition %
buys_computer = yes	6954	46	7000	99.34
buys_computer = no	412	2588	3000	86.27
Total	7366	2634	10000	95.52

C. Classifier Accuracy Measures

Using training data to derive a classifier or predictor and then to estimate the accuracy of the resulting learned model can result in misleading overoptimistic estimates due to overspecialization of the learning algorithm to the data. Instead, accuracy is better measured on a test set consisting of class-labelled tuples that were not used to train the model. The accuracy of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier. In the pattern recognition literature, this is also referred to as the overall recognition rate of the classifier, that is, it reflects how well the classifier recognizes tuples of the various classes. We can also speak of the error rate or misclassification rate of a classifier, *M*, which is simply *Acc*(*M*), where *Acc*(*M*) is the accuracy of *M*. If we were to use the training set to estimate the error rate of a model, this quantity is known as the re substitution error. This error estimate is optimistic of the true error rate (and similarly, the corresponding accuracy estimate is optimistic) because the model is not tested on any samples that it has not already seen.

The *confusion matrix* is a useful tool for analyzing how well your classifier can recognize tuples of different classes. A confusion matrix for two classes is shown in Table 3. Given *m* classes, a confusion matrix is a table of at least size *m* by *m*. An entry, *CMi*, *j* in the first *m* rows and *m* columns indicates the number of tuples of class *i* that were labelled by the classifier as class *j*. For a classifier to have good accuracy, ideally most of the tuples would be represented along the diagonal of the confusion matrix, from entry *CM*1, 1 to entry *CMm*, *m*, with the rest of the entries being close to zero. The table may have additional rows or columns to provide totals or recognition rates per class. Given two classes, we can talk in terms of positive tuples (tuples of the main class of interest, e.g., *buys computer = yes*) versus negative tuples (e.g., *buys computer = no*). True positives refer to the positive tuples that were correctly labelled by the classifier, while true negatives are the negative tuples that were correctly labelled by the classifier. False positives are the negative tuples that were incorrectly labelled (e.g., tuples of class *buys computer = no* for which the classifier predicted *buys computer = yes*).

TABLE 4. A CONFUSION MATRIX FOR POSITIVE AND NEGATIVE TUPLES

Predicted class

Actual Class

	C1	C2
C1	True positive	True negative
C2	False positive	False negative

Similarly, false negatives are the positive tuples that were incorrectly labelled (e.g., tuples of class *buys computer = yes* for which the classifier predicted *buys computer = no*). These terms are useful when analyzing a classifier's ability. *"Are there alternatives to the accuracy measure?"* Suppose that you have trained a classifier to classify medical data tuples as either *"cancer"* or *"not cancer."* An accuracy rate of, 90% may make the classifier seem quite accurate, but what if only, say, 3–4% of the training tuples are actually *"cancer"*? Clearly, an accuracy rate of 90% may not be acceptable—the classifier could be correctly labelling only the *"not cancer"* tuples, for instance. Instead, we would like to be able to access how well the classifier can recognize *"cancer"* tuples (the positive tuples) and how well it can recognize *"not cancer"* tuples (the negative tuples). The sensitivity and specificity measures can be used, respectively, for this purpose. Sensitivity is also referred to as the *true positive (recognition) rate* (that is, the proportion of positive tuples that are correctly identified), while specificity is the *true negative rate* (that is, the proportion of negative tuples that are correctly identified). In addition, we may use precision to access the percentage of tuples labelled as *"cancer"* that actually are *"cancer"* tuples. These measures are defined as

$$Sensitivity = t\_pos / pos$$
$$Specificity = t\_neg / neg$$
$$Precision = t\_pos / (t\_pos + f\_pos)$$

where *t pos* is the number of true positives (*"cancer"* tuples that were correctly classified as such), *pos* is the number of positive (*"cancer"*) tuples, *t neg* is the number of true negatives (*"not cancer"* tuples that were correctly classified as such), *neg* is the number of negative (*"not cancer"*) tuples, and *f pos* is the number of false positives (*"not cancer"* tuples that were incorrectly labelled as *"cancer"*). It can be shown that accuracy is a function of sensitivity and specificity:

$$accuracy = sensitivity \frac{pos}{(pos + neg)} + specificity \frac{neg}{pos + neg}$$

The true positives, true negatives, false positives, and false negatives are also useful in assessing the costs and benefits (or risks and gains) associated with a classification model. The cost associated with a false negative (such as, incorrectly predicting that a cancerous patient is not cancerous) is far greater than that of a false positive (incorrectly yet conservatively labelling a noncancerous patient as cancerous). In such cases, we can outweigh one type of error over another by assigning a different cost to each. These costs may consider the danger to the patient, financial costs of resulting therapies, and other hospital costs. Similarly, the benefits associated with a true positive decision may be different than that of a true negative. Up to now, to compute classifier accuracy, we have assumed equal costs and essentially divided the sum of true positives and true negatives by the total number of test tuples. Alternatively, we can incorporate costs and benefits by instead computing the average cost (or benefit) per decision. Other applications involving cost-benefit analysis include loan application decisions and target marketing mail outs. For example, the cost of loaning to a defaulter greatly exceeds that of the lost business incurred by denying a loan to a non-defaulter. Similarly, in an application that tries to identify households that are likely to respond to mail outs of certain promotional material, the cost of mail outs to numerous households that do not respond may outweigh the cost of lost business from not mailing to households that would have responded. Other costs to consider in the overall analysis include the costs to collect the data and to develop the classification tool.

"Are there other cases where accuracy may not be appropriate?" In classification problems, it is commonly assumed that all tuples are uniquely classifiable, that is, that each training tuple can belong to only one class. Yet, owing to the wide diversity of data in large databases, it is not always reasonable to assume that all tuples are uniquely classifiable. Rather, it is more probable to assume that each tuple may belong to more than one class. How then can the accuracy of classifiers on large databases be measured? The accuracy measure is not appropriate, because it does not take into account the possibility of tuples belonging to more than one class. Rather than returning a class label, it is useful to return a probability class distribution. Accuracy measures may then use a second guess heuristic, whereby a class prediction is judged as correct if it agrees with the first or second most probable class. Although this does take into consideration, to some degree, the non-unique classification of tuples, it is not a complete solution.

IV. CONCLUSION

This paper is representing predictive algorithms regarding data mining. We have implemented those algorithms in our software's. Further we used naïve Bayesian algorithm for prediction. Our software can predict successfully. Finally we worked on the accuracy of our implemented software. So far the accuracy level is moderate. In future we will compare our software with other predictive algorithms (such as: neural networks etc.) so check how well is our algorithms and code is working. And we will work on increasing the accuracy of prediction.

V. ACKNOWLEDGEMENT

First of all we acknowledge to Almighty for completing this research successfully. Then we are grateful to all of the authors individually. We would like to thank our parents, friends for their invaluable suggestions and critical review of our thesis.

VI. REFERENCES

[1] Decision Tree Algorithms: Integration of Domain Knowledge for Data Mining, Aukse Stravinskiene, Saulius Gudas, and Aiste Davrilaite; 2012
[2] S.B. Kotsiantis, Supervised Machine Learning: A Review of Classification Techniques, Informatica 31(2007) 249-268, 2007
[3] K. Karimi and H.J. Hamilton, Logical Decision Rules: Teaching C4.5 to Speak Prolog, IDEAL, 2000
[4] Rennie, J.; Shih, L.; Teevan, J.; Karger, D. (2003). "Tackling the poor assumptions of Naive Bayes classifiers"
[5] John, George H.; Langley, Pat (1995). "Estimating Continuous Distributions in Bayesian Classifiers". Proc. Eleventh Conf. on Uncertainty in Artificial Intelligence. Morgan Kaufmann. pp. 338–345
[6] D. Pregibon, "Data Mining", *Statistical Computing and Graphics*,vol. 7, no. 3, p. 8, 1996.
[7] U. Fayyad, *Advances in knowledge discovery and data mining*. Menlo Park, Calif.: AAAI, 1996.
[8] R. Kimball, "The Data Webhouse Toolkit: Building the Web-enabled Data Warehouse20001 The Data Webhouse Toolkit: Building the Web-enabled Data Warehouse. John Wiley & Son,, ISBN: 0-471-37680-9 £32.50 Paperback", *Industr Mngmnt & Data Systems*, vol. 100, no. 8, pp. 406-408, 2000.
[9] M. Betts, "The Almanac:Hot Tech", *Computerworld*, 2003. [Online]. Available: http://www.computerworld.com/article/2574084/data-center/the-almanac.html