Partial Match Queries Using Error Correcting Code Signatures

D.B. Skillicorn
skill@qucis.queensu.ca

November 1998
External Technical Report
ISSN-0836-0227-
1998-419

Department of Computing and Information Science
Queen's University
Kingston, Ontario, Canada K7L 3N6

Document prepared November 6, 1998

Partial Match Queries Using Error Correcting Code Signatures

Abstract

An important component of some data mining algorithms is determining the frequency of some set of attributes in an extremely large dataset. This requires partial match queries, in which some attributes have "don't care" values. We present a partial match query algorithm that uses the codewords of error-correcting codes as signatures.

1 Introduction

Data mining algorithms aim to extract concepts (for example, predicates) from data collected by enterprises, often originally for another purpose. Although such algorithms are computationally intensive, their performance tends to be dominated by the time required to access the data. Early data mining algorithms often assumed a database interface to the data, but it is now more common to assume that the data is extracted and stored in some flat format before algorithms are executed on it.

We will assume that the dataset is a table of n rows and m columns containing a single bit in each position. Each row corresponds to a 'customer' and each column to an 'attribute'. The bits then describe whether or not each customer has each attribute. For example, the rows may be the items purchased in a single visit to a supermarket.

A typical operation involving the data might be to determine how often a particular tuple of attributes occur together in a row. This requires examining every row of the table. In practice, the value of n is extremely large, perhaps 10^9, so direct lookups are expensive.

Such queries are *partial match* queries, since they can be expressed in the form - - 1 - - 1 - - where - denotes "don't care". We will treat only the case where the explicit values in the query are 1's, that is, we are interested only in the presence of an attribute in a row, and not in its absence. This form of query arises in the Frequent Set problem [6], an important component of many data mining algorithms, for example, association rules.

A partial match query is a special form of point access query. It is natural to represent each row as a point in m-dimensional space. However, standard point access methods do not handle partial match queries well. In data mining applications, m is large, perhaps 10^3, and the extent in each dimension is small (1 here); whereas most point access methods assume that the number of dimensions is small and the extents large. Thus grid files [5], and R-trees [4] and their extensions, are unlikely to perform well because the dimensionality is high. Techniques such as k-D trees [1] partition points based on attributes in some particular order. They will not work well if the attributes appearing first in this ordering are "don't care"s in the query.

Another approach to point queries is to map them to spatial queries and use a spatial access method. Space-filling curves [2] replace an m-dimensional space by a linear one. Unless there is some underlying geometric property determining the order in which the curve visits dimensions, a region may become a long sequence of blocks in the linear ordering. A partial match query

is a particular kind of region, a quadrant, and so suffers from this effect. There is no natural *a priori* ordering of the dimensions in data mining applications, since relationships between the dimensions, which represent attributes, are exactly what algorithms are trying to find. So partial match queries do have a spatial structure (they all select quadrants of $\mathbf{2}^m$) but they cannot assume a dimension ordering without potentially influencing their own results. This suggests that Hamming distance is an appropriate metric to use on $\mathbf{2}^m$.

Even though a typical data mining dataset is extremely large, it is still sparse in $\mathbf{2}^m$. Signature techniques, in which elements of the dataset are mapped to representatives against which queries can be rapidly checked, are a useful way to handle this sparsity. The use of Hamming distance suggests that the codewords of error-correcting codes would be appropriate signatures.

2 A method based on error-correcting codes

Choose a t-error-correcting code with N codewords of length m. For simplicity, we assume that the code is perfect, but that is not essential. These codewords will be signatures for rows in the dataset. As signatures, they are particularly appropriate because they are evenly distributed through $\mathbf{2}^m$ ($2t+1$ apart if the code is t-error-correcting).

The dataset is preprocessed by mapping each row of the table to its nearest codeword. Codewords maintain a list of those rows which have been mapped to them. Codewords to which no rows have been mapped may be discarded.

A partial match query consists of b positions set to 1 and $m-b$ positions that are "don't care"s. When a query is generated, its intersection with the set of codewords is computed. This reduces the problem to a partial match query on a particular fixed, well-structured dataset, the codewords. We describe the search technique used for this step in the next section.

The set of rows associated with the codewords selected may potentially match the query. A check on the appropriate bit positions returns that matching set.

We now examine the complexity of this search algorithm. The complexity of a direct search of the table is, of course, n.

Each codeword is at the centre of a sphere of radius t containing s points such that

$$N \leq 2^n/s$$

(equal if the code is perfect) and

$$s = \sum_{i=0}^{t} \left(\begin{array}{c} n \\ i \end{array} \right)$$

Thus the number of codewords whose spheres overlap a quadrant defined by b set bits is

$$\begin{array}{c} \dfrac{N}{2^b} \displaystyle\sum_{i=0}^{t} \left(\begin{array}{c} b \\ i \end{array} \right) \quad \text{if} \quad 2^b \leq N \\ \text{much smaller} \quad \text{if} \quad 2^b > N \end{array}$$

The term

$$\frac{N}{2^b} \left(\begin{array}{c} b \\ 0 \end{array} \right)$$

counts those spheres whose centres are in the quadrant, while terms of the form

$$\frac{N}{2^b} \left(\begin{array}{c} b \\ i \end{array} \right)$$

count spheres whose centres are distance i from the quadrant.

The complexity of a low-weight query is therefore

$$\begin{aligned} \text{Query complexity} &= \text{time to search codewords} + \\ &\qquad \text{number of codewords} \times \text{number of data points/sphere} \\ &= f(N) + \frac{N}{2^b} \sum_{i=0}^{t} \left(\begin{array}{c} b \\ i \end{array} \right) \frac{n}{N} \\ &= f(N) + \frac{n}{2^b} \left[1 + \sum_{i=1}^{t} \left(\begin{array}{c} b \\ i \end{array} \right) \right] \\ &\qquad \left\{ \sum_{i=1}^{t} \left(\begin{array}{c} b \\ i \end{array} \right) \leq \sum_{i=1}^{t} b^i = \frac{b^{t+1} - 1}{b - 1} - 1 \right\} \\ &\leq f(N) + \frac{n}{2^b} \left[\frac{b^{t+1} - 1}{b - 1} \right] \end{aligned}$$

where $f(N)$ is the cost of searching the codewords. The coefficient of n in the second term decreases rapidly as b increases, that is as the number of bits set in the query increases. Making t small makes this coefficient small too, but t is the radius of spheres around codewords, and making it small means that N must be increased. (Notice that we assume here that the table rows associated with each codeword are searched linearly, but small optimisations are surely possible here.)

3 Searching the codewords

It is clear that $f(N)$ is trivially no larger than N, and $N \ll n$. However, $f(N)$ may sometimes be as small as $\log N$ as we now show.

The problem has now been reduced to partial match queries over the codewords which have a special structure that we can exploit. We store the codewords using a *Patricia tree* [3]. A Patricia tree is a form of index structured as a partial binary tree. The leaves contain values (that is, answers) stored in such a way that the value corresponding to a particular binary key is at the leaf found by traversing a path from the root of the tree, using successive bits of the key to choose the left or right subtree at each internal node. Since the tree is partial, some nodes may have only one descendant. The bits along the paths this creates are usually concatenated and treated as a single branch.

Consider the Patricia tree indexed by the N codewords. Each codeword is at distance at least $2t+1$ from every other code word. A search down the tree of $m-(2t+1)+1 = m-2t$ steps must have uniquely determined a particular code word, since another code words may have agreed with it in every one of the $m-(2t+1)$ bit positions but must then differ from it in those that remain. A code word that differed in one of the first $m-(2t+1)$ bits has already been distinguished. Note that $m-2t \approx \log N$ so that the Patricia tree has effective depth $\log N$.

This establishes that we can satisfy a full query in logarithmic time. However, our queries contain "don't care"s. As a position containing a "don't care" is processed, the search must continue down both branches of the tree. When some set of leaves are reached, the leaf label of the remaining $2t+1$ bits of the code word associated with that leaf is sufficient to determine whether or not that code words satisfies the query. The complexity of a query satisfied by x codewords is therefore the maximum number of distinct edges of a balanced binary tree that must be traversed to reach x leaves. This function is approximately $x \log N$ for small x, and approximately N for large x. Note that x is inversely related to b, the number of bits set in the query.

4 Conclusions

We have presented a signature-based scheme for partial match queries in binary data in which the codewords of error-correcting codes are used as signatures. For queries in which the number of set bits, b is large the overall complexity

of query evaluation is

$$\text{query complexity} \approx \frac{\log N}{b} + n\frac{b^t}{2^b}$$

using a code with N codewords correcting t errors.

References

[1] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[2] V. Gaede and O. Günther. Survey on multidimensional access methods. Technical Report ISS-16, Institut für Wirtschaftsinformatik, Humboldt Universität zu Berlin, August 1995. **www.wiwi.hu-berlin.de/~gaede/survey.rev.ps.Z**.

[3] G. Gonnet. Unstructured data bases or very efficient text searching. In *ACM Principles of Database Systems*, pages 117–124, Atlanta, Georgia, 1983.

[4] A. Guttman. R-Trees: A dynamic index structure for spatial searching. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 47–57, June 1984.

[5] J. Nievergelt, H. Hinterberger, and K.C. Sevcik. The Grid File: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, 1984.

[6] H. Toivonen. Discovery of frequent patterns in large data collections. Technical Report A-1996-5, Department of Computer Science, University of Helsinki, 1996.