3 Low-Support, High-Correlation Mining

We continue to assume a "market-basket" model for data, and we visualize the data as a boolean matrix, where rows = baskets and columns = items. Key assumptions:

1. Matrix is very sparse; almost all 0's.
2. The number of columns (items) is sufficiently small that we can store something per column in main memory, but sufficiently large that we cannot store something per pair of items in main memory (same assumption we've made in all association-rule work so far).
3. The number of rows is so large that we cannot store the entire matrix in memory, even if we take advantage of sparseness and compress (again, sames assumption as always).
4. We are not interested in high-support pairs or sets of columns; rather we want highly correlated pairs of columns.

3.1 Applications

While marketing applications generally care only about high support (it doesn't pay to try to market things that nobody buys anyway), there are several applications that meet the model above, especially the point about pairs of columns/items with low support but high correlation being interesting:

1. Rows and columns are Web pages; $(r, c) = 1$ means that the page of row r links to the page of column c. Similar columns may be pages about the same topic.
2. Same as (1), but the page of column c links to the page of row r. Now, similar columns may represent mirror pages.
3. Rows = Web pages or documents; columns = words. Similar columns are words that appear almost always together, e.g., "phrases."
4. Same as (3), but rows are sentences. Similar columns may indicate mirror pages or plagiarisms.

3.2 Similarity

Think of a column as the set of rows in which the column has a 1. Then the *similarity* of two columns C_1 and C_2 is $Sim(C_1, C_2) = |C_1 \cap C_2|/|C_1 \cup C_2|$.

Example 3.1:

$$\begin{array}{cc} 0 & 1 \\ 1 & 0 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 0 & 1 \end{array} \quad = 2/5 = 40\% \text{ similar}$$

□

3.3 Signatures

Key idea: map ("hash") each column C to a small amount of data [the *signature*, $Sig(C)$] such that:

1. $Sig(C)$ is small enough that a signature for each column can be fit in main memory.
2. Columns C_1 and C_2 are highly similar if and only if $Sig(C_1)$ and $Sig(C_2)$ are highly similar. (But note that we need to define "similarity" for signatures.)

An idea that doesn't work: Pick 100 rows at random, and make that string of 100 bits be the signature for each column. The reason is that the matrix is assumed sparse, so many columns will have an all-0 signature even if they are quite dissimilar.

Useful convention: given two columns C_1 and C_2, we'll refer to rows as being of four types — a, b, c, d — depending on their bits in these columns, as follows:

Type	C_1	C_2
a	1	1
b	1	0
c	0	1
d	0	0

We'll also use a as "the number of rows of type a," and so on.

- Note, $Sim(C_1, C_2) = a/(a + b + c)$.
- But since most rows are of type d, a selection of, say, 100 random rows will be all of type d, so the similarity of the columns in these 100 rows is not even defined.

3.4 Min Hashing

Imagine the rows permuted randomly in order. "Hash" each column C to $h(C)$, the number of the first row in which column C has a 1.

- The probability that $h(C_1) = h(C_2)$ is $a/(a + b + c)$, since the hash values agree if the first row with a 1 in either column is of type a, and they disagree if the first such row is of type b or c. Note this probability is the same as $Sim(C_1, C_2)$.
- If we repeat the experiment, with a new permutation of rows a large number of times, say 100, we get a signature consisting of 100 row numbers for each column. The "similarity" of these lists (fraction of positions in which they agree) will be very close to the similarity of the columns.
- Important trick: we don't actually permute the rows, which would take many passes over the entire data. Rather, we read the rows in whatever order, and hash each row using (say) 100 different hash functions. For each column we maintain the lowest hash value of a row in which that column has a 1, independently for each of the 100 hash functions. After considering all rows, we shall have for each column the first rows in which the column has 1, if the rows had been permuted in the orders given by each of the 100 hash functions.

3.5 Locality-Sensitive Hashing

Problem: we've got signatures for all the columns in main memory, and similar signatures mean similar columns, with high probability, but there still may be so many columns that doing anything that is quadratic in the number of columns, even in main memory, is prohibitive. *Locality-sensitive hashing* (LSH) is a technique to be used in main memory for approximating the set of similar column-pairs with a lot less than quadratic work.

The goal: in time proportional to the number of columns, eliminate as possible similar pairs the vast majority of the column pairs.

1. Think of the signatures as columns of integers.
2. Partition the rows of the signatures into *bands*, say l bands of r rows each.
3. Hash the columns in each band into buckets. A pair of columns is a candidate-pair if they hash to the same bucket in any band.
4. After identifying candidates, verify each candidate-pair (C_i, C_j) by examining $Sig(C_i)$ and $Sig(C_j)$ for similarity.

Example 3.2: To see the effect of LSH, consider data with 100,000 columns, and signatures consisting of 100 integers each. The signatures take 40Mb of memory, not too much by today's standards. Suppose we want pairs that are 80% similar. We'll look at the signatures, rather than the columns, so we are really identifying columns whose *signatures* are 80% similar — not quite the same thing.

- If two columns are 80% similar, then the probability that they are identical in any one band of 5 integers is $(0.8)^5 = 0.328$. The probability that they are *not* similar in *any* of the 20 bands is $(1 - .328)^{20} = .00035$. Thus, all but about 1/3000 of the pairs with 80%-similar signatures will be identified as candidates.

- Now, suppose two columns are only 40% similar. Then the probability that they are identical in one band is $(0.4)^5 = .01$, and the probability that they are similar in at least one of the 20 bands is no more than 0.2. Thus, we can skip at least 4/5 of the pairs that will turn out not to be candidates, if 40% is the typical similarity of columns.

- In fact, most pairs of columns will be a *lot less* than 40% similar, so we really eliminate a huge fraction of the dissimilar columns.

□

3.6 k-Min Hashing

Min hashing requires that we hash each row number k times, if we want a signature of k integers. With *k-min hashing.* In *k-min hashing* we instead hash each row once, and for each column, we take the k lowest-numbered rows in which that column has a 1 as the signature.

To see why the similarity of these signatures is almost the same as the similarity of the columns from which they are derived, examine Fig. refkmin-fig. This figure represents the signatures Sig_1 and Sig_2 for columns C_1 and C_2, respectively, as if the rows were permuted in the order of their hash values, and rows of type d (neither column has 1) are omitted. Thus, we see only rows of types a, b, and c, and we indicate that a row is in the signature by a 1.

$$\begin{array}{ccc} & \begin{array}{cc} \text{Sig1} & \text{Sig2} \end{array} & \\ \begin{array}{c} 100 \\ 1\text{'s} \end{array} & \left(\begin{array}{cc} 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ 1 & 1 \\ \hline 1 & 0 \\ 1 & 1 \end{array} \right) & 100\ 1\text{'s} \end{array}$$

Figure 4: Example of the signatures of two columns using k-min hashing

Let us assume $c \geq b$, so the typical situation (assuming $k = 100$) is as shown in Fig. 4: the top 100 rows in the first column includes some rows that are not among the top 100 rows for the second column. Then an estimate of the similarity of Sig_1 and Sig_2 can be computed as follows:

$$|Sig_1 \cap Sig_2| = \frac{100a}{a+c}$$

because on average, the fraction of the 100 top rows of C_2 that are also rows of C_1 is $a/(a+c)$. Also:

$$|Sig_1 \cup Sig_2| = 100 + \frac{100c}{a+c}$$

The argument is that all 100 rows of Sig_1 are in the union. In addition, those rows of Sig_2 that are not rows of Sig_1 are in the union, and the latter set of rows is on average $100c/(a+c)$ rows. Thus, the similarity of Sig_1 and Sig_2 is:

$$\frac{|Sig_1 \cap Sig_2|}{|Sig_1 \cup Sig_2|} = \frac{\frac{100a}{a+c}}{100 + \frac{100c}{a+c}} = \frac{a}{a+2c}$$

Note that if c is close to b, then the similarity of the signatures is close to the similarity of the columns, which is $a/(a+b+c)$. In fact, if the columns are very similar, then b and c are both small compared to a, and the similarities of the signatures and columns *must* be close.

3.7 Amplification of 1's (Hamming LSH)

If columns are not sparse, but have about 50% 1's, then we don't need min-hashing; a random collection of rows serves as a signature. *Hamming LSH* constructs a series of matrices, each with half as many rows as the previous, by OR-ing together two consecutive rows from the previous, as in Fig. 5.

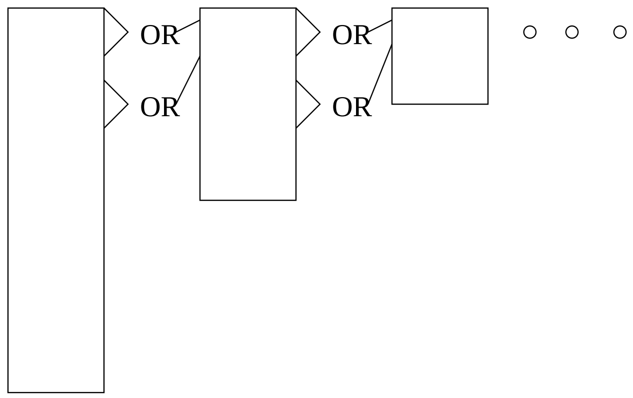

Figure 5: Construction of a series of exponentially smaller, denser matrices

- There are no more than $\log n$ matrices if n is the number of rows. The total number of rows in all matrices is $2n$, and they can all be computed with one pass through the original matrix, storing the large ones on disk.
- In each matrix, produce as *candidate pairs* those columns that:
  1. Have a medium density of 1's, say between 20% and 80%, and
  2. Are likely to be similar, based on an LSH test.
- Note that the density range 20–80% guarantees that any two columns that are at least 50% similar will be considered together in at least one matrix, unless by bad luck their relative densities change due to the OR operation combining two 1's into one.
- A second pass through the original data confirms which of the candidates are really similar.
- This method exploits an idea that can be useful elsewhere: similar columns have similar numbers of 1's, so there is no point ever comparing columns whose numbers of 1's are very different.