Matrix Algebra for Language Comparison

Michael Cysouw

Draft version of May 15, 2017

1 Tables and Matrices

table

matrix

Matrices are nothing special: they are simply tables that contain numbers. For introductory purposes, I will make a crucial terminological distinction here between the term ‘matrix’ and ‘table’. A TABLE is a 2-dimensional structure with rows and columns, and each cell on the cross-section of a row and a column can contain some kind of information. A MATRIX is almost the same, the only difference being that each cell can only contain a number. As long as all information in a data table consists of numbers, there is no difference between a table and a matrix. However, when there are characters or other non-numbers (like missing information) in the table, the conversion into a matrix needs just a little bit more attention (see Section 3).

As a start, I will simply assume that data tables consist of purely numerical data, and that all cells of the table are completely filled. Such data tables are equivalent to matrices. As a first example, consider data tables with some observations as rows (e.g. different languages or words) and various characteristics of these observations as columns. For example, Table 1 shows a data table with a few English words as observations and some simple characteristics of these words as columns. When working with large amounts of such purely

obser- vations	length in letters	frequency of symbol *s*	frequency of symbol *a*
some	4	1	0
words	5	1	0
as	2	1	1
example	7	0	1

Table 1: *Example of a data table with four rows and three columns*

numerical matrices, it turns out to be highly profitable to use the mathematical techniques of manipulating matrices, i.e. the domain of MATRIX ALGEBRA.

matrix algebra

An important consideration for the coding of a data table as a matrix concerns the coding of missing information. In the comparative study of languages it is highly common that a substantial amount of data is missing (or, more general, unevenly distributed), because we do not have enough information on the language(s) in question. This typically occurs in the case of lesser-described languages and in the case of old stages of languages that are only attested through scant manuscripts. For the proper algebraic treatment of missing data, it is best to code missing data as zero in the data matrix, and to add a second matrix to mark the missing data (or, equivalently, to mark the data that is available). If there are just a few cells with missing data, then this second matrix can mostly be ignored. However, when there are many cell with missing data, then some more care has to be taken when interpreting the data (see Section 9).

Before diving into the technical details of matrix algebra, a central question to be answered is why to take this perspective at all. The first and probably most important reason to take this approach is that available computational implementations of matrix algebra are highly efficient and quick, which makes computations on large datasets easy to manage and fast to perform. There is also a second benefit in that the computations used in a particular research project can easily be written down in the form of formulas. These formulas can both simplify the computer code to perform the calculations and simplify the documentation of the research in publishable papers. Finally, the reformulation of various research methods from the field of language comparison into matrix algebra highlights many parallels across methods, and promises a deeper understanding of the methodology of language comparison.

2 A Primer on Matrix Algebra

2.1 Basic Manipulations

When Table 1 is written as a matrix it will look as follows:

$$A = \begin{bmatrix} 4 & 1 & 0 \\ 5 & 1 & 0 \\ 2 & 1 & 1 \\ 7 & 0 & 1 \end{bmatrix}$$

transpose

The TRANSPOSE of a matrix, written here with a superscript T as A^T, is the same matrix, but with the rows and columns reversed (i.e. mirrored around the diagonal from top left to bottom right).

$$A^T = \begin{bmatrix} 4 & 5 & 2 & 7 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

vectorization

The VECTORIZATION of a matrix, written here as a function *vec*(A) is a row-by-row 'unfolding' of a matrix, leading to just a one-dimensional string of numbers, e.g. $vec(A^T) = [4\ 5\ 2\ 7\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1]$.

There are two main kind of algebraic operations on matrices, namely entry-wise operations and row-wise (or column-wise) operations. The most important row-wise operation is the so-called matrix product to be discussed shortly.

entry-wise operations

However, I will first discuss ENTRY-WISE OPERATIONS, which are operations that are applied to each number in the matrix separately. For example, the entry-wise the square of A will be the matrix with each entry squared, namely:

$$A^2 = \begin{bmatrix} 16 & 1 & 0 \\ 25 & 1 & 0 \\ 4 & 1 & 1 \\ 49 & 0 & 1 \end{bmatrix}$$

Likewise, the entry-wise multiplication of A with a scalar like 2 will result in the following:

$$2 * A = \begin{bmatrix} 8 & 2 & 0 \\ 10 & 2 & 0 \\ 4 & 2 & 2 \\ 14 & 0 & 2 \end{bmatrix}$$

In the same sense it is possible to add two matrices together entry-by-entry. Crucially, this entry-wise addition (or subtraction, division, etc.) is only defined when the two matrices have exactly the same number of rows and columns, for example $A + A$ is possible (which is of course the same as $2 * A$):

$$A + A = \begin{bmatrix} 4 & 1 & 0 \\ 5 & 1 & 0 \\ 2 & 1 & 1 \\ 7 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 4 & 1 & 0 \\ 5 & 1 & 0 \\ 2 & 1 & 1 \\ 7 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 8 & 2 & 0 \\ 10 & 2 & 0 \\ 4 & 2 & 2 \\ 14 & 0 & 2 \end{bmatrix} = 2 * A$$

row/column-wise operations

Besides such entry-wise operations there are ROW/COLUMN-WISE OPERATIONS that work on complete rows (or columns) of matrices. The most important of such operations is the special MATRIX PRODUCT between two matrices. A different row-wise product that will be used in various calculations is the KHATRI-RAO PRODUCT.

obser- vations	length in letters	frequency of symbol s	frequency of symbol a
another	7	0	1
set	3	1	0
of	2	0	0
string	6	1	0
examples	8	1	1

Table 2: *Another example of a data table with the same three columns as in Table 1, but with five different words as rows*

2.2 Similarity of Observations: Matrix Product

matrix product

To first understand the MATRIX PRODUCT, consider a second data table as shown in Table 2. This data table consists of five different observations as rows (viz. the words *another, set, of, string, examples*), but exactly the same three characteristics from Table 1 as columns. This is crucial condition for the application of the matrix product. The matrix product can only be applied to two matrices that share at least one dimension.

The idea of a matrix product is that all rows of Table 1 are compared to all rows of Table 2. This comparison is based on the shared characteristics in the columns of the two data tables. So, each word in the rows of Table 1 is compared to each word in the rows of Table 2 on the basis on the characteristics as specified in the columns of the data tables. The result of the matrix product is a new matrix that contains association values (or similarity values) for each pair of observations from Table 1 and 2.

dot product

The central question then is how to compute this similarity between rows, and there will be a lot of discussion of different kinds of similarities (and their counterparts, distances) in what follows. However, a basic notion of similarity between two rows from the data tables is defined as the addition of pairwise multiplication of the numbers in the rows, a function called DOT PRODUCT. For example, the dot product similarity between *some* from Table 1 and *another* from Table 2 is 28 ($= 4 * 7 + 1 * 0 + 0 * 1$).

The reason for exactly this way of computing similarities lies in the historical origin of matrix algebra. It will turn out to be a very useful way of defining similarity, and many other kinds of similarity can be derived from it, most prominently through various kinds of normalisation (see Section 7). Because of the widespread use of exactly this computation in mathematics and physics, there has been extensive research to speed up these calculations. As an effect, the calculation of matrix product is highly optimised and can effectively be used with large data sets.[1]

There is an long tradition to write a matrix product in such a way that the shared dimension (i.e. the columns in our example) appears as the columns of the first matrix, but as the rows of the second matrix (the so-called 'inner' dimensions). To achieve this with our tables, we write them as matrices and then one of the matrices will have to be transposed, as shown below. This results in two matrices that are CONFORMABLE, which simply means that the number of columns of the first matrix is the same as the number of rows of the second matrix.[2]

conformable matrices

$$A \cdot B^T = \begin{bmatrix} 4 & 1 & 0 \\ 5 & 1 & 0 \\ 2 & 1 & 1 \\ 7 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 7 & 3 & 2 & 6 & 8 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Now we have everything we need to define the matrix product, often written as $X \cdot Y$, using a midline dot, reminiscent of the dot product that forms the basis of the calculations (but note that in larger formulae this dot is often left out). It is crucial to differentiate the symbol for matrix product $\cdot$ from the symbol for a entry-wise product $*$, because these represent completely different operations on matrices.

The matrix product $Z = X \cdot Y$ is defined for two conformable matrices X (with n rows and m columns) and Y (with m rows and k columns). The result Z is a matrix listing similarities between all rows of X and columns of Y, so the matrix Z will have n rows and k columns. The entry in row i and columns j of Z is the dot product of row i of matrix X and column j of matrix Y. This is illustrated below for the matrix product $A \cdot B^T$ with the second row of A and the fourth column of B highlighted. The resulting similarity of these two rows

[1] The computations of the dot product can be summarised as $(+, *)$, i.e. 'addition on (top of) pairwise multiplication'. Although this dot product can be used to define many other kinds of similarity, there are other useful definitions of similarity that cannot be easily derived from it, like 'maximum on multiplication' $(\max, *)$, i.e. the maximum of the pairwise multiplications, or 'addition on maximum' $(+, \max)$, i.e. the sum of the pairwise maxima, or $(\max, +)$, or even $(\max, \min)$, which is related to finding shortest paths in graphs.

[2] An important additional distinction is the one between SYNTACTICALLY CONFORMABLE and SEMANTICALLY CONFORMABLE. Syntactically conformable simply means that the matrices have the right size for the matrix product to work. Semantically conformable is a stronger requirement, namely that the order of the inner dimensions is such that their 'meaning' is the same, i.e. the names of the column numbers of the first matrix are in the same order as the names of the rows of the second matrix.

is highlighted in the resulting matrix.

$$A \cdot B^T = \begin{bmatrix} 4 & 1 & 0 \\ 5 & 1 & 0 \\ 2 & 1 & 1 \\ 7 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 7 & 3 & 2 & 6 & 8 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 28 & 13 & 8 & 25 & 33 \\ 35 & 16 & 10 & 31 & 41 \\ 15 & 7 & 4 & 13 & 18 \\ 50 & 21 & 14 & 42 & 57 \end{bmatrix}$$

The meaning of the resulting matrix can be interpreted as shown in Table 3, with higher numbers implying larger similarity. Note that these 'similarities' are rather meaningless in the current form, because of the strange combination of characteristics in our data tables. The data tables consisted of two characteristics with low numbers (the frequencies of certain letters) and one characteristic with relatively high numbers (the total number of letters). The result of the matrix product is dominated by the characteristic with the high numbers. To counteract such influence of certain characteristics, various forms of weighting and normalisation are necessary. This will be discussed in Section 7.

	another	*set*	*of*	*string*	*examples*
some	28	13	8	25	33
words	35	16	10	31	41
as	15	7	4	13	18
example	50	21	14	42	57

Table 3: *Similarities between all words in the rows of Table 1 and Table 2 based on the matrix product*

For presentational purposes, I will in the following use subscripts to indicate the size of matrices. For example, the matrix A_{23} is a matrix with 2 rows and 3 columns. In general, the matrix A_{NP} is a matrix with N rows and P columns. Capitals A, B, C will be used to indicate matrices, while capitals N, M, P will be used to indicate the sizes of the matrices. Note that these capitals N, M, P thus are individual numbers, while A, B, C are matrices consisting of many numbers.

A noteworthy result of matrix multiplication happens with matrices that consist of either one row or one column. Technically, there is nothing special happening here, but the results might be surprising and confusing at first. Consider a matrix A_{41} (i.e. a matrix of size 4×1 with four rows and one column).

Then notice the large difference between $A \cdot A^T = B_{44}$ and $A^T \cdot A = B_{11}$:

$$A \cdot A^T = \begin{bmatrix} 4 \\ 5 \\ 2 \\ 7 \end{bmatrix} \cdot \begin{bmatrix} 4 & 5 & 2 & 7 \end{bmatrix} = \begin{bmatrix} 16 & 20 & 8 & 28 \\ 20 & 25 & 10 & 35 \\ 8 & 10 & 4 & 14 \\ 28 & 35 & 14 & 49 \end{bmatrix}$$

$$A^T \cdot A = \begin{bmatrix} 4 & 5 & 2 & 7 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 5 \\ 2 \\ 7 \end{bmatrix} = \begin{bmatrix} 94 \end{bmatrix}$$

The multiplication $A \cdot A^T$ can be thought of as the comparison of each of four observations to the same four observations in which the comparison is made on just one characteristic. Because there are 4 times 4 comparisons being made, the result has 16 different entries. In contrast, the multiplication $A^T \cdot A$ can be interpreted as a comparison of just one observation to itself on the basis of four different characteristics. The result is just one number for the single comparison being computed.

Matrix multiplication is thus a notion of similarity between all rows of one matrix and all columns of a second matrix. In many practical situations this similarity will be computed simply between all pairs of rows of a single matrix. This means, the first and the second matrix are simply the same matrix, i.e. $A \cdot A^T$.

binary product

A special matrix product that will sometimes be of use is the BINARY PRODUCT, which will be written as $(A \cdot B)_{\text{binary}}$. With the binary product, we are only interested in whether the result of matrix product is non-zero, i.e. there is some kind of similarity attested. In practice, this means that the binary product has an entry of 1 when the two rows share some characteristics, and 0 when they share nothing.

2.3 Cross-section of Characteristics: Khatri-Rao Product

khatri-rao product

A completely different row-wise operation is the KHATRI-RAO PRODUCT.[3] This product is used with a different purpose as the standard matrix product. The Khatri-Rao product is used when we have two matrices with the same observations (as rows), but with different characteristics (as columns). Note that this is the reversed situation as with the matrix product, where we have different

[3] The Khatri-Rao product is sometimes also called a ROW-WISE KRONECKER PRODUCT. Note that in the literature, this product is mostly applied column-wise, but that is completely equivalent to the current treatment, only for transposed matrices.

observations	has s	has o	has m
some	1	1	1
words	1	1	0
as	1	0	0
examples	1	0	1

Table 4: *First example data table for the Khatri-Rao product.*

observations	has e	has a
some	1	0
words	0	0
as	0	1
examples	1	1

Table 5: *Second example data table for the Khatri-Rao product.*

observation (as rows), but the same characteristics (as columns). To differentiate between the different kinds of products, the Khatri-Rao product is indicated with the symbol $\odot$.

As an example, consider the data Tables 4 and 5. These tables have the same observations (the four words: *some, words, as, examples*), but different characteristics. The characteristics are all binary yes/no questions, which makes this example easier to interpret. The Khatri-Rao product now takes the combination of these two data tables, looking at the cross-section of characteristics, as shown in Table 6.

Looking more in-depth into the Khatri-Rao product, this product actually

	has	has	has	has	has	has
observations	s & e	s & a	o & e	o & a	m & e	m & a
some	1	0	1	0	1	0
words	0	0	0	0	0	0
as	0	1	0	0	0	0
examples	1	1	0	0	1	1

Table 6: *Khatri-Rao product of Tables 4 and 5, in effect looking at the cross-section of characteristics.*

makes a three-dimensional array out of two matrices. For example, the first row of Table 4 [1 1 1] is crossed with the first row of Table 5 [1 0], leading to a matrix.

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$$

This is then performed for each of the four rows, leading to four matrices that are stacked on top of each other. Mathematically, the result is then normally called a three-dimensional TENSOR. Note that the data in Table 4 is a matrix A of dimensions 4 × 3 and the data in Table 5 is a matrix B of dimensions 4 × 2. As a three-dimensional array, the product $A \odot B$ would then be a tensor C with dimensions 4 × 3 × 2. This three-dimensional structure is then 'unfolded' as a matrix with dimensions $4 \times (3 * 2)$ by VECTORISING each 'plane' of the tensor, for example continuing the example from above:

tensor

vectorising

$$vec \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} = \left[\begin{array}{cc|cc|cc} 1 & 0 & 1 & 0 & 1 & 0 \end{array}\right]$$

Doing this for all rows of the original matrices A and B leads to the following complete Khatri-Rao product (cf. Table 6):

$$A \odot B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \odot \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} = \left[\begin{array}{cc|cc|cc} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{array}\right]$$

2.4 Combination of Characteristics: Block Matrices

note that $[A|B] \cdot [A|B]^T = AA^T + BB^T$

column binding

$$\begin{bmatrix} A \cdot B & C \cdot D \end{bmatrix} = \begin{bmatrix} A & C \end{bmatrix} \cdot \begin{bmatrix} B & 0 \\ 0 & D \end{bmatrix}$$

special case $A = C$

$$\begin{bmatrix} A \cdot B & A \cdot D \end{bmatrix} = A \cdot \begin{bmatrix} B & D \end{bmatrix}$$

row binding

$$\begin{bmatrix} A\cdot B \\ C\cdot D \end{bmatrix} = \begin{bmatrix} (A\cdot B)^T & (C\cdot D)^T \end{bmatrix}^T = \left(\begin{bmatrix} B^T & D^T \end{bmatrix} \cdot \begin{bmatrix} A^T & 0 \\ 0 & C^T \end{bmatrix} \right)^T$$

so

$$\begin{bmatrix} A\cdot B \\ C\cdot D \end{bmatrix} = \begin{bmatrix} A & 0 \\ 0 & C \end{bmatrix} \cdot \begin{bmatrix} B \\ D \end{bmatrix}$$

special case $B = D$

$$\begin{bmatrix} A\cdot B \\ C\cdot B \end{bmatrix} = \begin{bmatrix} A \\ C \end{bmatrix} \cdot B$$

3 Nominal Data

Although the prospect of using matrix algebra for language comparison is tantalising, there is an important proviso. Namely, a large part of the data tables arising in the context of language comparison is not numerical. Much data is of the so-called ‘nominal’ type (also called ‘categorical’). Instead of listing numbers, such data tables contain categories. An example of a data table illustrating such nominal data is shown in Table 7. This table contains four observations (viz. the words *some, words, as, example*) and two classifications. The first classification assigns to each word a word class, and the second classification describes what kind of symbol occurs at the end of the word. Such classifications consist of a fixed set of alternatives, with no obvious numerical equivalent. They are called NOMINAL VARIABLES or also categorical variables.

nominal variables

observations	word class	last symbol
some	adjective	e
words	noun	s
as	preposition	s
example	noun	e

Table 7: *Example of nominal (or categorical) characteristics*

To make such data amenable for matrix algebra, they have to be converted into a so-called INCIDENCE MATRIX. The basic idea of an incidence matrix is that each class of a nominal variables (e.g. ‘adjective’ or ‘noun’ in the example

incidence matrix

observations	adjective	noun	preposition	final *e*	final *s*
some	1	0	0	1	0
words	0	1	0	0	1
as	0	0	1	0	1
example	0	1	0	1	0

Table 8: *Incidence coding of the nominal variable shown in Table 7*

Table 7) is interpreted as a separated characteristic. Such a recoding is shown in Table 8. The cells of this table are now all numbers, so this table can be turned into a matrix.

However, there are a few special aspects of such incidence matrices that set them apart from 'regular' numerical matrices. First, although the numbers 1 and 0 seem to suggest a binary numerical characteristic, strictly speaking that is not the case. The columns of the incidence table are so-called 'unary' characteristics in which only the presence of 1 has meaning. The presence of 0 actually does not mean that the characteristic is not present. This seemingly hair-splitting distinction becomes important when there are more than one option possible and when there is missing data in the original data table (see Section 8 for the discussion of treating missing data).

Second, the columns of this incidence matrix form groups, namely those columns that belong to the same original nominal variable. This information is important for the proper algebraic usage of the data, and has to be coded in some way. A practical way to encode this information is to prepare an

index matrix

INDEX MATRIX for such incidence matrices that consist of recodings of multiple characteristics. The index matrix just codes which column of the incidence matrix belong to which original nominal variable (included as rows of the index table).[4] The data from Table 7 can then be coded as the following two matrices:

$$\text{Incidence: } A = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}, \text{ Index: } X = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

When the original nominal data table has N rows (observations) and M columns (nominal variables), and the nominal variables have in total K different classes, then the recoded incidence matrix A_{NK} has N rows and K columns, and the

[4] Note that the index matrix is actually just another incidence matrix of a nominal variable, namely the classification of the columns of the first incidence matrix.

index table X_{MK} has M rows and K columns. A direct consequence of this recoding is that the matrix product $A \cdot X^T$ is a matrix of the same size as the original data table:

$$A_{NK} \cdot X_{MK}^T = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

The product of incidence and index table represents the amount of information that was available in the original data table. In the present example this is rather uninteresting, because the original data table was a 4 × 2 table with exactly one piece of information in each cell. However, when there would have been missing data, then the $A \cdot X^T$ would tell us exactly where this missing data was located in the original table (by showing a zero). Also, it can happen that a particular observation belongs to more than one class (e.g. because of uncertainty in the classification, or because there has been some aggregating of slightly different classifications). This can easily be coded in the incidence matrix (the observation simply gets various 1s for different classes of one variable). In such a situation, the product of incidence and index matrix will include a number higher than one, indicating that there were more than one possibility in the original data.

4 Graphs and Matrices

vertices

edges

There is a close connection between matrices and graphs, and sometimes data manipulations using matrix algebra can be easier interpreted when looking at them as a graph. A graph consists of VERTICES ('nodes') and EDGES linking them. Edges can have a numeric value, called WEIGHT, and edges can be directed ('arrows') or undirected ('just links'). In the current context, all graphs will be undirected, and mostly unweighted.

adjacency matrix

The basic notion connecting matrices and graphs is the so-called ADJACENCY MATRIX. An adjacency matrix is a square matrix having a row and a column for each vertex. The position $[i, j]$ in the matrix will be zero if vertex i is not connected to vertex j. In contrast, when the vertices are connected, then the matrix will have an entry of 1. When the edges have weight, then the numerical weights are included in the matrix instead of just the 1s. Undirected graphs are symmetrical, i.e. the entry $[i, j]$ is identical to the entry $[j, i]$. Further, the entries on the diagonal will be 0, as in the current case we have no need for special ‘self-linking’ edges. For example, the graph shown in Figure 1 has the following adjacency matrix:

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

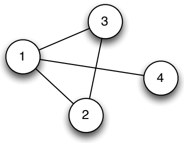

Figure 1: *A simple graph.*

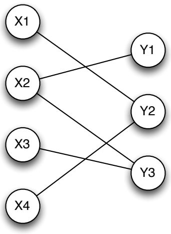

Figure 2: *A bipartite graph.*

biadjacency matrix

bipartite graph

A slight variation is a so-called BIADJACENCY MATRIX in which the rows and columns are not identical. This is used to represent a BIPARTITE GRAPH. Bipartite graphs are graphs in which all edges are between two groups of vertices. An example is shown in Figure 2. In such a case, the adjacency matrix can be reduced to just show one set of vertices as rows (e.g. the ones marked ‘X’ in the Figure) and the other as columns (marked ‘Y’). The biadjacency matrix then is:

$$B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

block matrix

and the corresponding adjacency matrix is (using a so-called BLOCK MATRIX notation, i.e. each entry in A is a matrix in itself):

$$A = \begin{bmatrix} 0_{44} & B \\ B^T & 0_{33} \end{bmatrix}$$

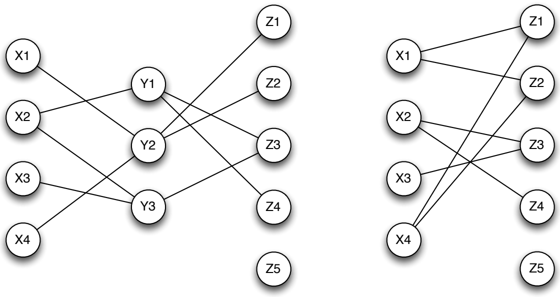

Figure 3: *Left: two linked bipartite graphs. Right: all connections between X and Y without intermediate nodes.*

The interesting question now becomes, what it means to take the matrix product between two graphs. As an example, consider the graph on the left side of Figure 3. Below, the connections between X and Y are coded as biadjacency matrix A, and the connections between Y and Z are coded in matrix B. The matrix product $A \cdot B$ then represents all connections that exist between X and Z, as shown to the right of Figure 3. So, matrix multiplication between biadjacency matrices represents a notion of finding all possible connections. Note that when there are two different paths linking a vertex from X with a vertex from Z (e.g. X_2 and Z_3 in Figure 3, being connected both via Y_1 and Y_3), then the matrix product lists the number of possible paths.

$$A \cdot B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

With weighted graphs the meaning of the matrix product becomes a bit less intuitive. Instead of just summarising all attested connections, the numbers in the resulting matrix are the multiplied weights of the individual connections. And when there are multiple paths, then the resulting value will the the sum of the products of all possible paths.[5] Often the only information of interest is whether two vertices are connected or not, then is suffices to look at the BINARY PRODUCT $(A \cdot B)_{\text{binary}}$, discussed at the end of Section 2.

binary product

[5] It seems more intuitive to have to matrix product of two graphs result in the shortest path, for example. This can be achieved by taking the $(\min, +)$ matrix product (see footnote 1 on page 5). Note that such alternative definitions of the matrix product are not available in implementations of fast matrix multiplication, so in practice this option is not viable.

5 Sparse Matrices

When using matrix algebra for large datasets, there will be many instances of so-called SPARSE MATRICES. Sparse matrices are matrices in which the large majority of entries are zero. Actually, there is nothing special with such sparse matrices: all algebra works just as with all other matrices. The only import consideration is that large matrices possibly take up a lot of RAM ("working memory") during computations. For that reason special implementations have been developed to more economically deal with sparse matrices. Basically, the principle is to only encode the non-zero entries of the matrix, and assume that everything else is zero. When dealing with matrices that only have non-zero numerical entries for every 10 or 100 thousand zeros (which is nothing special), such a sparse recoding makes a crucial difference for the practical feasibility of computations.

sparse matrices

In most cases, the available sparse matrix routines suffice completely for working with spare matrices.[6] However, there are two important consideration of which to take note. First, there are various different sparse formats, which are more or less economical for different operations. As a result, a sparse matrix might have to be converted to another format to optimise specific computations. The implementations in *R* or *Python* are being enhanced to do any such conversion on the fly when necessary, but this does not (yet) always work. So, sometimes a slow operation might be sped up considerably by manually changing the sparse matrix format.

Second, care has to be taken in computations that sparsity is not lost. This can happen quite easily. For example, when taking an entry-wise logarithm of a sparse matrix, all empty cells are suddenly filled with content (because $\log(0) = -\infty$), and sparsity is lost. This results in an immediate filling of the complete RAM of most computers, reducing the speed of any computation to a crawl. Such unwanted effects are normally not automatically caught by the implementations, so care has to be taken not to let this happen. For example, in the case of the logarithm above, it is mostly the case that the logarithm should only be taken for the non-zero cells. Yet, this has to be explicitly specified in all presently available implementations.

In this methodological survey, I will assume that zero cells in sparse matrices do not take part in any calculations. For example, the logarithm of zero remains zero, and zero divided by zero also remains zero. Or, differently formulated, for sparse matrices I will assume that $\log(0) = 0$ and $0/0 = 0$.

[6] For sparse matrix routines, see for example, the library *Matrix* in *R*, or *ScyPi ssm* in *Python*.

5.1 Building Sparse Matrices

There are many ways to actually build sparse matrices from some data table. coordinate format A useful approach is to use the so-called COORDINATE FORMAT, specifying the coordinates of the non-zero entries of the sparse matrix. This normally takes the form of a list of row numbers and a list of column numbers of the non-empty cells of the table. To turn a data table with nominal variables (see Section 3) factorization into coordinate format, we need a process which I will call FACTORIZATION. As an example, consider the data in Table 9.

Language	has phoneme
English	θ
English	x
English	f
German	x
German	f
German	pf

Table 9: *Example data linking languages to its phonemes*

Language	θ	x	f	pf
English	1	1	1	0
German	0	1	1	1

Table 10: *Incidence coding of the data in Table 9*

Basically, factorization performs two subtaskts, namely it (i) transforms a nominal variable into a list of unique NAMES and (ii) returns the original nominal variable as a list of RANKS. For example, the column 'Language' from Table 9 will be transformed into the unique category-names [*English, German*] and the ranks $[1, 1, 1, 2, 2, 2]$. Doing the same factorization for the second columns 'has phonemes' transforms this column into the unique names [*θ, x, f, pf*] and the ranks $[1, 2, 3, 2, 3, 4]$.

Now, with two factorized columns we can easily make a sparse matrix using the coordinate format. For example, we can take the factorized ranks from 'Language' as row coordinates $x = [1, 1, 1, 2, 2, 2]$ and the factorized ranks from 'has phonemes' as column coordinates $y = [1, 2, 3, 2, 3, 4]$ to obtain an incidence matrix as shown in Table 10, with the coordinates specifying the cells starting to count from the top left.

Stripping away the row names and the column names, we are left with the following matrix T. Instead of the expected zeros, I have used dots in the table to highlight the fact that this matrix is a sparse matrix. In a sparse matrix format the zeros are left unspecified. Only the entries as listed by the coordinates are explicitly mentioned.

$$C = \begin{bmatrix} 1 & 1 & 1 & \cdot \\ \cdot & 1 & 1 & 1 \end{bmatrix}$$

5.2 Special Sparse Matrices

type-token matrix

A special kind of a sparse matrix in coordinate format is a TYPE-TOKEN MATRIX. For a type-token matrix only one column of a data table is used. This column is factorized and the ranks are used as the column coordinates of the coordinate format. The row coordinates are simply the numbers 1 to N, with N being the number of rows of the original data table.

For example, the first column 'Language' from Table 9 is factorized into the ranks $[1,1,1,2,2,2]$ as column coordinates, with the row coordinates 1 through 6, i.e. $x = [1,2,3,4,5,6]$. This specifies the sparse type-token matrix C_L.

Doing the same for the second column 'has phoneme' from Table 9, we obtain a sparse type-token matrix C_P with column coordinates $y = [1,2,3,2,3,4]$ and row coordinates 1 through 6, i.e. $x = [1,2,3,4,5,6]$. Now note that the matrix product $C_L^T \cdot C_P$ gives exactly the matrix C from above.

$$C_L^T \cdot C_P = \begin{bmatrix} 1 & 1 & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdot \\ \cdot & 1 & 1 & 1 \end{bmatrix} = C$$

permutation matrix

A special kind of type-token matrix is a PERMUTATION MATRIX in which the column coordinates are a permutation of the row coordinates 1 through N. For example, for $N = 5$, the row coordinates are $x = [1,2,3,4,5]$, and the column coordinates are a permutation of the numbers 1 through N, e.g. $y = [3,5,1,4,2]$, specifying the following permutation matrix. Such permutation matrices can be used to reorder rows or columns of other matrices to make them SEMANTICALLY CONFORMABLE (see Footnote 2)

$$C = \begin{bmatrix} \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 \\ 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot \end{bmatrix}$$

6 Identity, Diagonal and Unit Matrices

As already alluded to various times in the previous sections, the basic matrix product will mostly not give precisely the intended results. In most cases some level of 'massaging' is necessary to balance the effects of higher and lower frequencies in the data. There is an enormous variety of possible weightings and normalisations proposed in the literature, but I will only discuss a few major possibilities here, using matrix manipulations to derive them. For these manipulations (that will be discussed in the next sections), we need some special matrices: (i) the identity matrix, (ii) diagonal matrices, (iii) diagonal block matrices (iv) shift matrices, and (v) unit matrices.

identity matrix

First, the IDENTITY MATRIX (mostly written with a capital I) is a square matrix with 1s on the diagonal, and zeros elsewhere. Any matrix product with I results in no change, i.e. $A \cdot I = I \cdot A = A$. So the identity matrix functions just like the number 1 with normal multiplication, i.e. $a * 1 = 1 * a = a$. The size of the identity matrix of course differs depending on the context. For example, the product $A_{47} \cdot I$ implies that the identity matrix is of size 7 × 7, otherwise it would not be conformable with A. To be explicit, one could write $A_{47} \cdot I_{77}$, but this is mostly left implicit.

$$I_{44} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

diagonal matrix

Second, a DIAGONAL MATRIX is similar to the identity matrix in that all entries are zero, except for those on the diagonal. However, the diagonal can consist of all kind of different numbers, and not just of 1s. The typical construction of a diagonal matrix is that a vector of numbers, e.g. $v = (2, 5, 8, 3)$

is turned into a diagonal matrix of size 4 × 4 by using the operator $\operatorname{diag}(v)$.

$$\operatorname{diag}(v) = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

Note that $\operatorname{diag}(v) \cdot A$ is not the same as $A \cdot \operatorname{diag}(v)$, just as in general the matrix product is not commutative (i.e. $A \cdot B \neq B \cdot A$), except for a few special cases (for examples, see below).

diagonal block matrix

Third, a DIAGONAL BLOCK MATRIX is like a diagonal matrix, except that the entries on the diagonal can be matrices themselves. For example, consider the index matrix X from Section 3. This is a diagonal block matrix, with two blocks of size 1 × 3 and 1 × 2.

$$X = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Also the product $X^T \cdot X$ is a block diagonal matrix, this time with two block of size 3 × 3 and 2 × 2:

$$X^T \cdot X = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

shift matrix

Fourth, a SHIFT MATRIX is similar to the identity matrix, except that the entries are not on the main diagonal, but on a 'shifted' diagonal. The two most important shift matrices are the UPPER SHIFT MATRIX U and the LOWER SHIFT MATRIX L. Note that $U = L^T$.

$$U_{55} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, L_{55} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

When multiplying a matrix A with a shift matrix, this has the effect that all entries of A are shifted, for example:

$$U_{44} \cdot A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

unit matrix

Finally, a UNIT MATRIX, sometimes also called a 'matrix of ones,' is a matrix that consists completely of 1s. Such matrices can have any size, so the size mostly has to be explicitly indicated. As a notation, either a number '1' with subscripts is used, or a capital J. Note the difference between the following two products:

$$\text{diag}(v) \cdot 1_{44} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 \\ 8 & 8 & 8 & 8 \\ 3 & 3 & 3 & 3 \end{bmatrix}$$

$$1_{44} \cdot \text{diag}(v) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 5 & 8 & 3 \\ 2 & 5 & 8 & 3 \\ 2 & 5 & 8 & 3 \\ 2 & 5 & 8 & 3 \end{bmatrix}$$

A very useful unit matrix is a unit matrix with only one row or one column. These unit matrices effectively compute the column sums or rows sums of another matrix A, respectively. For example:

$$\text{Row sums: } A_{43} \cdot 1_{31} = \begin{bmatrix} 4 & 1 & 0 \\ 5 & 1 & 0 \\ 2 & 1 & 1 \\ 7 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 4 \\ 8 \end{bmatrix}$$

$$\text{Column sums: } 1_{14} \cdot A_{43} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 4 & 1 & 0 \\ 5 & 1 & 0 \\ 2 & 1 & 1 \\ 7 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 18 & 3 & 2 \end{bmatrix}$$

However, note that most matrix algebra implementations have special, highly efficient, functions for taking row or column sums that are preferably over the explicit multiplication with a unit-matrix. However, for notational reasons the product with the unit-matrix is preferable. Also note that large unit-matrices are not sparse in current implementations.[7]

The following combination of diagonal and unit matrices will become useful later when dealing with sparse matrices. Basically, when a 1-column matrix A_{N1} is multiplied with a 1-row matrix B_{1M}, then the result will be an $N \times M$ matrix C_{NM} (see the end of Section 2). This product can be reformulated as a product of diagonal matrices and a unit matrix as follows:

$$A_{N1} \cdot B_{1M} = \text{diag}(A_{N1}) \cdot 1_{NM} \cdot \text{diag}(B_{1M})$$

[7] Sparse unit-matrices would of course be possible, for example by explicitly specifying a constant 1, instead of assuming a constant zero by default.

7 Weighting, centering and normalisation

Three different kinds of 'massaging' of matrices will be discussed in this section: (i) weighting of the rows and/or columns, (ii) centering, and (iii) normalisation of the rows (i.e. normalisation of the entities to be compared). Normally not all of them are used at the same time, but technically it would be possible. It is still to a large extend a question of gut feeling (or trial and error) which option is the most profitable for a data set at hand. When these three different operations are combined, then they are normally performed in the order discussed here, namely, first weighting, then centering of the weighted matrices, and then normalisation of the weighted and/or centred matrices.

7.1 Weighting

column weighting

Row and/or column weighting of a matrix are different ways to weight the values in the matrix. For example, COLUMN WEIGHTING is found in the concept of 'inverse document frequency' (IDF) as used in information retrieval. Also the notion of the 'gewichteter Identitätswert' (GIW) in dialectometry, as introduced by Hans Goebl, is a weighting of this kind. The basic principle is to compute the sums of each column of a matrix, and then inversely weight the frequencies of the columns by those sums. Formulated in this way, Goebl actually uses the square root of the inverse as a weight, i.e. $\sqrt{1/\text{column sum}}$, while in information retrieval mostly the logarithm of this fraction is used, specifically $\log(D/\text{column sum})$, with D being the length of the column. Technically, this can be formulated as a matrix manipulation as follows. Given a matrix A_{NP}, the column sums are given by $1_{1N} \cdot A_{NP}$. Then, for example, the inverse of these sums are put into a diagonal matrix, and then the product is taken with the original matrix A. An example using a simple fraction is shown below:

$$A_{43} \cdot \operatorname{diag}\left(\frac{1}{1_{14} \cdot A_{43}}\right) = \begin{bmatrix} 4 & 1 & 0 \\ 5 & 1 & 0 \\ 2 & 1 & 1 \\ 7 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{18} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{4}{18} & \frac{1}{3} & 0 \\ \frac{5}{18} & \frac{1}{3} & 0 \\ \frac{2}{18} & \frac{1}{3} & \frac{1}{2} \\ \frac{7}{18} & 0 & \frac{1}{2} \end{bmatrix}$$

row weighting

The same procedure also works for row weighting, although this is less frequently used. ROW WEIGHTING basically works just as column weighting, so one could for example normalise both rows and columns by the inverse of the row or column sums. Note that the row weighting will be a diagonal matrix added *in front* of the to-be-normalised matrix A. The following formula describes a combination of row and column weighting, both by a simple inverse weighting schema:

$$\text{diag}\left(\frac{1}{A_{NP} \cdot 1_{P1}}\right) \cdot A_{NP} \cdot \text{diag}\left(\frac{1}{1_{1N} \cdot A_{NP}}\right)$$

The practical implementation using diagonal matrices is not the only possibility to compute weighted matrices. However, this approach is crucial to retain sparsity.

7.2 Centering

centering

CENTERING is a special normalisation on rows, typically used in the Pearson product moment correlation and the closely related Cramer's ϕ. The basic principle is that for each row the average is computed, and then this average is deduced from the row. For the row average, we first have to compute the row sum, though in a special way to get a matrix back of the same size as A, namely $A_{NP} \cdot 1_{PP}$, and divide this by the number P to get the average. The result is then subtracted from the original matrix A:

$$A_{43} \cdot \frac{1_{33}}{3} = \begin{bmatrix} 4 & 1 & 0 \\ 5 & 1 & 0 \\ 2 & 1 & 1 \\ 7 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} & \frac{5}{3} & \frac{5}{3} \\ \frac{6}{3} & \frac{6}{3} & \frac{6}{3} \\ \frac{4}{3} & \frac{4}{3} & \frac{4}{3} \\ \frac{8}{3} & \frac{8}{3} & \frac{8}{3} \end{bmatrix}$$

$$A_{43} - \left(A_{43} \cdot \frac{1_{33}}{3}\right) = \begin{bmatrix} 4 & 1 & 0 \\ 5 & 1 & 0 \\ 2 & 1 & 1 \\ 7 & 0 & 1 \end{bmatrix} - \begin{bmatrix} \frac{5}{3} & \frac{5}{3} & \frac{5}{3} \\ \frac{6}{3} & \frac{6}{3} & \frac{6}{3} \\ \frac{4}{3} & \frac{4}{3} & \frac{4}{3} \\ \frac{8}{3} & \frac{8}{3} & \frac{8}{3} \end{bmatrix} = \begin{bmatrix} \frac{7}{3} & -\frac{2}{3} & -\frac{5}{3} \\ \frac{9}{3} & -\frac{3}{3} & -\frac{6}{3} \\ \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ \frac{13}{3} & -\frac{8}{3} & -\frac{5}{3} \end{bmatrix}$$

This normalisation can be nicely rewritten as

$$A_{NP} - \left(A_{NP} \cdot \frac{1_{PP}}{P}\right) = A_{NP}\left(I - \frac{1_{PP}}{P}\right).$$

centering operator

The portion between brackets $C = I - \frac{1_{PP}}{P}$ is called the CENTERING OPERATOR, as it performs the centering on a numerical matrix via a matrix product. Note that the centering operator can easily be combined with the previous row/column weighting as follows:

$$\text{diag}\left(\frac{1}{A_{NP} \cdot 1_{P1}}\right) \cdot A_{NP} \cdot \text{diag}\left(\frac{1}{1_{1N} \cdot A_{NP}}\right) \cdot \left(I - \frac{1_{PP}}{P}\right)$$

7.3 Centering for Sparse Matrices

The centering operator has the interesting properties that, first, $C \cdot C = C$, as centering an already centred matrix does not change anything anymore, and, second, $C = C^T$ because C is symmetric. This implies that the matrix product between two centred matrices reduces to just one centering operation:

$$(A_{NP} \cdot C_{PP}) \cdot (B_{MP} \cdot C_{PP})^T = A \cdot C \cdot C^T \cdot B^T = A \cdot C \cdot B^T$$

An important drawback of centering is that the centering operator is not sparse, and thus a centred matrix is likewise not sparse. For large sparse matrices this approach to centering is thus not a real option. However, it is possible to compute a matrix product between two centred sparse matrices by assuming that only those pairs of rows that do share some characteristics have to be centred (i.e. those cells for which $(A \cdot B^T)_{\text{binary}} = 1$). This can be implemented by taking into account that:

sparse centering

$$A_{NP} \cdot 1_{PP} \cdot B_{MP}^T = (A_{NP} \cdot 1_{P1}) \cdot \left(1_{1P} \cdot B_{MP}^T\right) = (A_{NP} \cdot 1_{P1}) \cdot (B_{MP} \cdot 1_{P1})^T$$

and, using the identity proposed at the end of Section 6, that:

$$(A_{NP} \cdot 1_{P1}) \cdot (B_{MP} \cdot 1_{P1})^T = \text{diag}\,(A_{NP} \cdot 1_{P1}) \cdot \left(A_{NP} \cdot B_{MP}^T\right)_{\text{binary}} \cdot \text{diag}\,(B_{MP} \cdot 1_{P1})$$

leading to the following approach for the centred matrix product of two sparse matrices A and B:

$$A \cdot C \cdot B^T = A \cdot \left(I - \frac{1_{PP}}{P}\right) \cdot B^T = A \cdot I \cdot B^T - A \cdot \frac{1_{PP}}{P} \cdot B^T =$$
$$A \cdot B^T - \text{diag}\,(A \cdot 1_{P1}) \cdot \frac{(A \cdot B^T)_{\text{binary}}}{P} \cdot \text{diag}\,(B \cdot 1_{1P})$$

Note that this last step is an approximation to centering, because the zeros are not centred. In practice with sparse matrices this mostly leads to just small differences to the non-sparse implementation of centering (i.e "Pearson correlation"). The strongest deviation occurs in the slightly negative correlation values. When those values are important for any computation, then this approximation should not be used. The problem is most virulent in situation in which many non-zero values are correlated with zero values. Any correction for these errors will greatly decrease the computational efficiency.

7.4 Normalisation

row normalisation

Third, ROW NORMALISATION technically works just as row weighting (as discussed above). However, in practice row normalisation is mostly used after other 'massaging' has already been performed. The most common normalisation uses an EUCLIDEAN NORM, which is the same as STANDARD DEVIATION in statistics. The reason for these decisions is that the resulting similarity value in the end will then nicely be normalised between −1 and +1.

euclidean norm

standard deviation

In practice, given any matrix A (possibly with already some columns normalisation or centering applied), the Euclidean norm is computed for all row vectors of the matrix. The Euclidean norm of a vector $(v_1, v_2, v_3 \dots v_n)$ is written as $||v||_2$ and defined as:

$$||v||_2 = \sqrt{\sum_{i=1}^{n} v_i^2}$$

In terms of matrices, applying the Euclidean norm to all rows at once, this becomes the following (note that the various powers of matrices are entry-wise operations, not a matrix product of A with itself):

$$\sqrt{A_{NP}^2 \cdot 1_{P1}} \text{ or written differently: } \left(A_{NP}^2 \cdot 1_{P1}\right)^{1/2}$$

The second formulation already suggests that instead of the number 2 also other numbers are possible. Indeed, there is a whole family of norms depending on the number used, i.e. $(A_{NP}^x \cdot 1_{P1})^{1/x}$. Most interestingly, with $x = 1$ this reduces to the row sums that we have already seen before. The inverses of these norms are then used as row normalisation. For example, when the matrix A is first centred as B and then row-normalised as C it will look as follows:

$$B = A_{NP} \cdot \left(I - \frac{1_{PP}}{P}\right) \text{ and then } C = \text{diag}\left(\frac{1}{\sqrt{B_{NP}^2 \cdot 1_{P1}}}\right) \cdot B$$

The crucial difference between row normalisation and row weighting (which are technically pretty similar) is that row normalisation is only applied after all other 'massaging' has already taken place. Row normalisation is only applied to normalise all results to values between 0 and 1, so it is performed on top of any weighting or centering. If there is no weighting or centering applied, then row normalisation is in practice the same as row weighting with an Euclidean norm.

7.5 Famous Measures of Association

correlation coefficient
cosine similarity
TF-IDF

With these different kinds of 'massaging' some famous measures of association (or similarity) can be computed. For all these association measures we want to compute the association between all rows of a matrix A and all rows of a matrix B (Note that these two matrices might be the same matrix). Now, instead of just taking the matrix product $A \cdot B$, we first 'massage' the individual matrices, and then take the matrix product. Depending on the normalisation, this matrix product will results in the following association measures: (i) Pearson's CORRELATION COEFFICIENT r, by using no weighting, but centering and row normalisation, (ii) Cramer's ϕ is the same as the Pearson correlation coefficient when the original matrices only contain zeros and ones, (iii) the COSINE SIMILARITY, by using no weighting, no centering, but only row normalisation, and (iv) TF-IDF ('term frequency - inverse document frequency') by column weighting, no centering, and row normalisation.

8 Observed vs. Expected

observed association
expected association

Another, slightly different approach to the 'massaging' of a matrix product $A{\cdot}B^T$ defines a notion of statistical expectation for the result of the matrix product. So we start with two matrices A and B (which might be weighted, but not centred or normalised as described in the previous paragraph) and then define the matrix with the OBSERVED ASSOCIATION 'O' as $A{\cdot}B^T$. This observed association matrix is compared with a matrix with the EXPECTED ASSOCIATION 'E'. There exist various different ways to combine O and E into a measure of association. But the first question is how to define the expectation. For this definition we have to distinguish between numerical and nominal data.

8.1 Expectation of Numerical Data

Consider as an example two vectors $v = (3, 7, 4, 9)$ and $w = (1, 5, 5, 8)$. The dot product gives the observed association $O = v \cdot w$ equals $(3 * 1) + (7 * 5) + (4 * 5) + (9 * 8) = 130$. As a basic statistical expectation for this similarity one can just start with the average of both vectors, e.g. $\hat{v} = {}^{3+7+4+9}/_{4} = {}^{23}/_{4}$ and $\hat{w} = {}^{1+5+5+8}/_{4} = {}^{19}/_{4}$. These averages represent a basic estimate of the probabilities of the values in these vectors. The product of these values will be an expectation, and that product has to be added together four times (as there are four numbers in the vectors). So, the expected association would then be $E = \hat{v} * \hat{w} * 4 = {}^{23}/_{4} * {}^{19}/_{4} * 4 \approx 109$. This means that the observed association ($O = 130$) is higher than the expected association ($E = 109$), which suggests a conclusion that the vectors v and w are more similar than expected from chance alone.

This basic estimate of expectation can easily be formulated for complete matrices. We simply take the sum of each row from A_{NP} divide this by the lengths of the rows P, and do likewise for each row from B_{MP}. This represents taking the averages of all rows. Then we take all pairwise products of these values, and multiply this with P. Note that the P's partially cancel each other, resulting in a expectation matrix E of size $N \times M$:

$$E_{NM} = \frac{(A_{NP} \cdot 1_{P1})}{P} \cdot \frac{(B_{MP} \cdot 1_{P1})^T}{P} * P = \frac{(A_{NP} \cdot 1_{P1}) \cdot (B_{MP} \cdot 1_{P1})^T}{P}$$

Strictly speaking, this is just one of the numerous ways to define an expectation. Depending on what is known about the data, other expectations might be formulated.

8.2 Expectation, Centering and Sparsity

There is a strong similarity between the calculation of the expectation and centering. This is no coincidence, as centering is the process reducing the expectation to zero. This is the reason why comparing observed vs. expected is not feasible for centred matrices, because some simple arithmetic shows that the average of a centred matrix is zero.

$$A_{NP} \cdot \left(I - \frac{1_{PP}}{P} \right) \cdot 1_{P1} = A_{NP} \cdot 1_{P1} - A_{NP} \cdot \frac{1_{PP}}{P} \cdot 1_{P1} = A_{NP} \cdot 1_{P1} - A_{NP} \cdot 1_{P1} = 0$$

We can also see the similarity between the calculation of centering and the calculation of the expectation by slightly reformulating the formula for the matrix with the expected values:

$$E_{NM} = \frac{(A_{NP} \cdot 1_{P1}) \cdot (B_{MP} \cdot 1_{P1})^T}{P} = \frac{A_{NP} \cdot (1_{P1} \cdot 1_{1P}) \cdot B_{MP}^T}{P} = A_{NP} \cdot \frac{1_{PP}}{P} \cdot B_{MP}^T$$

This reformulation makes clear that the calculation of an expectation matrix loses sparsity when the matrices A and B are sparse. There is a trick to keep sparsity, basically by only calculating the expectation for those cells of the matrix in which there is at least some similarity between A and B, i.e. for those cells in which $(A \cdot B^T)_{\text{binary}} = 1$. This leads to the following sparse calculation of the expectation matrix:

$$E_{NM} = \text{diag}\,(A_{NP} \cdot 1_{P1}) \cdot \frac{(A \cdot B^T)_{\text{binary}}}{P} \cdot \text{diag}\,(B_{MP} \cdot 1_{1P})$$

8.3 Expectation of Nominal Data

Two conformable incidence matrixes A_{NP} and B_{MP} necessarily share the same index matrix X_{QP} (see Section 3). The expectation matrix is then given by:

$$E_{NM} = A \cdot \frac{X^T \cdot X}{X^T \cdot X \cdot X^T \cdot X} \cdot B^T$$

Although this looks radically different, this expectation is actually similar to the definition of expectation for numerical data above. Basically, the fraction in the middle is a block diagonal matrix consisting of square blocks of the form ${}^{1_{ZZ}}/_{Z}$. All Q nominal variables are represented by one such block, in which Z is the number of values of each variable. Note that the product $X^T \cdot X$ is sparse, and thus that here $0/0 = 0$.

In the situation with only one nominal variable, the index matrix X_{QP} reduces to 1_{1P} and thus:

$$E_{NM} = A \cdot \frac{1_{PP}}{P} \cdot B^T$$

8.4 Famous Measures of Association

There are many measures of association that can be formulated as a combination of observed O and expected E matrices, and only a very limited selection of examples will be presented here. Most of these measures are positive when $O > E$, negative when $O < E$, and zero when $O = E$. They are normally not bound between -1 and $+1$, but can (theoretically) attain infinite high or low results.

Care has to be taken in the following calculations with handling the zeros in sparse matrices. Basically, when both O and E are sparse (with identical empty cells), then the following measures should not be calculated for those empty cells. Or, differently formulated, we have to define that ${}^0/_0 = 0$ and that $\log({}^0/_0) = 0$.

pearson residuals

The first measure is called PEARSON RESIDUALS (PR). This name derives from fact that Pearson's χ^2 statistic is the sum of the squares of these values.

$$\text{Pearson Residuals (PR)} = \frac{O - E}{\sqrt{E}}$$

pointwise mutual information

The second measure presented here is the POINTWISE MUTUAL INFORMATION (PMI). This name derives from the fact that mutual information can be seen as a weighed average of these values. PMI is defined in terms of probabilities, but assuming that our values are uniformly distributed (which is actually log-odds scores

in most linguistic cases a wrong assumption), these probabilities can easily transformed into our O and E values, as shown below. In bioinformatics this is known as the LOG-ODDS SCORES.[8]

$$\text{Pointwise Mutual Information (PMI)} = \log\left(\frac{p_{xy}}{p_x * p_y}\right)$$
$$= \log\left(\frac{N * p_{xy}}{N * p_x * p_y}\right) = \log\left(\frac{O}{E}\right)$$

A slightly more complex computation of the PMI assumes that the probabilities follow a poisson distribution. This is typically suitable in situation with binary characteristics and with heavy-tailed phenomena of interest (i.e. some observations are highly frequent, while there are very many that are rare). Situations like comparing typological parameters across languages, or sounds or words in texts are typical examples of such distributions. The probability is then defined as follows, with k_i being the observed frequency of the phenomenon i, and λ a parameter to be estimated:

$$p(i) = \frac{\lambda^{k_i} e^{-\lambda}}{k_i!}$$

The parameter λ can be estimated as the average frequency of all phenomena k_i. If there are n different phenomena to be studied (e.g. letters in a text), then λ can be understood as a token-type ratio.

$$\lambda = \frac{1}{n}\sum_{i=1}^{n} k_i = \frac{\text{number of tokens}}{\text{number of types}}$$

Expressed in terms of matrices, this all results in the following measure of association. Given A_{NP} and B_{MP}, then comparing all N to all M using the binary characteristics P, we have the row sums $F^A_{N1} = A_{NP}1_{P1}$ and $F^B_{M1} = B_{MP}1_{P1}$, and the token/type constants $\lambda^A = {}^{1_{1N}F^A_{N1}}/_{N}$ and $\lambda^B = {}^{1_{1M}F^B_{M1}}/_{M}$, so using

$$\log p_i = \log\left(\frac{\lambda^{k_i}\ e^{-\lambda}}{k_i!}\right) = k_i \log\lambda - \lambda - \log k_i!$$

[8] Confusingly, this is not related to log-odds ('logits') or the odds ratio from logistic regression. Actually, log-odds is a bit a misnomer here, the coinage seems to go back to Altschul 1991:557. The reason for the log-odds name seems to be that $\frac{p(x,y)}{p(x)*p(y)} = \frac{p(x|y)}{p(x)}$, and this is interpreted as the 'odds' of (i) x occurring when also y occurs, relative to (ii) x occurring overal.

we can define

$$\begin{aligned} D^A &= \text{diag}\ \left(F^A \log \lambda^A - \lambda^A - \log F^A!\right) \\ D^B &= \text{diag}\ \left(F^B \log \lambda^B - \lambda^B - \log F^B!\right) \end{aligned}$$

Then, as above, defining $O_{NM} = A_{NP} \cdot B^T_{MP}$ and O_{bin} as the binary version of O, we derive the token/type constant $\lambda^O = 1_{1N}O_{NM}1_{M1}/(N*M)$, so that we can define:

$$\begin{aligned} P^O &= O \log \lambda^O - \lambda^O - \log O! \\ P^A &= D^A \cdot O_{bin} \\ P^B &= O_{bin} \cdot D^B \end{aligned}$$

Then finally using definition of PMI

$$\text{PMI} = \log\left(\frac{p_{xy}}{p_x p_y}\right) = \log p_{xy} - \log p_x - \log p_y$$

poisson-based pointwise mutual information

we get the POISSON-BASED POINTWISE MUTUAL INFORMATION measure of association

$$\text{PMI}_{poisson} = P^O - P^A - P^B$$

The third measure of association between two matrices is based directly on the Poisson distribution, which might thus be calculated as follows:

$$\frac{E^O * e^{-E}}{O!}$$

However, this formulation is only defined when the observed values O are whole numbers, and the results of this formula gives a probability value, i.e. a good association will give values close to zero (e.g. with significance below 0.01 or so). A more suitable measure of association is obtained by taking the negative logarithm of this, which results in:

$$-\log\left(\frac{E^O * e^{-E}}{O!}\right) = E + \log(O!) - O * \log(E)$$

Note that the $\log(O!)$ is easy to approximate computationally by using the logarithm of the Γ-function instead of the factorial (e.g. using the function *lfactorial* in *R*). However, there are still some difficulties with this measure of association. The first problem is that this formula evaluates the *difference* between O and E, which implies that the result is both positive when $O > E$ and when $O < E$. To separate between these crucially different cases, it is necessary to add a correct sign, namely a plus when $O > E$ and a minus when $O < E$. This can be achieved by multiplying the formula with, for example, something like *sign*$(O - E)$. The second problem is that the formula does not reach zero when $O = E$, but the result for $O = E$ turns out to depend on the size of O (which is difficult to interpret). Fortunately, this can easily be corrected by subtracting the case with $O = E$, leading to a somewhat surprising result.[9]

$$[E + \log(O!) - O * \log(E)] - [O + \log(O!) - O * \log(O)] = O * \log({}^{O}\!/\!{}_{E}) - (O - E)$$

poisson association

As explained above, the correct sign has to be added, so the resulting POISSON ASSOCIATION measure becomes:

$$\text{Poisson Association} = \text{sign}(O - E) * (O * \log({}^{O}\!/\!{}_{E}) - (O - E))$$

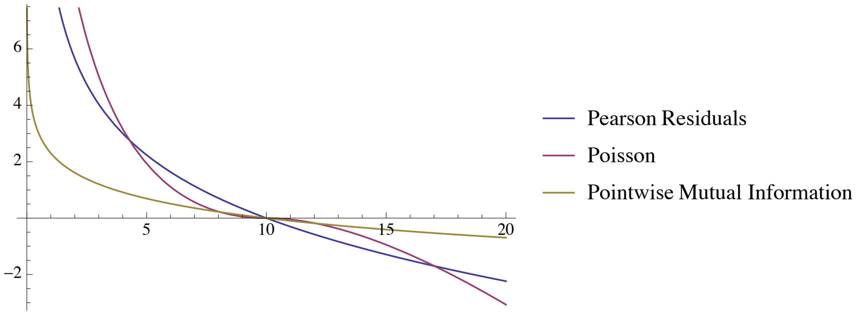

Figure 4: *Comparing three measures of association for an observed value of 10, and expected values between 0 and 20.*

Figure 4 shows a comparison of these three measures of association for an observed value of $O = 10$, and the expectation ranging between $E = 0$ and $E = 20$ on the x-axis. For all these measures, the result is positive when $O > E$ (left), negative when $O < E$ (right), and zero when $O = E$. Note that the Poisson measure shows the strongest differentiation, in that for expected values close to the observed value of 10, the Poisson measure is closest to zero, while for expected values far from the observed value of 10, the Poisson measure is further away from zero than the other measures.

9 Correction for Missing Data

A more intricate approach to estimate the expected values takes into account the distribution of missing information. With linguistic data, missing infor-

[9] The result looks a bit like a combination of the Pointwise Mutual Information (PMI) and the Pearson Residuals (PR), namely $O * \text{PMI} - \sqrt{E} * \text{PR}$.

mation is a substantial problem, possible strongly influencing any measure of association. Fortunately, it turns out to be relatively easy to add a correction for missing information. As proposed in Section 1, when there is missing data in a data table, this is best coded as zero in the data matrix A. The presence of missing data is then coded in a separate 'data availability' matrix D of the same size as A.

For a data table with nominal data (see Section 3) the matrix with available data D can be derived easily from the incidence matrix A and the accompanying index matrix X by taking the product of these two matrices: $D = A \cdot X^T$. Note that D will have the size of the original data table, but not the size of the incidence or index matrices.

9.1 Correcting the Matrix Product

availability product

Now, given two data matrices A and B and a 'massaged' similarity $A \cdot B^T$, and two data availability matrices D_A and D_B, then the AVAILABILITY PRODUCT $D_A \cdot D_B^T$ can be used to correct for data availability. Any weighting and normalisation that was applied to A and B should also be applied to D_A and D_B (but not centering!). Note that all matrices have to be weighted and normalised individually: it is not the same weighting or normalising matrices that can be reused! The corrected similarity is then simply:

$$\frac{A \cdot B^T}{D_A \cdot D_B^T}$$

hamming similarity

relative hamming similarity

For example, given a nominal incidence matrix A (with accompanying index matrix X), then a simple, uncorrected HAMMING SIMILARITY (i.e. just counting the number of similarities) is $A \cdot A^T$. A corrected RELATIVE HAMMING SIMILARITY (which is the same as Goebl's *relativer Identitätswert*, RIW) is then (finally leaving away all those central dots for the matrix product):

$$\frac{AA^T}{(AX^T)(AX^T)^T} = \frac{AA^T}{A\,(X^TX)\,A^T}$$

gewichteter identitätswert

Adding a column weight W, for example to emulate Goebl's GEWICHTETER IDENTITÄTSWERT (GIW), let's define $W = \text{diag}(1_{1N} \cdot A_{NP})^{-1/2}$. Remember that $W = W^T$ because W is diagonal, and note that $W \cdot W = W^2 = \text{diag}(1_{1N} \cdot A_{NP})^{-1}$. Then the corrected weighted similarity is:

$$\frac{(AW)(AW)^T}{(AX^TW)(AX^TW)^T} = \frac{A\,(W^2)\,A^T}{A\,(X^TW^2X)\,A^T}$$

However, note that Goebl uses a special correction, which is actually a combination of an unweighted correction for the differences (viz. $AX^TXA^T - AA^T$) and a weighted correction for the similarities (viz. $AWWA^T$), leading to the following formula:

$$GIW = \frac{AWWA^T}{AX^TXA^T - AA^T + AWWA^T} = \frac{A\,(W^2)\,A^T}{A\,(X^TX - I + W^2)\,A^T}$$

9.2 Correcting the Expectation

With a comparison of observed vs. expected matrices, it is the expectation matrix that will have to be corrected on the basis of the data availability. For numerical data, a corrected expectation matrix is calculated as follows. Note that this formula is not feasible for large sparse matrices. See the earlier discussion for methods to change this to sparse-friendly computations.

$$E = \left(A\,\frac{1_{PP}}{P}\,B\right) * \left(\frac{D_A\,D_B^T}{D_A\,\frac{1_{PP}}{P}\,D_A^T}\right) = \frac{A\,1_{PP}\,B^T}{D_A\,1_{PP}\,D_B^T}\left(D_A\,D_B^T\right)$$

Interestingly, the definition for the expectation matrix for nominal variables automatically incorporates the treatment of missing data.

10 Data Description

A specific data set will be conceived here as a collection of matrices. These matrices interlink different kinds of information. For example, there might be one matrix linking language (as rows) to language families (as columns). Another matrix might link languages (as rows) to linguistic characteristics (as columns), etcetera. To keep the matrices conformable, it is important that the identity and the order of the entities is kept constant, so, for example, exactly the same set of language in the same order occurs in both matrices.

The entities in the rows and the columns of the matrices fall into two crucially different kinds. In the comparison of languages it is important to clearly separate LANGUAGE-SPECIFIC ENTITIES from COMPARATIVE ENTITIES. The failure to clearly distinguish between these two kinds of information easily leads to confusion. For example, phonemes of a language are language-specific entities, i.e. the *phoneme* /a/ from language X is not the same thing as the phoneme /a/ from language Y. For comparative purposes these two language-specific phonemes might be equated as being examples of the same *phone* ‘a’. However, such a linking of language-specific entities from different languages to comparative entities is never easy, and always only approximately true.

language-specific entities

comparative entities

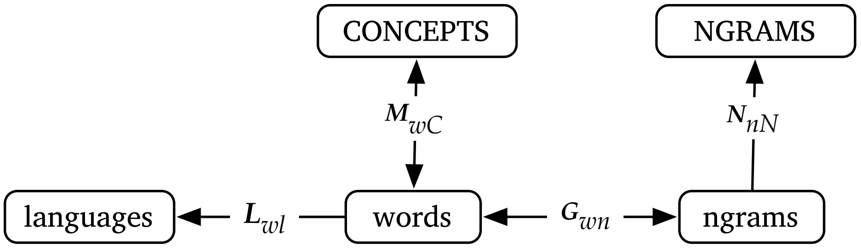

Figure 5: *Linking schema of a wordlist data set. Language-specific entities are written in lower case, and comparative entities are written with capitals. Comparative linking is shown with vertical connections. Bidirectional arrows indicate many-to-many linkings, unidirectional arrows indicate many-to-one linkings.*

comparative linking

Yet, such COMPARATIVE LINKING between language-specific entities and comparative entities is crucial to perform any kind of language comparison. Somewhere in a data set the language-specific entities of different languages will have to be connected to each other to allow for language comparison at all. Without any such comparative linking, language comparison is impossible. However, such linking is the most difficult, laborious, and error-prone part of any comparative project, so such linking should be reduced to a minimum of easy-to-perform and hopefully uncontroversial cross-linguistic equations. One of the goals of the current quantitative approach is to show that abstract high-level cross-linguistic comparison is possible on the basis of low-level comparative linking.

linking schema

Figure 5 shows an LINKING SCHEMA of a wordlist data set (e.g. a collection of Swadesh lists). Language specific entities are written in lowercase, while comparative entities are written with capitals. For such a schema, I propose to strictly display comparative entities at the top, and language-specific entities at the bottom. In this way, any comparative linking is always shown by vertical connections, while horizontal connections show language-specific connections. The example in Figure **??** links five different kinds of entities by four set of links, each of which is encoded as a matrix. These linking matrices are given a mnemonic abbreviation (M for ‘meanings’, L for ‘languages’, G for ‘graphemes’, and N for ‘ngrams’).

comparative matrix

For example, the COMPARATIVE MATRIX M links language-specific words to comparative concepts. The dimensions of this matrix is $w \times C$ (also note the capitalisation here), indicating that the words are listed as rows and the concepts are listed as columns. This linking has bidirectional arrows in the schema, indicating that the linking is many-to-many. A single concept is of course linked to many different words from different languages (and possibly also to multiple words from one language). However also one word can be linked to various concepts, i.e. one and the same word from a specific language can be linked to various comparative concepts. Note that in Swadesh lists this is uncommon, but in larger wordlist projects this occurs rather often.

language-specific matrix

A simpler matrix in this example is L, which links all the words in the data set to the language to which these words belong. This is a many-to-one linking (indicated by the unidirectional arrow), as each word only belongs to one language. Homographic words from different languages will of course have to be treated separately. A more interesting linking is the LANGUAGE-SPECIFIC MATRIX G. This matrix links each word to the ngrams of which this word consists. Crucially, these links are completely language-specific, i.e. the ngrams are separately specified for each language. Matrix G will thus be a block diagonal matrix, with one block for each language.

11 Wordlist Analysis

A collection of wordlists is analysed here as shown in the linking schema Figure 5. In this practical applications, whenever a matrix product is written, it will be assumed that there is some kind of 'massaging' necessary. However, at first, these extra operations will not be explicitly specified, and only in later steps the details will be discussed. Likewise, problems related to sparsity (or better 'density') of some of the following calculations will be ignored. So, be aware that just using the calculations below with large sparse datasets are not computationally optimised yet.

11.1 Basic Wordlist Analysis

polysemy

For any comparative conclusions, the comparative linking M_{wC} is crucial. The simplest conclusion of interest is the similarity between concepts, based on their encoding by words. The underlying rationale of this similarity is that sometimes one word is linked to two different meanings, i.e. cases of homophony or POLYSEMY. The following formula thus counts how often two meanings are encoded by the same word across all languages.

$$S_{CC} = M_{wC}^{T} \cdot M_{wC}$$

partial homophony

A variation on this theme is to establish the similarity between concepts based on partial similarity of words. The following formula counts for all pairs of meanings how often they share a language-specific ngram, i.e. the distribution of PARTIAL HOMOPHONY.

$$S_{CC} = (M_{wC}^T G_{wn}) \cdot (M_{wC}^T G_{wn})^T = M_{wC}^T G_{wn} G_{wn}^T M_{wC}$$

Reversing this formula results in another interesting result. The distribution of language-specific ngrams over concepts $G_{wn}^T M_{wC}$ can be used to investigate the similarity between ngrams. When two ngrams from different languages show a similar distribution across concepts, this is an indication of a regular SOUND CORRESPONDENCE. Note that this formula calculates the similarity between all pairs of ngrams between all pairs of languages. There is not yet a selection of 'interesting' sound correspondences.

sound correspondence

$$S_{nn} = (M_{wC}^T G_{wn})^T \cdot (M_{wC}^T G_{wn}) = G_{wn}^T M_{wC} M_{wC}^T G_{wn}$$

The similarity between words from different languages is a more complex issue. For a comparison of words across language we need some mapping between graphemes. This could be derived from the previous S_{nn}, but that involves a further step of alignment that will be discussed later. An easier calculation of WORD SIMILARITY involves another comparative linking N_{nN}, which connects the language-specific ngrams to some kind of cross-linguistically harmonised ngrams (e.g. IPA-based, or based on sound classes). Note that this formula computes the similarity between all pairs of words, irrespective of the distribution over concepts.

word similarity

$$S_{ww} = (G_{wn} N_{nN}) \cdot (G_{wn} N_{nN})^T = G_{wn} N_{nN} N_{nN}^T G_{wn}^T$$

Instead of looking at the similarity between words, it is probably even more interesting to look at the similarity between whole languages. One simple way to compare languages is to look a the GLOBAL NGRAM similarity. In this comparison the distribution of words over concepts is ignored, and a language is simply interpreted as an unordered set of words. For each language, the global distribution of ngrams is computed by summing all ngrams per language $L_{wl}^T G_{wn} N_{nN}$ and compare these (note that normalisation and sparsity management become crucial here at the latest).

global ngram

$$S_{ll} = (L_{wl}^T G_{wn} N_{nN}) \cdot (L_{wl}^T G_{wn} N_{nN})^T = L_{wl}^T G_{wn} N_{nN} N_{nN}^T G_{wn}^T L_{wl}$$

parallel ngram

More telling is a so-called PARALLEL NGRAM comparison. In such a comparison, only the words that belong to the same concept are compared, and these comparison are summed up to compare two languages. The result is strongly correlated with an aggregated Levenshtein distance between two word lists, but it is computationally much easier and quicker. Instead of looking at all word comparisons S_{ww}, we are now only interested in those word comparisons within one concept. Logically, this can be computed by taking the following entry-wise product $S_{ww} * (M_{wC} M_{wC}^T)$, but note that this needlessly computes many word similarities that are subsequently ignored. A better option is to use a row-based Khatri-Rao product $(G_{wn} N_{nN}) \odot M_{wC}$, which results in a matrix of size $w \times (N * C)$. This can be used to compute a language similarity as follows (note that in this formula all four linking matrices are used).

$$S_{ll} = (L_{wl}^T \cdot ((G_{wn} \cdot N_{nN}) \odot M_{wC})) \cdot (L_{wl}^T \cdot ((G_{wn} \cdot N_{nN}) \odot M_{wC}))^T$$

Note that $(A \odot B) \cdot (A \odot B)^T = (A \cdot A^T) * (B \cdot B^T)$ so we can indeed easily show that using $S_{ww} * (M_{wC} M_{wC}^T)$ is equivalent to using $(G_{wn} N_{nN}) \odot M_{wC}$:

$$\begin{aligned} S_{ll} = (L_{wl}^T \cdot ((G_{wn} \cdot N_{nN}) \odot M_{wC})) \cdot (L_{wl}^T \cdot ((G_{wn} \cdot N_{nN}) \odot M_{wC}))^T = \\ L_{wl}^T \cdot ((G_{wn} \cdot N_{nN}) \odot M_{wC}) \cdot ((G_{wn} \cdot N_{nN}) \odot M_{wC})^T \cdot L_{wl} = \\ L_{wl}^T \cdot ((G_{wn} \cdot N_{nN}) \cdot (G_{wn} \cdot N_{nN})^T * M_{wC} \cdot M_{wC}^T) \cdot L_{wl} = \\ L_{wl}^T \cdot (S_{ww} * M_{wC} \cdot M_{wC}^T) \cdot L_{wl} \end{aligned}$$

11.2 Normalisation

[*details to be added*]

11.3 Extended Wordlist Analysis

Figure 6 shows an extended version of a linking schema for wordlist data. Instead of pre-established ngram parsing, this schema uses a combination of ‘segments’ and ‘graphemes’ from which the ngrams can be derived. Also note that there is a unidirectional arrow between words and concepts M_{wC}. This means that all words that occur with more than one concept are listed twice. In which cases complete identity is present can be derived from the ‘segments’ and ‘graphemes’ linking.

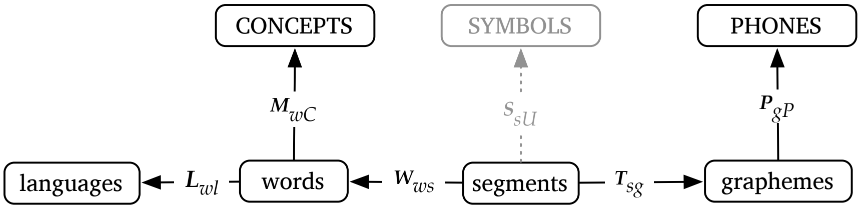

Figure 6: *Extended wordlist linking schema, including graphemic parsing. Ngrams can be derived from this.*

The basic idea of the ‘segments’ entities is to parse all words into segments and interpret all these units as one long vector. For example, the three words *this is that* might for example be parsed as 14 segments (including boundaries) as follows: [*# th i s # i s # th a t #*]. The matrix W linking these 14 segments (as columns) to 3 words (as rows) would then look as shown below. Note that the order of the segments is the same as in the original strings. In this way we will be be able to extract ordering information. Likewise, the matrix T links these 14 (ordered) segments to the 6 different graphemes that occur (viz. [*# th i s a t*]). This is an example of a TYPE-TOKEN MATRIX (see Section 5.2). The order of these graphemes in the columns is not important. Only the order of the entities 'segments' in the rows is important.

$$W_{ws} = \left[\begin{array}{r|c|ccc|c|cc|c|ccc|c}
 & \# & th & i & s & \# & i & s & \# & th & a & t & \# \\ \hline
\textit{this} & \cdot & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\textit{is} & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\textit{that} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot
\end{array}\right]$$

$$L_{sg} = \left[\begin{array}{r|cccccc}
 & \# & th & i & s & a & t \\ \hline
\# & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \hline
th & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
i & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
s & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \hline
\# & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \hline
i & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
s & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \hline
\# & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \hline
th & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
a & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
t & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \hline
\# & 1 & \cdot & \cdot & \cdot & \cdot & \cdot
\end{array}\right]$$

The graphemes are language-specific, implying that the type-token matrix T consists of language-specific links. In practice, the linking in T will probably most easily be constructed by using a linkage S to 'universal' symbols in the data (i.e. all identical unicode symbols are treated as being the same, even then they come from different languages). The language specific linking T can be derived from the 'universal' linking using the language matrix L using the Khatri-Rao Product $\odot$ (see Section 2.3) as follows:

$$T_{sg} = S_{sU} \odot \left(W_{ws}^{T} \cdot L_{wl}\right)$$

Now, the product of these two matrices, $W_{ws} \cdot T_{sg}$, gives the matrix G_{wg} from Figure 5 linking words to language-specific unigrams. When two rows in this G_{wg} matrix are completely identical, then the words are identical.

$$G_{wg} = W_{ws} \cdot T_{sg} = \left[\begin{array}{r|cccccc} & \# & th & i & s & a & t \\ \hline \textit{this} & \cdot & 1 & 1 & 1 & \cdot & \cdot \\ \textit{is} & \cdot & \cdot & 1 & 1 & \cdot & \cdot \\ \textit{that} & \cdot & 1 & \cdot & \cdot & 1 & 1 \end{array} \right]$$

More interestingly, if we make a shifted variant of T by removing the first row, and adding an extra row with zeros at the bottom, i.e by taking $U \cdot T$ (see Section 6 on the upper shift matrix U), then we can derive the language-specific linking of words to bigrams using a Khatri-Rao Product as follows:

$$G_{wg^2} = W_{ws} \left(T_{sg} \odot U_{ss} T_{sg} \right)$$

The language-specific similarity between graphemes based on the neighbouring frequency can be computed as follows (note that the product indicated by the middle dot has to be normalised for frequency):

$$S_{gg} = \left(T_{sg}^{T} U_{ss} T_{sg} \right) \cdot \left(T_{sg}^{T} U_{ss} T_{sg} \right)^{T}$$

Note that these similarities are purely language-specific, so the resulting S_{gg} matrix is a block diagonal matrix, with each block representing the similarities within one language. These within-language similarities should be a good predictor for a vowel-consonant differentiation, and maybe also for other sound classes.

12 Notes

"transition matrix" can either mean substitution matrix for alignment or stochastic matrix for Markov chains. In both cases, the identity matrix is the simplest model: every substitution between state a to a gets 1, every substitution between two different states gets 0. As a stochastic matrix, the identity matrix represents stasis.

Substitution matrices use often pointwise mutual information (called 'log-odds scores' in bioinformatics). Stochastic matrices use matrices P with probabilities for transitions, in which each row and/or column adds up to zero, and the stochastic matrix then is $I - P$.

(Note that this is only formulated for the case that the rows and columns consist of the same entities. The situation with different entities is slightly different.)

(Note that this is only formulated for nominal variables. continuous variables will need more complex transition specifications.)