A fast general purpose procedure to calculate

Inter-rater Agreement

for coreference annotation

Erwin R. Komen
Radboud University, Nijmegen, Netherlands
June 2009

1. Introduction

When the inter-rater agreement between two persons, who have annotated a text with coreference information, is to be measured, then a procedure is necessary that can work with different categories. The raters may have disagreed on the coreference target points, so when the inter-rater agreement with regard to these target points is measured, and when the categories to be considered are the Id's of the target points within a text, then these two raters will not have identical categories.

Inter-rater agreement in general can be measured using Cohen's kappa (Cohen, 1960). Formula (1) gives a definition of Cohen's kappa.

(1) $$\kappa = \frac{P_0 - P_c}{1 - P_c}$$

P_0 = proportion agreement observed: $\frac{\#\,agreement}{\#\,possibilities}$

P_c = Proportion agreement based on chance

A program like SPSS fails to calculate Cohen's kappa, when the categories chosen by two raters do not completely overlap. If one wants to calculate Cohen's kappa, then one could either resort to calculation by hand, or one could use an internet utility called RecalFront (Freelon, 2008). The URL provided in the reference's section 4 points to the program RecalFront, where one can upload the information in the form of a comma separated file (CSV) and receive a report on the inter rater agreement. The author of RecalFront has not provided the mathematical basis behind his calculation of Cohen's kappa, but his implementation does allow for raters to have no complete overlap between categories chosen.

This paper reports on a procedure and an implementation to calculate Cohen's kappa, which does not require raters to have identical categories.

2. Calculating Kappa

2.1. Symmetrical calculations

Suppose someone wants to determine whether sentences from a text contain a topic (1) or not (0). In order to test the inter-rater agreement two people, say John and Mary, rate the same 10 sentences of a text. Their judgments are listed in (2). They agree on their judgments 6 times—in sentences 1, 2, 4, 5, 7, and 10. So their agreement is 60%. This is P_0 from formula (1).

(2) Sentence *x* contains a topic (1) or not (0).

Sentence	1	2	3	4	5	6	7	8	9	10
John	*0*	*1*	*1*	*0*	*0*	*1*	*0*	*1*	*0*	*0*
Mary	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*

In order to calculate P_c, which is the "chance agreement", we make the symmetrical matrix in (3), where the rows are formed by evaluating how far John agreed with Mary, and the columns by evaluating how far Mary agreed with John. The cell with column "John: 0" and row "Mary: 0" contains the number of times John agreed with Mary that a sentence did not contain a topic (i.e. 5 times they both chose "0"). The cell to the right of it contains the amount of times John did not agree with Mary's judgment that the sentence contained no topic (3 times). If John and Mary would have been in total agreement, the matrix would have held zero values except for those on the diagonal.

(3) Symmetrical agreement matrix for John and Mary's judgments

	John: 0	John: 1	
Mary: 0	*5*	*3*	8
Mary: 1	*1*	*1*	2
	6	4	10

The chance agreement P_c is now calculated by the sum of the product of the rows and columns along the diagonal, divided by the total number of possibilities. The formula for calculating P_c in a symmetrical matrix is given in (4). For every point $M_{i,i}$ on the diagonal of matrix M, the total of the values in row *i* divided by *n* is multiplied by the total of the values of column *i* divided by the total number of judgments *n*. The value for P_c thus becomes (8*6 + 2*4)/100 = 0,56. Since the agreement was 0,6, the value for kappa now becomes (0,6-0,56)/(1-0,56) = 0,09.

(4) $$P_c = \sum_{i=1}^{n} \left(\frac{\sum_{j=1}^{n} M_{i,j}}{n} * \frac{\sum_{j=1}^{n} M_{j,i}}{n} \right)$$

Crucial for the above calculation is the fact that both John and Mary used the same judgments—the values "0" and "1". If John would have used "0", "1", and "2" (where "2" could have meant "there may be a topic, but I don't know"), while Mary had stuck to "0" and "1", then the chance agreement matrix M would have been asymmetrical, as illustrated in (5). Such a matrix contains 3 columns for John, but only 2 rows for Mary. The formula given in (4) now becomes useless, since it assumes that the number of rows equals the number of columns, equals *n*.

(5) Asymmetrical agreement matrix for John and Mary's judgments

	John: 0	John: 1	John: 2	
Mary: 0	*5*	*1*	*1*	7
Mary: 1	*1*	*1*	*1*	3
	6	2	2	10

2.2. Restoring symmetry

For a more general calculation we at least have to turn back to the original list of judgments shown in table (2). Let us assume again that John uses judgment "2" twice, as in X.

(6) Sentence *x* contains no topic (0), a topic (1) or undeterminable (2).

Sentence	1	2	3	4	5	6	7	8	9	10
John	*0*	*1*	*2*	*0*	*0*	*1*	*0*	*2*	*0*	*0*
Mary	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*

Calculating P_0, the percentage agreement, stays as is, since we only need to calculate the number of times John agrees with Mary, and that hasn't changed.

The chance agreement value P_c would need a symmetrical matrix M. One might get the idea to supply zeros for Mary judging "2". We would then get the restored symmetrical matrix, as shown in (7). Calculation of P_c would then yield (8*6+2*2+0*2)/100 = 0,52. The value for kappa would then become (0,6-0,52)/(1-0,52) = 0,17. This value would suggest that the inter-rater agreement between John and Mary is better, while our intuition would say that this is not the case. However, this is something intrinsically of the kappa value, but the calculation is correct.

(7) Restored symmetrical agreement matrix for John and Mary's judgments

	John: 0	John: 1	John: 2	
Mary: 0	*5*	*1*	*2*	8
Mary: 1	*1*	*1*	*0*	2
Mary: 2	*0*	*0*	*0*	0
	6	*2*	*2*	10

A drawback of the method above is that we would be forced to always come up with symmetrical matrices for our judgments. That is to say, for a program like SPSS, which is able to calculate the kappa value, we would have to specify that Mary has used the value "2" zero times. This can be circumvented by building our own procedure, as will be explained in the next section.

2.3. A general approach

A more general approach to calculating P_c would be to use a method that is not based on working with a matrix. Taking the values in (6) as a starting point, the following procedure could be followed:

a) Take the percentage of sentences where John chose "0" and multiply this with the percentage of sentences where Mary chose "0".

b) Do the same for the sentences where each chose "1".

c) Take the percentage of sentences where John chose "2" (this is 20%) and multiply this with the percentage of sentences where Mary chose "2" (this is 0%). This yields 0.

d) P_c is the sum of the products in a, b and c: (6*8 + 2*2 + 2*0)/100 = 0,52.

Now suppose John and Mary are not judging sentences as to their topicality, but they are actually providing coreference information—they are laying relations from an NP to a preceding IP or NP. They use labels like "Inferred", "Identity", "CrossSpeech" and "Assumed" for their coreference relations. Ten of their results are shown in (8).

(8) NPs coreferring with type 0 'nothing', 1 'Identity", 2 'Inferred', 3 'Assumed', and 4 'CrossSpeech'.

NP Id	21	23	27	34	50	62	78	82	84	90
John	*0*	*1*	*3*	*0*	*0*	*1*	*0*	*4*	*0*	*0*
Mary	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*

The agreement value is again 0,6, and our procedure would yield a P_c of (6*8 + 2*2 + 1*0 + 1*0)/100 = 0,52, which is the same as the one calculated previously.

John and Mary have not only indicated what the *types* of the coreference relations are, but they have also connected the NPs with antecedents. The ids of the antecedents are shown in table (9). Note that if we were to use the matrix method, we would have to build a sparsely populated 62*62 matrix, which would not be very effective. Instead, our procedure again yields a P_c of 0,52, since we only need to calculate the number of times the actually occurring values *0, 13, 23,* and *62* are used by John and Mary.

(9) Id's of the antecedents to which the NPs are connected.

NP Id	21	23	27	34	50	62	78	82	84	90
John	*0*	*13*	*23*	*0*	*0*	*13*	*0*	*62*	*0*	*0*
Mary	*0*	*13*	*0*	*0*	*0*	*0*	*0*	*0*	*13*	*0*

It seems we now have found a robust procedure. But how can this procedure be described in a formula, and, perhaps more importantly, how can it be calculated?

The method I use constitutes of two cycles. The first cycle visits all n sentences (or measuring points) and derives the m (where $m < n$) different category values used by the two raters. The second cycle visits all m different values and sums the frequency of occurrence for rater 1 multiplied by the frequency of occurrence for rater 2. The formula for this method is shown in (10). The arrays R1 and R2 coincide with the rows for John and Mary in (9), while the array Value contains all m different judgments (e.g. target Id's or coreferencing types in the examples above) used by both raters.

(10) $$P_c = \sum_{i=1}^{m} \left(\frac{\sum_{j=1}^{n} R1_j = Value_i}{n} * \frac{\sum_{j=1}^{n} R2_j = Value_i}{n} \right)$$

2.4. Implementation

The implementation of the two-step-method introduced in 2.3 follows the procedure laid out in the previous paragraph, including the formula provided in (10). The function `CohensKappa()` in section 5, the appendix, provides a Visual Basic realization of the implementation. The first cycle, as shown in (11), fills the array V with the values used by rater 1 and/or rater 2 (these values are in column `intCol1` and `intCol2` of the string array `arData`). Each element in array V has a field with the actual value, a field with the number of times this value is used by rater #1 and a field with the frequency of occurrence for rater #2. The function `IncrVal()` finds the category value `intVal` and increments the frequency

(11) First cycle of the kappa calculation

```
' Visit all points
For intI = 0 To intN
  ' Get the data for this line
  arLine = Split(arData(intI), ";")
  ' Get the values for rater #1 and rater #2
  intVal1 = CInt(arLine(intCol1)) : intVal2 = CInt(arLine(intCol2))
  ' Put these values in one array
  IncrVal(arVal, intVnum, intVal1, 1)
  IncrVal(arVal, intVnum, intVal2, 2)
  ' Keep track of agreement
  If (intVal1 = intVal2) Then intM += 1
Next intI
' Calculate P0
dblP_0 = intM / (intN + 1)
```

The first cycle furthermore calculates the agreement percentage by keeping track of how many times there is full agreement between rater 1 and rater 2.

The second cycle, as shown in X, visits all the values used by rater 1 and/or 2 that are stored in the array V. It keeps track of the sum of the frequency of occurrence of each value for rater #1 multiplied by that of rater #2. This sum is later on divided by the square of the total number of measuring points *n*, in accordance with the formula above, yielding P_c.

(12) Second cycle of the kappa calculation

```
' Visit all actual values stored in array [arVal]
For intI = 1 To intVnum
  ' Keep track of the sum
  intM += arVal(intI).Freq1 * arVal(intI).Freq2
Next intI
' Calculate total Pc
dblP_c = intM / ((intN + 1) * (intN + 1))
```

The final step is to calculate the actual kappa using formula (1). The code behind this step is shown in (13).

(13) Calculation of the kappa value on the basis of P_0 and P_c

```
' Pass back Kappa and Agreement
dblKappa = (dblP_0 - dblP_c) / (1 - dblP_c)
dblAgr = dblP_0
```

3. Conclusions

Cohen's kappa provides a measure that can be used to determine inter-rater agreement between different persons having annotated texts with coreference information. The set of target Id's where the anaphoric references provided by rater #1 point to might not completely overlap with the set of target Id's provided by rater #2. Such rating results cannot be processed with a program like SPSS. An internet tool like RecalFront provides the researcher with the possibility to determine Cohen's kappa for ratings with non-identical category sets. However, the procedure followed by such a program is not available. This current paper proposes a fast two-stage algorithm to calculate Cohen's kappa for category sets that do not completely overlap. An implementation of this algorithm in Visual Basic is provided.

4. References

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement 20*(1), 37–46.

Freelon, D. (2008). ReCal: reliability calculation for the masses [Electronic Version] from http://dfreelon.org/utils/recalfront/.

5. Appendix: the code

This appendix provides a module of Visual Basic code that can be used to calculate Cohen's kappa. The main function is called `CohensKappa`, and it should be called with the parameters indicated.

```
Module modStat
  ' ================================================================
  ' Name :  modStat
  ' Goal :  Statistical functions. In particular: asymmetric Cohen's Kappa
  ' History:
  ' 24-06-2009  ERK Created
  ' 17-11-2009  ERK Changed method slightly
  ' ================================================================
  ' ========================== LOCAL TYPES ==============================
  Private Structure ValFreq
    Dim Value As Integer  ' The value
    Dim Freq1 As Integer  ' The frequency of rater #1 for this value
    Dim Freq2 As Integer  ' The frequency of rater #2 for this value
  End Structure
  ' ========================== LOCAL VARIABLES ========================
  Dim arVal() As ValFreq  ' Values for rater #1 and rater #2
  Dim intVnum As Integer  ' Number of values in [arVal]
  ' ----------------------------------------------------------------------
  ' Name:   CohensKappa
  ' Goal:   Calculate Cohen's Kappa for non-symmetric matrices
  ' Input:  arData() is a string array where each line contains integer
  '           data separated by ";" signs (i.e. the content of a CSV file)
  '         intCol1 and intCol2 define which columns in [arData] belong to
  '           rater #1 and rater #2
  ' Return: dblKappa is Cohen's kappa (0 ... 1)
  '         dblAgr is the agreement (0 ... 1)
  ' History:
  ' 24-06-2009  ERK Created using two arrays [arV1] and [arV2]
  ' 17-11-2009  ERK Adapted for faster method using only one [arVal]
  ' ----------------------------------------------------------------------
  Public Function CohensKappa(ByRef arData() As String,
      ByVal intCol1 As Integer, _
      ByVal intCol2 As Integer, ByRef dblKappa As Double,
      ByRef dblAgr As Double) As Boolean
    Dim arLine() As String  ' The values of one line
    Dim intI As Integer     ' Counter
    Dim intN As Integer     ' Number of data points
    Dim intM As Integer     ' Intermediate number
    Dim intVal1 As Integer  ' Value for rater #1
    Dim intVal2 As Integer  ' Value for rater #2
    Dim dblP_0 As Double    ' P0 from the Kappa formula = % agreement
    Dim dblP_c As Double    ' Pc from the Kappa formula: k=(P0-Pc)/(1-Pc)

    ' Initialise the Value sets
    intN = UBound(arData)
```

```
  ' Adapt [intN] for empty elements
  While (arData(intN) = "") OrElse (InStr(arData(intN), ";") = 0)
    intN -= 1
  End While
  ' Initialise valFreq sets
  ValFreqClear(intN)
  intM = 0
  ' Visit all points
  For intI = 0 To intN
    ' Get the data for this line
    arLine = Split(arData(intI), ";")
    ' Get the values for rater #1 and rater #2
    intVal1 = CInt(arLine(intCol1)) : intVal2 = CInt(arLine(intCol2))
    ' Put these values in one array
    IncrVal(arVal, intVnum, intVal1, 1)
    IncrVal(arVal, intVnum, intVal2, 2)
    ' Keep track of agreement
    If (intVal1 = intVal2) Then intM += 1
  Next intI
  ' Calculate P0
  dblP_0 = intM / (intN + 1)
  ' Calculate Sum for Pc
  intM = 0
  ' Visit all actual values stored in array [arVal]
  For intI = 1 To intVnum
    ' Keep track of the sum
    intM += arVal(intI).Freq1 * arVal(intI).Freq2
  Next intI
  ' Calculate total Pc
  dblP_c = intM / ((intN + 1) * (intN + 1))
  ' Pass back Kappa and Agreement
  dblKappa = (dblP_0 - dblP_c) / (1 - dblP_c)
  dblAgr = dblP_0
  ' Return success
  CohensKappa = True
End Function
' ------------------------------------------------------------------------
' Name:   ValFreqClear
' Goal:   Clear and initialise the ValFreq arrays
' History:
' 24-06-2009  ERK Created
' ------------------------------------------------------------------------
Private Sub ValFreqClear(ByVal intN As Integer)
  Dim intI As Integer     ' Counter

  ' Set size
  ReDim arVal(0 To intN)
  ' Visit all potential members (intVnum will be smaller than intN)
  For intI = 0 To intN
    ' Access this member
    With arVal(intI)
      ' Clear the members
      .Freq1 = 0    ' Default frequency is zero
      .Freq2 = 0    ' Default frequency is zero
      .Value = -1   ' This value should be overridden by one of 0
                    ' or higher
    End With
  Next intI
  ' Reset the size to ZERO
  intVnum = 0
End Sub
```

```
  ' ------------------------------------------------------------------------
  ' Name:   IncrVal
  ' Goal:   Increment the frequency for the value [intVal] in array [arV]
  '         Array [arV] right now has [intNum] members (from 1 to intNum)
  '         The number of the rater is in [intRater]
  ' History:
  ' 24-06-2009  ERK Created
  ' 17-11-2009  ERK Adapted for Freq1/Freq2 by adding [intRater]
  ' ------------------------------------------------------------------------
  Private Sub IncrVal(ByRef arV() As ValFreq, ByRef intVnum As Integer, _
                      ByVal intVal As Integer, _
                      ByVal intRater As Integer)
    Dim intI As Integer   ' Counter

    ' Check all members of [arV]
    For intI = 0 To intVnum
      ' Does this member contain the value?
      If (arV(intI).Value = intVal) Then
        ' Which rater?
        If (intRater = 1) Then
          ' Increment the frequency of it
          arV(intI).Freq1 += 1
        Else
          ' Increment the frequency of it
          arV(intI).Freq2 += 1
        End If
        ' Exit
        Exit Sub
      End If
    Next intI
    ' Member was not found, so add to [arV]
    intVnum += 1
    With arV(intVnum)
      .Value = intVal
      ' Which rater?
      If (intRater = 1) Then
        .Freq1 = 1
      Else
        .Freq2 = 1
      End If
    End With
  End Sub
End Module
```