The Rolling Soundex Formula: A Revision Proposal

Filipp Krasovsky

Economics, BA - University of California, Los Angeles '19

June 8, 2019

Introduction

The Soundex Formula is a useful tool in data science and string analysis for finding the similarity between two words, sentences, or passages. In a practical, enterprise-level application, it can often amplify the effectiveness of auditing procedures in an Excel or MS Office environment where two tables need to be compared rapidly with automation. The soundex formula exists for a variety of languages such as SQL, R, Ruby, etc. but despite its robust usage, the Visual Basic for Applications environment does not come with standard-availability string processing that functions in a nonbinary setting. This report seeks to not only optimize the usage of said algorithm for specialized auditing environments, but to raise visibility for the work of VBA developers who have come together to make such functionality available.

During the time of publication, the author uses the Soundex function to automate the auditing process at the Debt Assistance Program, namely in the scope of comparing correspondence from collection agencies to the data we have on file for our clientelle; the function is used to calculate the accuracy of the alleged balances, names, issuing bank names, and other relevant information to provide support for disputes with 3rd party credit collection agencies later down the line.

Motivation

Although it broadly outforms a number of built-in Excel functions like Instr() and StrComp(), the Soundex formula provides counterintuitive similarity results that, while technically correct, create operational roadblocks in auditing modules with a tolerance for dissimilarity. some of these examples will be reviewed below; as a result, the rolling Soundex formula has been proposed as a combination of the non-binary nature of the original Soundex formula and the indexing strength of VBA's Instr() function.

To clarify, the strength of the Soundex Formula comes from the ability to provide a figure suggesting approximation between one string and another, rather than the binary nature of StrComp() or Instr(), which both search the exact string value. The weakness of the Soundex function comes from the fact that it analyzes the similarity of both strings fully, rather than subsections thereof, which often provide more interpretable results that have practical utility. Namely, the similarity decreases between two strings as the number of characters in either one increases. Essentially, the purpose of the rolling approach is to determine if one string roughly contains the "essence" of the other while offsetting any random noise in the data.

Clarification: all functions constructed have been optimized for use in MS Office Excel's VBA IDE. Users interested in experimenting with the code themselves may access this IDE by enabling the developer ribbon in Excel and clicking "view code" or viewing the link at the bottom of this report for more help.

Initial Soundex Formula

VB Source Code and Analysis

The Soundex Formula takes two arguments, in this particular instance, two strings/character combinations and returns a decimal value ranging from 0 to 1 that describes how similar they are. Original source data comes from VBForums.com user *si_the_geek*, published on November 24th, 2006, which contains the main similarity function as well as a separate, recursive component:

```
Public Function Similarity(ByVal String1 As String, _
    ByVal String2 As String, _
    Optional ByRef RetMatch As String, _
    Optional min_match = 1) As Single
    Dim b1() As Byte, b2() As Byte
    Dim lngLen1 As Long, lngLen2 As Long
    Dim lngResult As Long

    If UCase(String1) = UCase(String2) Then
        Similarity = 1
    Else:
        lngLen1 = Len(String1)
        lngLen2 = Len(String2)
        If (lngLen1 = 0) Or (lngLen2 = 0) Then
            Similarity = 0
        Else:
            b1() = StrConv(UCase(String1), vbFromUnicode)
            b2() = StrConv(UCase(String2), vbFromUnicode)
            lngResult = Similarity_sub(0, lngLen1 - 1, _
            0, lngLen2 - 1, _
            b1, b2, _
            String1, _
            RetMatch, _
            min_match)
            Erase b1
            Erase b2
            If lngLen1 >= lngLen2 Then
                Similarity = lngResult / lngLen1
            Else
                Similarity = lngResult / lngLen2
            End If
        End If
    End If

End Function
```

```
        Private Function Similarity_sub(ByVal start1 As Long, ByVal end1 As Long, _
    ByVal start2 As Long, ByVal end2 As Long, _
    ByRef b1() As Byte, ByRef b2() As Byte, _
    ByVal FirstString As String, _
    ByRef RetMatch As String, _
    ByVal min_match As Long, _
    Optional recur_level As Integer = 0) As Long
    '* CALLED BY: Similarity *(RECURSIVE)

                Dim lngCurr1 As Long, lngCurr2 As Long
                Dim lngMatchAt1 As Long, lngMatchAt2 As Long
                Dim i As Long
                Dim lngLongestMatch As Long, lngLocalLongestMatch As Long
                Dim strRetMatch1 As String, strRetMatch2 As String

                If (start1 > end1) Or (start1 < 0) Or (end1 - start1 + 1 < min_match) _
                Or (start2 > end2) Or (start2 < 0) Or (end2 - start2 + 1 < min_match) Then
                        Exit Function '(exit if start/end is out of string, or length is too short)
                End If

                For lngCurr1 = start1 To end1
                        For lngCurr2 = start2 To end2
                                i = 0
                                Do Until b1(lngCurr1 + i) <> b2(lngCurr2 + i)
                                        i = i + 1
                                        If i > lngLongestMatch Then
                                                lngMatchAt1 = lngCurr1
                                                lngMatchAt2 = lngCurr2
                                                lngLongestMatch = i
                                        End If
                                        If (lngCurr1 + i) > end1 Or (lngCurr2 + i) > end2 Then Exit Do
                                Loop
                        Next lngCurr2
                Next lngCurr1

                If lngLongestMatch < min_match Then Exit Function

                lngLocalLongestMatch = lngLongestMatch
                RetMatch = ""

                lngLongestMatch = lngLongestMatch _
                + Similarity_sub(start1, lngMatchAt1 - 1, _
                start2, lngMatchAt2 - 1, _
                b1, b2, _
                FirstString, _
                strRetMatch1, _
                min_match, _
                recur_level + 1)
                If strRetMatch1 <> "" Then
                        RetMatch = RetMatch & strRetMatch1 & "*"
                Else
                        RetMatch = RetMatch & IIf(recur_level = 0 _
                        And lngLocalLongestMatch > 0 _
                        And (lngMatchAt1 > 1 Or lngMatchAt2 > 1) _
                        , "*", "")
                End If


                RetMatch = RetMatch & Mid$(FirstString, lngMatchAt1 + 1, lngLocalLongestMatch)


                lngLongestMatch = lngLongestMatch _
                + Similarity_sub(lngMatchAt1 + lngLocalLongestMatch, end1, _
                lngMatchAt2 + lngLocalLongestMatch, end2, _
                b1, b2, _
                FirstString, _
                strRetMatch2, _
                min_match, _
                recur_level + 1)

                If strRetMatch2 <> "" Then
                        RetMatch = RetMatch & "*" & strRetMatch2
                Else
                        RetMatch = RetMatch & IIf(recur_level = 0 _
                        And lngLocalLongestMatch > 0 _
                        And ((lngMatchAt1 + lngLocalLongestMatch < end1) _
                        Or (lngMatchAt2 + lngLocalLongestMatch < end2)) _
                        , "*", "")
                End If

                Similarity_sub = lngLongestMatch

        End Function
```

Without intense analysis of the source code, we can describe the steps of the function as taking two string arguments and determining how many characters are positionally and numerically identical to another another. The returned value is the portion of characters in the longer string that matches with those in the shorter. If the strings are the same length, either string is selected for this metric. The returned percentage value provides a more dynamic auditing application.

Results from standard Soundex Function [Public Function Similarity ()]

String 1	String 2	Similarity
Credit One	Credig Onn	0.8
Michael Welseley	Ashton Welseley	0.625
120 Corporate	5620 Southwyck	0.285
$5,622	$5,622	1.0
Credit One Bank	Credit One Bank, USA, NA	0.625
Finance Credit Bank, USA	Credit One Bank, USA, NA	0.66
Series A	Series A Type	0.615
Series A	Series A Type Document	0.36

While the function provides a robust analysis of similarity, it fails to capture elements that are intuitively in line with proper auditing expectations. For instance, in an auditing module with a similarity tolerance of 70%, the second-last entry would be considered an incorrect creditor name for a client. However, by all accounts, we can qualitatively assess that Credit One Bank is, indeed, a proper name for the issuing bank, as the full name of the legal entity is not always required for correspondence in debt validation. Furthermore, it's notable that in the two "Series A" comparisons, the accuracy decreases as the second string gets larger, although the interpretability of the comparison is essentially the same; Series A is an adequate entry-value in a practical setting, but a module with low tolerance for dissimilarity would not admit it as such. Consequently, while the similarity function provides us with more insight on the accuracy of our entry than the Instr() or StrComp() functions, it still remains insufficient in edge cases where data entry does not utilize the full legal name of an issuing bank or other corporate entity. We could lower the tolerance for dissimilarity in an auditing module to allow for 60% similarity, but that would lead us to encounter accuracy loss, as demonstrated by the very last entry, which is technically more similar to Credit One Bank's full name than Credit One Bank, but this is clearly misleading.

Proposal: The "Rolling" Soundex Function

In order to offset and eliminate the tradeoff between increasing dissimilarity tolerance and getting more, but less accurate results, the Rolling Soundex Function builds on the utility of "segmenting" the larger of two strings into cross-sections, the length of which is equal to the length of the shorter string. Simply put, if two strings A and B with character sets [1..Na] and [1...Nb], respectively, are assessed, this proposed function would examine cross sections of B as follows:

`Where B=>[1,2,3...,Nb] and A=>[1,2,3,...,Na] and` **`Nb > Na`**

Iteration	Cross-Section of B
1	[1...Na]
2	[2...Na+1]
3	[3...Na+2]
...	...
...	...
...	...
Nb-Na+1	[Nb-Na+1...Nb]

And return the largest Soundex Similarity of all assesses cross-sections between the two strings. With obvious constraints, if either string is zero-length, then a Soundex Score of 0 would be returned. Furthermore, if the two strings are the same length, the results would be identical to that of a regular soundex function mentioned above. Finally, we propose the following Module on top of the two functions declared above:

"Rolling" Soundex Function:

```
Function rollingSim(baseStr As String, searchStr As String) As Double
    Dim rollLength As Long
    Dim bestSim As Double
    Dim crossSection As String
    Dim crossSim As Double
    'a crucial condition here is that the base string must be larger than the search string.
    'if we do not satisfy it, we recursively call the inverse.
    If Len(searchStr) >= Len(baseStr) Then
        rollingSim = rollingSim(searchStr, baseStr)
        Exit Function
    End If

    'how long each mid-section of the base string is.
    rollLength = Len(searchStr)
    'the closest Soundex Distance
    bestSim = 0

    For i = 1 To Len(baseStr) - rollLength+1
        crossSection = Mid(baseStr, i, rollLength)
        crossSim = Similarity(crossSection, searchStr)
        If crossSim > bestSim Then
            bestSim = crossSim
        End If
    Next

    rollingSim = bestSim
End Function
```

As a minor constraint, this function distinguishes between a base string and a search string, similar to the original formula proposed earlier. In this version, the base string and search string are reversed if the length criteria fails. Improvements can be made to this version with ease, as this was the result of a minor time constraint during the time of publication. The key distinction in this formula is that it iterates through the cross-sections mentioned above and returns the soundex score of the closest-matching cross section. The final return value is the most accurate Soundex value of all cross-sections assessed. From here, we can evaluate performance.

Results from Rolling Soundex Function [Public Function RollingSim()]

String 1	String 2	Similarity
Credit One	Credig Onn	0.8
Michael Welseley	Ashton Welseley	0.6
120 Corporate	5620 Southwyck	0.2857
$5,622	$5,622	1.0
Series A	Series A Type	1.0
Series A	Series A Type Document	1.0
Credit One Bank	Credit One Bank, USA, NA	1.0
Finance Credit Bank, USA	Credit One Bank, USA, NA	0.66

The primary takeaway here is that while most Soundex results stayed roughly the same with slight variation, the second-to-last entry was evaluated as being 100% similar at the optimal cross-section, which is more in line with the intuition one would take up during an auditing process for creditor and collector names.

Summary & Application

To be clear, this function is not designed to fully replace the traditional Soundex function outline earlier. In general, the Soundex function does a robust job in 90% of cases, while a rolling Soundex might serve as a secondary approach for edge cases, such as the one with Credit One Bank's example. In auditing modules, a proposed conditional statement would assess if the regular Similarity() function returns a result higher than the specified dissimilarity tolerance, and if not, then to re-evaluate two strings using a rolling method, largely to conserve computing power. That being said, the rolling similarity function demands much more processing than the regular one, and has drawbacks both in processing requirements and data "overfitting" in cases where a recognition of dissimilarity might be important on an operational level.

Reference:

Original Post for VB-Soundex Conversion, November 2006

Soundex - Wikipedia

Accessing the VBA Editor

Github: SoundexRevision