Application of Approximate String Matching in Consumer POS Data

Riki Saito

August 14, 2016

Contents

Introduction

In the summer of 2016, I worked as an intern at Logitech in the Market Analytics team.

Background

Market Analytics is a function in business that evaluate business performance by studying and analyzing the market to obtain insights that help inform decisions and optimize the profitability of the business.

Due to the increasing competitiveness in business and speed in production and manufacturing, there is a growth in need for Market Analytics functions, especially in the consumer products industry. For any business it has become necessary to understand the market size and demand of their products/services, trends in the market and consumer behaviors, and the proportion of the market (market share) controlled by the business and by competitors in the same market.

With the increase in availability and quality of data that can be obtained due to the rise of the digital world and the web, businesses are able to even further leverage the value of Market Analytics to the performance of their business. From a wide variety of sources, businesses are now able to obtain a large amount of market data. With that, the potential to dig deeper insights about market trends and consumer behaviors have grown drastically.

Logitech is a company that has a wide variety of product categories in their portfolio. Currently Logitech (and its subsidiaries) has products in Computer Accessories, Tablet Accessories, Webcams, Mobile Speakers, PC Speakers, Headphones and Earphones, and more.

Definitions

Product Line (PL): a group of products comprising of different sizes, colors, or types, produced or sold under one unique product or model name (i.e. Apple iPhone 5, UE Boom 2)

Stocking Keeping Unit (SKU): a particular product identified by its product line as well as by its size, color, or type, where its identification is typically used for inventory purposes (i.e. Apple iPhone 5 64GB Black, UE Boom 2 Blue)

Data Mapping

Edit Distance

Problem

With the increasing size and availability of data, the need for mainting data quality is also an issue that has surged in all industries, including market analytics.

In market analytics at Logitech, due to the variety of product categories and the global presence of the company, we obtain data from a number of different sources. As a result, we face the issue of data reported slightly differently. A major issue that we encountered were the levels of granularity or hierarchy that the data is reported in, and inconsistencies with names of the Stocking Keeping Units and Product Lines.

A typical data hierarchy in market data might appear like this:

Data Hierarchy	Description	Example
Category	General Category of products	Pointing Devices
Brand	Name of Brand or Company	Logitech

Data Hierarchy	Description	Example
Product Line	Group of products, defined by a model name	3Button M325 Wireless Optical Scroll Mouse
Stock Keeping Unit	A product, defined by its model and features (color, size, type, etc)	3Button M325 Wireless Optical Scroll Mouse Brick Red

The problem we face is that some data sources only provide the Product Line (PL), while others will only provide the Stock Keeping Unit (SKU). Because of this, we are not able to join data at neither the Product Line nor the Stock Keeping Unit, and therefore we are not able to get an accurate measure of sales for each SKU or PL. This hinders the ability to perform market analysis and uncover meaningful insights that could be used to inform business decisions, such as increasing or decreasing production of certain SKUs or PLs.

In this paper, I will discuss two scenarios that I encountered, and the solutions I came up with to resolve this issue.

1: Identifying SKU that belong to the same Product Line

In one of our data source, the sales were reported at the SKU level, but no data was available for the Product Line to which these SKUs belong to. Here is an example of how the data might appear:

Tables in TOC!

Product Line	SKU
-	Product A Bluetooth Mobile Speaker Black
-	Product A Bluetooth Mobile Speaker Green
-	Product A Bluetooth Mobile Speaker Gray
-	Product B Tablet Case for iPad Black
-	Product B Tablet Case for iPad Gray

Problem here: we are not able to product Product Line Rankings and only compare sales at the SKU level

What we need:

Product Line
Product A Bluetooth Mobile Speaker
Product A Bluetooth Mobile Speaker
Product A Bluetooth Mobile Speaker
Product B Tablet Case for iPad
Product B Tablet Case for iPad

2: Multiple instances of the same Product Line appearing with different names

Between two data sources, the names of the Product Line was inconsistent. This was a problem for us because we could not map the data by Product Line, and therefore we could not aggregate the sales data of the Product Line.

Product Line	Unique PL	Sales
Product A Bluetooth Mobile Speaker	-	25
A BLUETOOTH SPEAKER	-	30
Product B Tablet Case for iPad	-	40

Product Line	Unique PL	Sales
B TABLET CASE IPAD	-	5

Problem here: not accurate ranking of product line (by this Product B is most selling, where actually Product A has most sales)

What we need :

Unique PL	Sales
A Bluetooth Speaker	25
A Bluetooth Speaker	30
B Tablet Case iPad	40
B Tablet Case iPad	5

Method

Essentially what we are hoping to accomplish is match and group strings that are very similar but not quite exactly the same. Some simple and intuitive solutions would be to generate a substring of one string to match another, or identify and remove irrelevant words, but these solutions may not be the best solution simply because of the varied structures of how these strings are constructed and the ambiguity in determine whether words are irrelevant or not in identifying the product line.

Since we are dealing with a large data set (~250,000 rows) with varied structures and vocabulary of words, we sought for a fast and scalable approach that is robust to different string structures and vocabularies.

The method that we ultimately selected is called Fuzzy String Matching (also called Approximate String Matching). Fuzzy String Matching is a pattern matching algorithm that computes the degree of similartiy between two strings, and produces a quantitative metric of distance that can be used to classify the strings as a match or not a match.

(Source)

Fuzzy String Matching is very appropriate for our scenario for three reasons:

1) Flexible: robust to varied string constructions, symbols/characters/words that appear in the string, and sequence of words
2) Scalable: can be applied different data sizes, small or large, and can accomodate new data and instances
3) Adjustable: large selection of algorithms and parameters that can be used directly, or adjusted/tweaked

Now we will discuss two scenarios I encountered, and the algorithms I used to match

A large issue with assessing the performance of different fuzzy string matching algorithms is that, we do not have a means of checking the accuracy of the matching other than by manual inspection. Therefore, rather than blindly test algorithms and check each SKU for the accuracy of matching, we made some generalizations and assumptions about the data, and selected/tweaked algorithms and parameters that most closely aligned the assumptions.

Scenario 1: Grouping Stocking Keeping Units (SKU) to Product Lines

As we saw in table (?), the first scenario is identifying the Product Line from the descriptions of the SKU. In general, the product descriptions are constructed with words that are specific to the Product Line (e.g. model name, type of product) as well as those specific to the SKU (color, size, etc). Out of a number of product descriptions, our goal is to identify ones that belong to the same Product Line.

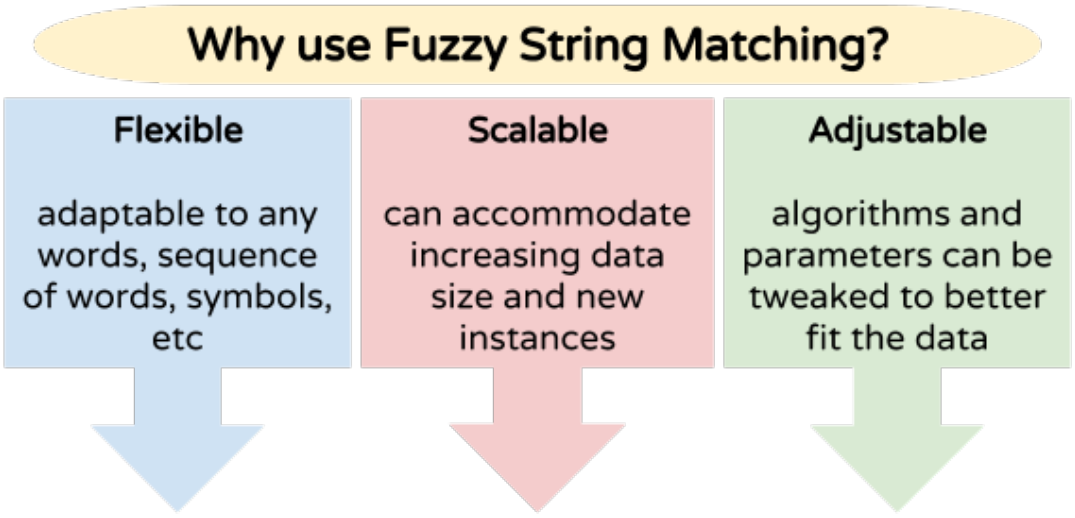

Figure 1:

Initially, we noted a clear patterns in these product descriptions, specifically in the similarity of sequence/choice of words in descriptions that belong to the same Product Line, and its distinction with sequence/choice of words of other Product Lines. To summarize this first pattern:

1) Similar Naming Conventions shared within Similar Product Lines (presence and sequence of words)

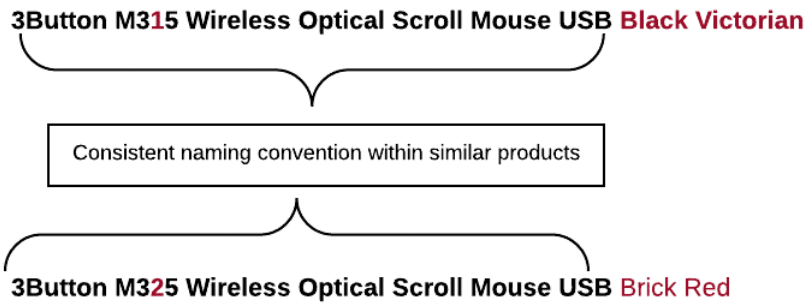

Figure 2:

For example, between these two product descriptions, a majority of the description can be matched, with differences occuring in the Product Line name (M315 vs M325) and SKU-specific words (Black Victorian vs. Brick Red)

This assumption is very straightforward. We will apply an algorithm that is robust to this assumption:

Optimal String Alignment

Also known as Damerau Levenshtein Algorithm (wikipedia), Optimal String Alignment is an algorithm that calculates the "edit distance" between two strings. An "edit" is identified by one of these four:

Edit	Description	Example
Deletion	deletion of a single symbol	**b**eat -> eat
Insertion	insertion of a single symbol	eat -> **b**eat
Substitution	substitution of a single symbol with another	**b**eat -> **h**eat
Transposition	swapping of two adjacent symbols	be**at** -> be**ta**

– [Levenshtein, 1966][leven]

We will use the package `stringdist` in R to compute the edit distance of the example from above.

```
library(stringdist)

a = "M315 Wireless Optical Scroll Mouse Black Victorian"
b = "M325 Wireless Optical Scroll Mouse Brick Red"

stringdist(a, b, method = "osa")
```

```
## [1] 12
```

The edit distance is the number of edits required to transform one description to the other, which is a total of 12 edits.

Edit distance can be converted into a score by dividing by maximum number of possible edits (length of longer string).

```
max(nchar(a), nchar(b))
```

```
## [1] 50
```

```
stringdist(a, b, method = "osa")/max(nchar(a), nchar(b))
```

```
## [1] 0.24
```

With a maximum number of edits of 50, the score is 12/50, or 0.24, out of a maximum of 1.

Pros

It captures the difference in model name and in color variations

Why it wouldn't work in our scenario

You may have noticed that the difference in model name is only one "edit", whereas the difference in SKU variation accounts for 11 single edits. Out of the total number of edits possible (50 single-character edits), we have a score of .24 out of a total score of 1, but only .02 of this is accounted by different product line, and the rest of the .22 are captured by the difference in SKU-specifc words. If our goal is to identify matching product lines and not SKU, this method does not quite satisfy our goal.

Here is an example scenario where this poses an issue

```
library(stringdist)

a = "M315 Wireless Optical Scroll Mouse Black Victorian"
b = "M325 Wireless Optical Scroll Mouse Brick Red"
c = "M325 Wireless Optical Scroll Mouse Black Victorian"

(a.to.c = stringdist(a, c, method = "osa")/max(nchar(a), nchar(c)))
```

```
## [1] 0.02
```

We can see that even though we want to map b to c, we end up mapping a to c with this method. The reason being that, even though b and c should belong together as the same product line, the a string is closer to c. We need a method that is robust to differences in product line, not in SKU.

Weighted Optmial String Alignment

1) Naming Convention same (presence and sequence of words describing product)
2) Words specific to the SKU towards end of description
3) Words specific to the Product Line towards beginning of description

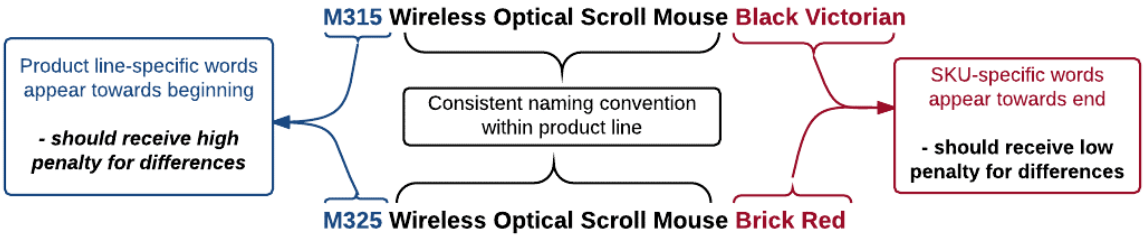

Figure 3:

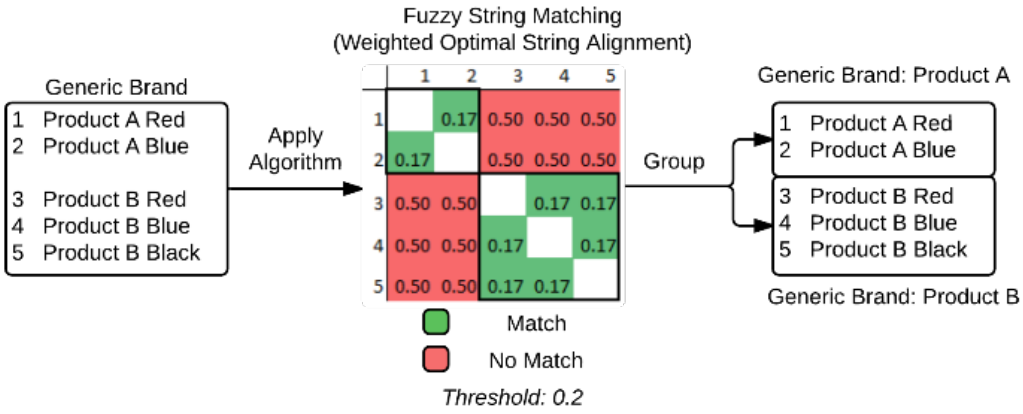

Figure 4:

Using this assumption about the data, we alter the algorithm by adding a location-based weighting.

Parameters

Parameter	Description	Default
Type	Option to apply algorithm by whole item or by single character. Since we want to apply algorithm to WHOLE WORDS rather than single characters, by item is ideal.	character
Weight	Weightfunction (linear, quadratic, root). In the previous example, linear was used	linear

Parameter	Description	Default
Sum.Right	Flagfor whether or not to consider everything after the first mismatch (from the left-hand side) as mismatches (in other words, sum scores of all mismatches to the right of first mismatch)	F

Example: Amazon Products

https://www.amazon.com/dp/B00N32I2Q6/ref=twister_B01AS57B0I?_encoding=UTF8&psc=1

```
wosa <- function(a, b, type = "character", weight = "linear", sum.right = F){
  if(type == "character"){
    A <- strsplit(a, '')[[1]]
    B <- strsplit(b, '')[[1]]
    d <- array(0, c(nchar(a)+1, nchar(b)+1))
  } else if(type == "item"){
    A <- strsplit(a, ' ')[[1]]
    B <- strsplit(b, ' ')[[1]]
    d <- array(0, c(length(A)+1, length(B)+1))
  }
  for(i in seq_len(length(A))+1){
    for(j in seq_len(length(B))+1){
      if(A[i-1] == B[j-1]) cost <- 0 else cost <- 1
      d[i,j] <- min(c(  d[i-1,j] + 1,       #deletion
                d[i, j-1] + 1,      #insertion
                d[i-1, j-1] + cost))    #substitution
    if(i > 2 & j > 2) if(A[i-1] == B[j-2] & A[i-2] == B[j-1])
      d[i,j] <- min(d[i,j], d[i-2, j-2] + cost) #transposition
    }
  }
  osa <- d[length(A)+1, length(B)+1]
  #want to weight score based on where character/item edits occured
  if(ncol(d) == nrow(d)) edits <- diff(diag(d)) else {
    short_len <- min(length(A), length(B))+1
    edits <- diff(c(diag(d[1:short_len, 1:short_len]), d[short_len:(length(A)+1), short_len:(length(B)+
  }
  edit.at <- which(edits != 0)
  long_len <- max(length(A), length(B))
  if(weight == "linear") w <- long_len:1 else
  if(weight == "quadratic") w <- (long_len:1)^2 else
  if(weight == "root") w <- sqrt(long_len:1) else stop("Define Weight: linear, quadratic, or root")
  wosa <- NULL
  if(length(edit.at) >0) {
    if(sum.right) wosa$score <-  sum(w[min(edit.at):long_len])/sum(w) else wosa$score <- sum(w[edit.at])
  } else wosa$score <- 0
  wosa$weights <- w/sum(w)
  wosa$edit.loc <- edit.at
  return(wosa)
}

wosa(a, b)$score
```

```
## [1] 0.09411765
```

```
wosa(a, c)$score
```

```
## [1] 0.03764706
```

Using a Weighted Optimal String Alignment (WOSA), we are able to differentiate between a and c, while capturing an even closer distance between a and b.

But more parameters

```
wosa(a, b,  type = "item", weight = "linear", sum.right = T)$score
```

```
## [1] 1
```

```
wosa(a, c,  type = "item", weight = "linear", sum.right = T)$score
```

```
## [1] 1
```

Now let us look at an example from amazon

```
wosa("Bose SoundLink Color Bluetooth Speaker (Blue)", "Bose SoundLink Color Bluetooth Speaker (Black)",
```

```
## $score
## [1] 0.04761905
##
## $weights
## [1] 0.28571429 0.23809524 0.19047619 0.14285714 0.09523810 0.04761905
##
## $edit.loc
## [1] 6
```

Assessment of all parameters?

Scenario 2: Matching Product Lines

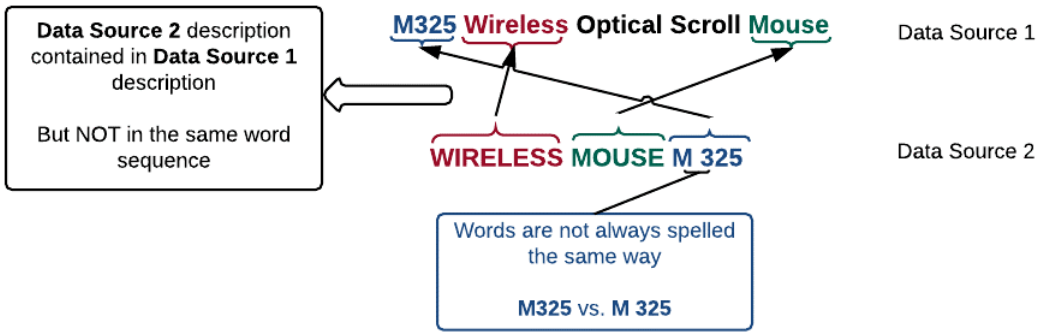

Figure 5:

n-gram/q-gram

Definition: All subsequences of N/Q consecutive items from a given sequence or string (N/Q is an integer)

Item can be identified as whole words, syllables, single characters, etc (example to the right is a 5-gram by single characters)

N-gram is a frequency table of all subsequent occurrences

Q-gram is a distance measure based on common n-gram between two strings

Ideal for catching typos

Jaccard Distance

distance measure using q-gram

tables from slides

Adjusted Jaccard Distance

matching diagrams

parameters

```
#adjusted jaccard distance
ajd <- function(a, b, q = 2){
  require(stringdist)
  qa <- qgrams(tolower(a), q = q)
  qb <- qgrams(tolower(b), q = q)
  qab <- merge(data.frame(t(qa)), data.frame(t(qb)), by = "row.names", all.x = T)
  qab[is.na(qab)] <- 0
  qmatch <- sum(as.numeric(apply(qab, 1, function(x) min(x[2:3]))))
  qdenom <- min(sum(qa), sum(qb))
  score <- 1 - qmatch/qdenom
  return(score)
}

A = "M325 WIRELESS OPTICAL SCROLL MOUSE"
B = "WIRELESS MOUSE M 325"
ajd(A,B)
```

```
## [1] 0.2105263
```

Example: Merging with Internal PL Unique Identifier

```
![http://plaza.ufl.edu/jgu/public_html/C-uppsats/figure5-1.jpg](http://plaza.ufl.edu/jgu/public_html/C-
```

$$Jaccard\ Distance = 1 - \frac{\#\ matches}{\#\ total\ n\text{-}grams} = 1 - \frac{5}{8} = 37.5\%$$

Figure 6:

$$Adjusted\ JD = 1 - \frac{\#\ matches}{\#\ total\ n\text{-}grams\ of\ shorter\ string} = 1 - \frac{5}{6} = 16.7\%$$

Figure 7:

Discussion of

good if assumptions are met

need to be careful if assumptions are met

Conclusion

location weighting can be applied to other algorithms

References

[leven]:Vladimir I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Doklady Akademii Nauk SSSR, 163(4):845-848, 1965 (Russian). English translation in Soviet Physics Doklady, 10(8):707-710, 1966.