User Guide

Data Washing Machine – Python Refactor, Version 2.21

Contents

[Project Overview 5](#_Toc85638730)

[Code Repository 5](#_Toc85638731)

[Setting Parameters and Running the Program 5](#_Toc85638732)

[Annotated Test Data 6](#_Toc85638733)

[Logic Overview 7](#_Toc85638734)

[Tokenization Process 7](#_Toc85638735)

[inputFileName 8](#_Toc85638736)

[delimiter 8](#_Toc85638737)

[hasHeader 8](#_Toc85638738)

[tokenizerType 8](#_Toc85638739)

[truthFileName (Optional) 9](#_Toc85638740)

[runIterationProfile (Optional) 9](#_Toc85638741)

[addRefsToLinkIndex (Optional) 9](#_Toc85638742)

[Global Spelling Correction Process (Optional) 9](#_Toc85638743)

[runGlobalCorrection 10](#_Toc85638744)

[minFreqStdToken 10](#_Toc85638745)

[maxFreqErrToken 10](#_Toc85638746)

[minLenStdToken 11](#_Toc85638747)

[globalCorrectionDetail 11](#_Toc85638748)

[Blocking Process 11](#_Toc85638749)

[beta 12](#_Toc85638750)

[minBlkTokenLen 12](#_Toc85638751)

[excludeNumericBlocks 13](#_Toc85638752)

[blockByPairs 13](#_Toc85638753)

[Block Correction Process (Optional) 13](#_Toc85638754)

[blockCorrection 13](#_Toc85638755)

[blockCorrection 13](#_Toc85638756)

[Linking Process 13](#_Toc85638757)

[comparator 14](#_Toc85638758)

[mu 14](#_Toc85638759)

[muIterate 15](#_Toc85638760)

[matrixNumTokenRule 15](#_Toc85638761)

[matrixInitialRule 15](#_Toc85638762)

[sigma 16](#_Toc85638763)

[removeDuplicateTokens 16](#_Toc85638764)

[removeExcludedBlkTokens 16](#_Toc85638765)

[Cluster Evaluation Process (Separation of Good and Bad Clusters) 16](#_Toc85638766)

[epsilon 17](#_Toc85638767)

[epsilonIterate 17](#_Toc85638768)

[System Outputs 17](#_Toc85638769)

[<input\_name>-LinkIndex.txt 17](#_Toc85638770)

[DWM\_Log\_<date>\_<hour>\_<minute>.txt 18](#_Toc85638771)

[DWM100\_<date>\_<hour>\_<minute>.xlsx 19](#_Toc85638772)

[Appendix A: Release Notes 20](#_Toc85638773)

[Release 1.7 20](#_Toc85638774)

[Release 1.8 20](#_Toc85638775)

[Release 1.9 20](#_Toc85638776)

[Release 2.0 21](#_Toc85638777)

[Release 2.1 21](#_Toc85638778)

[Release 2.2 21](#_Toc85638779)

[Appendix B: Parameter File Template 22](#_Toc85638780)

Document Version Control

|  |  |  |
| --- | --- | --- |
| Draft | Author | Description |
| 1.0 | Jtalburt | Initial draft of the documentation for Code Version 1.5 |
| 2.0 | Jtalburt | Second draft updated for Code Version 1.6 |
| 3.0 | Jtalburt | Updated for Code Version 1.7 |
| 4.0 | Jtalburt | Updated for Code Version 1.8 |
| 5.0 | Jtalburt | Updated for Code Version 1.9 |
| 6.0 | Jrtalburt | Updated for Code Version 2.0 |
| 7.0 | keanderson | Updated for Code Version 2.1 |
| 8.0 | jrtalburt | Updated for Code Version 2.2 |
| 8.1 | Jrtalburt | Fixed bugs in DWM25\_Global\_Token\_Replace & DWM90\_IterateClusters |

# Project Overview

Data curation is the process of acquiring multiple sources of data, assessing, and improving data quality, standardizing, and integrating the data into a usable information product, and eventually disposing of the data. The Data Washing Machine (DWM) described here is a proof-of-concept for an unsupervised data curation. The DWM is focused on a basic form of data cleansing in the form of identifying redundant records through entity resolution and spelling corrections. The novelty of the approach is to use ER as the first step using an unsupervised blocking and stop word schemes based on token frequency. A variant of the Monge-Elkan comparator, called a scoring matrix, is used for linking unstandardized references. Linking is followed by an unsupervised process for evaluating the quality of the linking results based on variation of Shannon entropy.

The ER process is iterative, and in each iteration, the reference similarity threshold is increased. The prototype was tested on 18 fully annotated test samples of primarily synthetic person data varied in two different ways, good data quality versus poor data quality, and a single record layout versus two different record layouts. In samples with good data quality and using both single and mixed layouts, the final clusters had an average F-measure of 0.91, precision of 0.96, and recall of 0.87 outcomes comparable to results from a supervised ER process. In samples with poor data quality whether mixed or single layout, the average F-measure was 0.78, precision 0.74, and recall 0.83 showing that data quality assessment and improvement is still a critical component of successful data curation. The results demonstrate the feasibility of building an unsupervised ER engine to support data integration for good quality references while avoiding the time and effort to standardize references sources to a common layout, design, and test matching rules, design blocking keys, or test blocking alignment.

## Code Repository

The DWM is written entirely in Python. The most current version of the code can be found on BitBucket.org under the name “DWM Refactor V1”   
<https://bitbucket.org/oysterer/dwm-refactor-v1/src/master/>

The code was developed using the Anaconda Jupyter Notebook development environment. All DWM python modules in the repository are provided in both notebook format (.ipynb) and standard python format (.py).

## Setting Parameters and Running the Program

The DWM actions are controlled by a set of parameters read at the start of the program. Currently, there are 29 parameters that can be set by the user to control the actions of the DWM. To assist users, the code base includes a parameter file template named “parms\_File\_Template.txt”. This file includes all parameters for current code base including comments summarizing the function of each parameter and its default value, if any. The value for each parameter in the template is the string “???”. The parameters are listed in the order in which they are used by the program. Appendix B shows the structure of the parameter file template.

The parameter files can be fed into the program either one at a time or in a batch (a list of parameter files). When the DWM is started, it will prompt the user to enter either the value “1” or “2” to indicate which method will be used.

* Entering 1 indicates that the user will enter the name of a single parameter file
* Entering 2 indicates that the user will enter the name of a text file containing a list of parameter file names

The lines of the parameter file have the format

<parameter name>=<parameter value>

Any line starting with the # character is ignored as a comment. If a parameter is omitted from the parameter file, the default value will be used provided a default value is defined for the parameter. Required parameters do not have a default value and must be given in the parameter file.

To assist users, the code base includes a parameter file template named “parms\_File\_Template.txt”. This file includes all parameters for current code base including comments summarizing the function of each parameter and its default value, if any. The value for each parameter in the template is the string “???”. The parameters are listed in the order in which they are used by the program. Appendix B shows the structure of the parameter file template.

In addition, two example parameter files “S2-parms.txt” and “S8-parms.txt” are included in the code base fully populated with parameter values for running test Samples “S2G.txt” and “S8P.txt”, respectively.

# Annotated Test Data

The BitBucket repository also includes 18 test datasets. These datasets all have associated truth sets (annotations) that allow the user to check the accuracy of the clustering for a given set of parameter settings. Table 1 gives the name and description each test dataset and its associated truth file. If the truth file name is given as the value of the “truthFileName” parameter, then the program will calculate the precision, recall, and F-measure of the clustering.

Table 1: Test Dataset Characteristics and Associated Truth File

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| File Name | Size | Characteristics | Quality | Layout | Truth File Name |
| S1G.txt | 50 | Person name & address | Good | Single | truthABCgoodDQ.txt |
| S2G.txt | 100 | Person name & address | Good | Single | truthABCgoodDQ.txt |
| S3Rest.txt | 868 | Business name & address | Good | Single | truthRestaurant.txt |
| S4G.txt | 1,912 | Person name & address | Good | Single | truthABCgoodDQ.txt |
| S5G.txt | 3,004 | Person name & address | Good | Single | truthABCgoodDQ.txt |
| S6GeCo.txt | 19,998 | Person name & address | Good | Single | truthGeCo.txt |
| S7GX.txt | 2,912 | Person name & address | Good | Mixed | truthABCgoodDQ.txt |
| S8P.txt | 1,000 | Person name & address | Poor | Single | truthABCpoorDQ.txt |
| S9P.txt | 1,000 | Person name & address | Poor | Single | truthABCpoorDQ.txt |
| S10PX.txt | 2,000 | Person name & address | Poor | Mixed | truthABCpoorDQ.txt |
| S11PX.txt | 3,999 | Person name & address | Poor | Mixed | truthABCpoorDQ.txt |
| S12PX.txt | 6,000 | Person name & address | Poor | Mixed | truthABCpoorDQ.txt |
| S13GX.txt | 2,000 | Person name & address | Good | Mixed | truthABCgoodDQ.txt |
| S14GX.txt | 5,000 | Person name & address | Good | Mixed | truthABCgoodDQ.txt |
| S15GX.txt | 10,000 | Person name & address | Good | Mixed | truthABCgoodDQ.txt |
| S16PX.txt | 2,000 | Person name & address | Poor | Mixed | truthABCpoorDQ.txt |
| S17PX.txt | 5,000 | Person name & address | Poor | Mixed | truthABCpoorDQ.txt |
| S18PX.txt | 10,000 | Person name & address | Poor | Mixed | truthABCpoorDQ.txt |

# Logic Overview

The diagram in Figure 1 shows the basic flow of the DWM.

**Source 1**

**Source 2**

**Merge**

**Tokenize**

**Global Correction**

**Block & make pair list**

**Link pairs**

**Transitive Closure**

**Clusters**

**Evaluate Clusters**

**Bad Clusters**

**Linked Pairs**

**Good Clusters**

**Empty?**

**Increment mu**

**No**

**Yes**

**End**

**beta**

**sigma mu**

**epsilon**

**Pair List**

**Make block corrections, re-block, & make new pair list**

**Block Correct?**

**Yes**

**No**

Figure 1: Data Washing Machine Process Flow

The references are first merged into a single file without any standardization or cleaning. Any header records or other metadata annotation is discarded.

## Tokenization Process

The merged file is tokenized removing any non-word characters leaving only letters and digits. All letters are converted to upper case. During the tokenization, the frequency of each unique token in the file is counted and stored in a dictionary for use in later processes including token correction, token standardization, and record blocking.

The tokenization process also generates several statistics related to the distribution of token frequencies and token lengths. Some of these statistics such as “Total Tokens Found” will vary for the same dataset depending upon the tokenizer used. These statistics include

* Total References Read
* Total Tokens Found
* Total Unique Tokens
* Unique Token Ratio = Total Unique Tokens/Total Tokens Found
* Total Numeric Tokens Found (tokens of all digits)
* Numeric Token Ratio = Total Numeric Tokens/Total Tokens Found
* Minimum Token Frequency
* Maximum Token Frequency
* List of the 10 highest frequency tokens
* Average Token Frequency (weighted average by frequency)
* Standard Deviation of Token Frequency (weighted standard deviation by frequency)
* Minimum Token Length
* Maximum Token Length
* Average Token Length (weighted average by frequency)
* Standard Deviation of Token Length (weighted standard deviation by frequency)

Tokenization Parameters

### inputFileName

Path to the merged input file of references.

**Constraint**: Must be a valid path to a text file of references

**Default** Value if not Given: **None**, must be specified

**Example:** inputFileName=S2G.txt

### delimiter

A single character used to separate attribute values in the input file

**Constraint**: must be a single character

**Default** Value if not Given: ‘**,’ (comma)**

**Example:** delimiter=;

### hasHeader

Boolean value **True**/**False** where   
**True** indicates the first record in the file is a header record and should not be processed as a reference,   
**False** there is no header record

**Constraint**: True or False

**Default** Value if not Given: **False**

**Example:** hasHeader=True

### tokenizerType

The method for tokenizing the references. The value must be either “**Splitter**” or “**Compress**”.   
**Splitter** indicates the tokens will be split on every non-word character.   
**Compress** indicates the non-word characters will be replaced with an empty string.

For example, if “tokenizerType=Compress” and if the input value is “Ab-78”, then  
**Compress** produces one token “AB78”, whereas  
**Splitter** produces two tokens “AB” and “78”

**Default** Value if not Given: **None**, must be specified

**Example:** tokenizerType=Splitter

### truthFileName (Optional)

The path to a text file giving the correct linkage of the references in the input file. The first line in the file must be a header record such as “RecID, TruthID”. Every line after the header record of must have two values separated by a comma. The first value is a unique record identifier, and the second value is a link identifier. The truth file can have more record identifiers than the input file, but every record identifier in the input file must be in the truth file. When a value is given for this parameter, the DWM will use the truth file to calculate the precision, recall, and F-measure of the final clusters.

**Constraint**: Values must be empty string (‘’) or a valid path to a truth file.

**Default** Value if not Given: empty string (‘’)

**Example:** truthFileName=truthABCgoodDQ.txt

### runIterationProfile (Optional)

**True** indicates the cluster profile will be output at the end of each iteration. If a truth file was given, it will also output the precision, recall, and F-measure at the end of each iteration. These statistics include both the good and the bad clusters after the clusters generated for the iteration have been assessed against the epsilon value.

**False** indicates the profile and cluster accuracy statistics are only produced after all iterations are complete.

**Default** value if not given: **False**

**Example:** runIterationProfile=True

### addRefsToLinkIndex (Optional)

**True** indicates the complete (tokenized) reference will be included in the Link Index in addition to the reference identifier and the cluster identifier.

**False** indicates the Link Index will only include the reference identifier and the cluster identifier

**Default** value if not given: **False**

**Example:** addRefsToLinkIndex =True

## Global Spelling Correction Process (Optional)

After the tokenization process, an optional process is to correct the spelling for non-numeric tokens. The process is driven by 3 parameters, minFreqStdToken, maxFreqErrToken, and minLenStdToken, and an external list of dictionary and name words (WordList). The process examines every pair of non-numeric tokens (A, B) where the freq(A)≥minFreqStdToken, and the freq(B)≤maxFreqErrToken. The pairs are systematically examined in descending order of A frequency and ascending order of B frequency.

If A and B satisfy the following conditions,

* Frequency(A)≥minFreqStdToken
* Frequency(B)≤maxFreqErrToken
* Damerau-Levenshtein edit distance between token A and B is equal to 1 (i.e., they differ by at most one character or the transposition of two adjacent characters)
* Length(A)≥minLenStdToken
* B does not appear in WordList

Then, A will replace B everywhere B is found in the tokenized input file.

Once A is determined to be a replacement for B, then B is no longer a candidate for replacement by another token. However, A can replace more than one misspelling.

To illustrate, let minFreqStdToken = 6, minLenStdToken = 3, and maxFreqErrToken = 3. Also, suppose the token “AMOS” has a frequency of 12, and “AOMS” has a frequency of 2 and is not found in the WordList (name list). In this case, “AMOS” (the standard token) would replace the 2 occurrences of “AOMS” (the error token) because the edit distance between the two tokens is 1 because they differ by the transposition “O” and “M”. Also, the frequency(“AMOS”) = 12 > minLenStdToken = 6, and frequency(“AOMS”) = 2 < maxFreqErrToken = 3.

Global Spelling Correction Parameters

### runGlobalCorrection

Boolean value **True**/**False** where   
**True** indicates running the correction process,   
**False** do not run correction

**Constraint**: True or False

**Default** Value if not Given: **False**

**Example:** runReplacement=True

### minFreqStdToken

An integer value representing the minimum frequency of a token for it to be considered as a correct (standard) token.

**Constraint**: Integer value ≥ 1

**Default** Value if not Given: **None**, must be specified if runReplacement=True

**Example:** minFreqStdToken=5

### maxFreqErrToken

An integer value representing the maximum frequency of a token for it to be considered a misspelling (error) token.

**Constraints**:   
maxFreqErrorToken ≥ 1  
minFreqStdToken > maxFreqErrToken

**Default** Value if not Given: **None**, must be specified if runReplacement=True

**Example:** maxFreqErrToken=3

### minLenStdToken

An integer value representing the minimum length of a token for it to be considered a correct (standard) token.

**Constraint**: Integer value ≥ 1

**Default** Value if not Given: **None**, must be specified if runReplacement=True

**Example:** minLenStdToken=3

### globalCorrectionDetail

Boolean value **True**/**False** where   
**True** will write each of the token corrections to the log file,   
**False** corrections are not written to the log file

**Constraint**: True or False

**Default** Value if not Given: **False**

**Example:** globalCorrectionDetail=True

## Blocking Process

Blocking is required to reduce the total number of comparisons between references to a reasonable quantity. At the same time, references representing the same entity should be placed in the same block. Based on the assumption that two equivalent references (references to the same entity) will almost certainly share at least one token in common, the DWM blocks comprise all references that same the same token up to a certain token frequency threshold. For example, if the blocking frequency threshold beta is set to 5, then all tokens with a frequency of 2, 3, 4, and 5 will be used to create blocks.

The disadvantage of the frequency blocking is that creates many redundant comparisons. For example, suppose two references R1 and R2 both contain the tokens “JOHN” and “DOE”, and suppose both “JOHN” and “DOE” have frequencies greater than 1 and less than or equal to beta. This means that the references R1 and R2 with be in the “JOHN” block (all references sharing the token “JOHN”) and in the “DOE” block. If all the pairs in the “JOHN” block are compared to each other, and all the pairs in the “DOE” are compared to each other, then R1 and R2 will be compared to each other twice. Most likely, there will be many instances where the same pair of references occur in many different blocks and are repeatedly compared. Because comparing two references is computationally expensive, frequency blocking is inefficient because it makes so many unnecessary comparisons of the same two references.

To make the blocking process more efficient, the DWM performs a pre-process to reduce the number of comparisons. Instead of generating all the pairs of references in each block and comparing them, the process first generates the pairs of reference identifiers in the block. These pairs are added to list of pairs, and after this has been done for all blocks, this list is deduplicated so that final list of pairs only contains the unique pairs across all blocks. It is then only the unique pairs of references across all blocks that are compared.

So, following the previous examples, the “JOHN” block adds the pair (R1, R2) to the list, and the “DOE” block also adds the pair (R1, R2) to the list. After the list is sorted and deduplicated, the pair (R1, R2) only occurs once in the final pair list.

After the pair list is created it either goes directly to the linking process or to an optional block correction process that tries to make additional corrections to the references before going into the linking process. In the linking process, each unique pair of references are compared to similarity where the decision is made to link the references or not to link the references.

While by default every token with a frequency between 2 and beta forms a block, there three ways to add additional qualifications before a token can be used as a blocking token. The first is to not allow tokens of a certain length to form blocks. The parameter minBlkTokenLen sets a minimum length for a blocking token. So, while a token may have the right frequency, if its length is below the value of minBlkTokenLen, it will not form a block. For example, if minBlkTokenLen = 4, then “DOE” in the previous example would not be used to form a block because its length is only 3, below the minimum.

The second qualification is to not allow numeric tokens to form blocks. A numeric token is any token formed entirely from digits. This controlled by the parameter excludeNumericBlocks. When the value of the parameter is True, then even if a token such as “1234” has the correct frequency and meets the minimum length set by minBlkTokenLen, it will not form a block because it comprises only digits.

The third qualification is to require references to only be in the same block if they share two otherwise qualified tokens, instead of just one. This is controlled by the parameter blockByPairs. When this parameter is set to True, references must share two otherwise qualified tokens to be in the same block.

Blocking Parameters

### beta

An integer value representing the maximum frequency of a token for it to be considered as a blocking token.

**Constraint**: beta > 1

**Default** Value if not Given: **None**, must be specified

**Example:** beta = 6

### minBlkTokenLen

An integer value representing the minimum length of a token that can be used for blocking. represents an additional condition for a token to be a blocking token. In addition to the frequency of the token being between 2 and beta, the length of the token must   
be >= minBlkTokenLen Constraint: minBlkTokenLen >= 0

**Constraint**: Integer value

**Default** value if not given in parameter file: **0**

**Example:** minBlkTokenLen = 4

### excludeNumericBlocks

A Boolean value representing indicating whether numeric tokens should be used for blocking. This represents an additional condition for a token to be a blocking token. In addition to the frequency of the token being between 2 and beta, the token must not be numeric, i.e., comprises only digit characters.

**Constraint**: **True** or **False**

**Default** value if not given in parameter file: **False**

**Example:** minBlkTokenLen = 4

### blockByPairs

A Boolean value representing indicating whether references in the same block should share at least two tokens otherwise qualified by frequency, length, and composition. If this parameter is True, then two references will appear in the same block only if they share at least two qualified blocking tokens, otherwise they are only required to share at least one qualified blocking token.

**Constraint**: **True** or **False**

**Default** value if not given in parameter file: **True**

**Example:** blockByPairs = False

## Block Correction Process (Optional)

Block Correction Parameters

### blockCorrection

Boolean value **True**/**False** where **True** indicates running the block correction process to update missing and misspelled tokens between blocks of input data,   
**False** do not run block correction

**Constraint**: True or False

**Default** Value if not Given: **False**

**Example:** blockCorrection = True

### blockCorrection

Boolean value **True**/**False** where **True** indicates running the block correction process to update missing and misspelled tokens between blocks of input data,   
**False** do not run block correction

**Constraint**: True or False

**Default** Value if not Given: **False**

**Example:** blockCorrection = True

## Linking Process

In the Linking Process, every pair of references given in the pair list are compared for similarity using a similarity function. The similarity function will produce a value from 0.0 to 1.0 indicating the degree of similarity between the two references. A high value near 1.0 indicates a higher level of similarity and lower values represent lower similarity.

Pairs of references in the pair list with a similarity above a given similarity threshold (mu) are linked together. The Linking Process, the references are collected into “clusters” through the Transitive Closure Process. For example, is Ref1 is linked to Ref2 after processing the “A-block, and Ref2 is linked to Ref3 after processing the “B-block”, then Ref1, Ref2, and Ref3 will all be in the same cluster.

Linking Parameters

### comparator

The name of the similarity function to be used in comparing pairs of references in the pair list.  
Constraint: The valid choices are

* “ScoringMatrixStd” – this is the standard scoring matrix. The reference with the fewest tokens is the selected as the first reference, and its tokens label the rows of the matrix. The tokens in the in the longer second reference label the columns of the matrix. The cells of the matrix hold the similarity between the token labeling the row and the token labeling the column. The largest similarity in each row and column are summed and averaged to calculate the final similarity between the two references.
* “ScoringMatrixKris” – operates the same as the ScoringMatrixStd except the similarities are given a weight, with the similarities between tokens are the beginning of the references given more weight than the similarities between tokens near the end of the references. The weights are calculated as (M – J+1)/Base where M is the number of tokens in the first (shorter) reference, J is the row position of the similarity, and the Base value is the sum of the integers from 1 to M given by Base = M\*(M+1)/2. For example, if there are 9 row tokens, then the similarity in the first row of the matrix will have a weight of 9/45, in the second row 8/45, and so on the row 9 a weight of 1/45.
* “Cosine” – the Cosine similarity function from the Python textdistance Library
* “MongeElkan” – MongeElkan similarity function from the Python textdistance Library

**Default** Value if not Given: **None**, must be specified

**Example:** comparator=ScoringMatrixStd

### mu

A decimal value representing the starting minimum match threshold for linking two references. The threshold will increase with each iteration by the value given by the parameter muIterate.

Constraint: 1.0 ≥ mu >0.0

**Default** Value if not Given: **0.5**

**Example:** mu=0.60

### muIterate

A decimal value representing the value added to mu after each iteration of the DWM.

**Constraint**: 1.0 ≥ muIterate > 0.0

**Default** Value if not Given: **0.10**

**Example:** muIterate=0.05

### matrixNumTokenRule

(This parameter only applies to the ScoringMatrixStd and the ScoringMatrixKris comparators)

A Boolean value of True/False where  
True indicates if either token is all digits (numeric), the similarity value must be either 1.0 if the two tokens are an exact match, or 0.0 if they are not an exact match.   
False indicates the tokens should be compared as strings allowing approximate similarity.

**Constraint**: **True** or **False**

**Default** Value if not Given: **False**

**Example:** matrixNumTokenRule=True

### matrixInitialRule

(This parameter only applies to the ScoringMatrixStd and the ScoringMatrixKris comparators)

A Boolean value of True/False where  
True indicates if either token is single character, the similarity value must be either 1.0 if the two tokens are an exact match, or 0.0 if they are not an exact match.   
False indicates the tokens should be compared as strings allowing approximate similarity.

**Constraint**: **True** or **False**

**Default** Value if not Given: **False**

**Example:** matrixInitialRule=True

Stop Word Parameters

While lower frequency tokens bring together references most likely to be equivalent, high frequency tokens are not helpful in disambiguating references. If a token is likely to be in any reference, then it does not help to discriminate. Therefore, these high-frequency tokens, called stop words, are removed from the reference before comparing.

Tokens with a frequency above sigma are treated as stop and do not participate in the comparison process. In addition, tokens excluded as blocking tokens can also be treated as stop words by setting the removeExcludedBlkTokens parameter to True. Note, stop words remain as part of the reference. They are only removed from a copy of the reference sent to the comparison process.

### sigma

An integer value representing the minimum frequency of a token for it to be considered as a stop word. Stop words (tokens) are removed from the reference before they are compared.

**Constraint**: sigma > beta > 1

**Default** Value if not Given: **None**, must be specified

**Example:** sigma = 12

### removeDuplicateTokens

Boolean value **True**/**False** where   
**True** indicates duplicates of the same token in the same reference are removed,   
**False** all tokens are kept

**Constraint**: True or False

**Default** Value if not Given: **False**

**Example:** removeDuplicateTokens=True

### removeExcludedBlkTokens

A Boolean value representing indicating whether the tokens excluded from blocking by the parameters minBlkTokenLen and excludeNumericBlocks should also be treated as stop words. If removeExcludedBlkTokens is True, these tokens will be removed from the tokenized references and will not participate in the comparison operation

2 <= freq(T) <= beta, and len(T) >= minBlkTokenLen

**Constraint**: minBlkTokenLen >= 0

**Default** value if not given in parameter file: **False**

**Example:** minBlkTokenLen = 4

## Cluster Evaluation Process (Separation of Good and Bad Clusters)

After the blocking and linking process, the clusters of mutually linked references are evaluated for linking quality. The linking quality is a measure of the overall organization of the clusters using a variation of the Shannon entropy formulation. While the comparator evaluates a pair of references for similarity, the cluster quality process evaluates the entire cluster for similarity (consistency). The quality value is a number between 0.00 and 1.00 measuring the level of inconsistency within the cluster. Higher values mean higher levels of similarity between references in the same cluster. If the cluster quality is 1.0, then all references in the cluster must have the same tokens. If the cluster quality is 0.0, it means that none of the references in the cluster share any mutual tokens.

Cluster Evaluation Parameters

### epsilon

A decimal value representing the starting quality threshold for a “good quality” cluster. Optionally, the threshold can be made to increase or decrease with each iteration by the value given for the parameter epsilonIterate.

**Constraint**: 0.0 ≤ epsilon ≤ 1.0

**Default** Value if not Given: **0.5**

**Example:** epsilon=0.83

### epsilonIterate

A decimal value representing the value added to epsilon after each iteration of the DWM. If the value is positive, the value of epsilon will increase each iteration. If the value is 0.0, epsilon will stay constant for each iteration, and if the value is negative, epsilon will decrease each iteration.

**Constraint**: -1.0 ≤ epsilonIterate ≤ 1.0

**Default** Value if not Given: **0.0**

**Example:** epsilonIterate=0.0

As each the quality of each cluster is evaluated against epsilon, the clusters with a quality greater than or equal to epsilon are kept and moved to the link file. The cluster with a quality less than epsilon are moved to a reprocess file where they are re-blocked and re-linked at a higher linking threshold (mu + muIterate). This process continues until there are no references to reprocess or there are no linked references.

# System Outputs

Each run produces four outputs

### <input\_name>-LinkIndex.txt

This text file is always produced from a successful run. It shows the final linking of the input references. There should be N+1 lines in the LinkIndex file where N is the number of references in the input file. The first line of the file contains the header “RefID, ClusterID” and each of the following lines contains two comma separated values where the first value is one of the unique reference identifiers from the input file, and the second value is the link identifier.

For convenience, the lines are sorted in order by the link identifiers (ClusterID).

RefID, ClusterID

A735485,A735485

A739417,A739417

A750205,A750205

A942770,A750205

A750209,A750209

A827462,A750209

### DWM\_Log\_<date>\_<hour>\_<minute>.txt

The run log is always produced from a successful run. The run log records of all actions taken during the run. The run log is useful for research because it captures the key outputs along with the parameter values producing the output. It shows what happened during each iteration of the blocking-linking-evaluation cycle.

The final part of the run shows the final cluster profile, and it a Truth File was given, it shows the final Precision, Recall, and F-Measure of the linking results.

Data Washing Machine Refactor Version 1.8

Data/Time 20210709\_15\_57

>> Starting DWM20

Input Reference File Name = S8P.txt

Tokenized Reference Output File Name = S8P-Tokenized.txt

Input File has Header Records = True

Input File Delimiter = ,

Tokenizer Function Type = Compress…

…

Cluster Profile

Size Count

1 192 192

2 31 62

3 17 51

4 24 96

5 13 65

….

>>Starting DWM99

Truth File Name= truthABCpoorDQ.txt

L= 3068.0 E= 2811.0 TP= 2116.0

Precision= 0.6897

Recall= 0.7528

F-measure= 0.7199

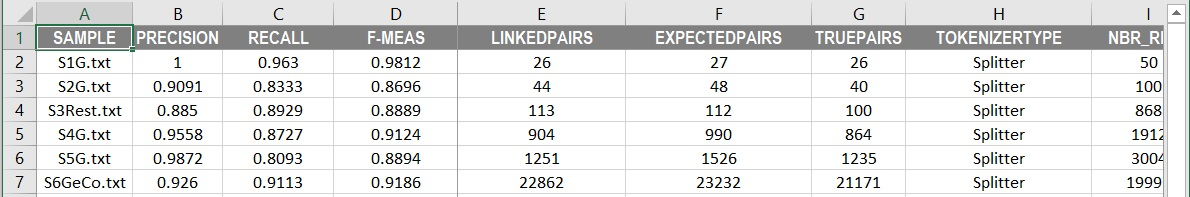
Total Runtime = 38.64404010772705

End of Program

### DWM100\_<date>\_<hour>\_<minute>.xlsx

This file is always produced from a successful run. It is created with the statistical output from the run along with the parameter settings for the run.

Spreadsheet output includes: Sample Name, Precision, Recall, F-measure, Linkedpairs, Expectedpairs, Truepairs, Tokenizertype, Nbr\_Refs, Total\_Tokens, Unique\_Tokens, Unique\_Ratio, Total\_Numeric, Numeric\_Ratio, Min\_Freq, Max\_Freq, Avg\_Freq, Stdev\_Freq, Min\_Len, Max\_Len, Avg\_Len, Stdev\_Len, Run\_Global\_Correction, Min\_Freq\_Std\_Token, Min\_Len\_Std\_Token, Max\_Freq\_Err\_Token, Beta, Block\_By\_Pairs, Min\_Blk\_Token\_Len, Exclude\_Numeric\_Blocks, Sigma, Remove\_Duplicate\_Tokens, Remove\_Excluded\_Blk\_Tokens, Epsilon, Epsilon\_Iterate, Mu, Mu\_Iterate, Comparator, Matrix\_Num\_Token\_Rule, Matrix\_Initial\_Rule, Blockcorrection



# Appendix A: Release Notes

## Release 1.7

* Added new parameter “minBlkTokenLen”
* Fixed bug in Global Correction (DWM25\_Global\_Token\_Replace) where DWM\_WordList was being read and used incorrectly.
* Added data/time to log file

## Release 1.8

* Added new parameter “excludeNumericBlocks”
* Added new parameter “removeExcludedBlkTokens”
* Add elapsed time to the output and log file
* Added default values for parameters to be backward compatible with Release 1.7
* Fixed “divide by zero” error when the parameter settings fail to link any references

## Release 1.9

* Added new parameter “blockByPairs”
* Made substantial changes to blocking logic of the program

Module DWM42\_BuildBlockPairs replaces several modules in Version 1.8. The inputs for DMW42 are the tokenized references in the dictionary “refDict” and frequencies of the tokens in token frequency dictionary “tokenFreqDict”. The output from DWM42 is a list of unduplicated reference ID pairs (“blockPairList”) representing all possible comparison between references across all blocks.

Step 1 of DWM42 builds the blocks either by single tokens or pairs of tokens according to the parameter “blockByPairs. Step 2 iterates over all pairs in each block and generates a list of reference ID pairs representing the pairs of reference to be compared in each block. Step 3 sorts and deduplicates the reference ID pairs to produce a single, unduplicated list “blockPairList” of reference ID pairs across all blocks. For example, if Ref2 and Ref4 are to be compared in say the “JOHN” block and also in a separate “TALBURT” block, the pair (Ref2, Ref4) will only occur once in blockPairList, the deduplicated reference ID pair list produced by DWM25.

Another important change is related to stop word removal. In previous versions, stop words were permanently removed from references during the blocking process. In Release 1.9, stop words are removed during the linking process in module DWM55\_LinkBlockPair. In addition, stop words are not permanently removed from the references stored in the reference dictionary “refDict”. Instead, the stop words are removed from a copy of the references sent to the comparator. The tokens in the references stored in “refDict” are only changed by the token correction processes. In Version 1.9, only Global Correction process in module DWM25\_GlobalTokenReplace has been implmented. However, future versions will incorporate corrects processes during the blocking phase and the cluster phase.

Another change implemented in Version 1.9 is using a dictionary “linkIndex” to manage the “good cluster” versus “bad cluster” separation process. In prior versions, the bad cluster references and good cluster references were written out to separate text files. In Version 1.9, this process is managed by a dictionary “linkIndex” where the key is a reference ID, and its value is the cluster ID. However, at the start of program, all cluster IDs are set to Null (‘’). During each iteration, if a good cluster is identified, the dictionary entry in “refDict” for each reference in the good cluster is update with the value of the cluster ID. At the start of each iteration, the “linkIndex” dictionary is scanned and only references with null cluster IDs are selected for reprocessing.

## Release 2.0

* Added new parameter “runIterationProfile”
* Added new statistic for count and ratio of numeric (all-digit) tokens
* Added new statistics for token length distribution

## Release 2.1

* Added new parameter “blockCorrection” for new DWM45\_Block\_Cleaning and a change to DWM42 for initiation of the block cleaning process.
* Added new DWM100\_ReportData for reporting with a spreadsheet (.xlsx) output of run statistics and all parameter files settings.
* Changes to DWM25 and DWM80 for faster processing of data.

## Release 2.2

* Added 3 new parameters “blockCorrectionDetail”, “globalCorrectionDetail”, and “addRefsToLinkIndex. When set to True, blockCorrectionDetail and globalCorrectionDetail cause the details of corrections to be written to the run log, but not to the user terminal. When False, changes the details of changes are not captured.
* Eliminated separate output files for changes made by global correction and block correction processes
* Changed block correction process logic to rebuild blocks and pair list after block corrections.
* Moved the call to DWM45 block correction from DWM42 module to the DWM00 driver.
* Changed logic to only write the final results to the DWM100 spreadsheet regardless of runIterationProfiles setting. In Version 2.1, if the runIterationProfiles=True, intermediate results were recorded in the DWM100 spreadsheet.
* Setting addRefsToLinkIndex to True will add the tokens of the references to be added to the Link Index output so that each line contains RefID, ClusterID, t1 t2 t3 … (list of tokens for RefID)

## Release 2.21

* This version does not add any new features. It corrects two bugs found in Release 2.2
* DWM25\_Global\_Token\_Replace was not returning the corrected references to the main program DWM00\_Driver. This is corrected.
* DWM90\_IterateClusters was sending references to DWM95 where the references were being modified in the quality assessment. Now DWM90 only sends a copy of each reference to DWM95 so that the original reference is not changed.

# Appendix B: Parameter File Template

#######################################

# Parameter File TEMPLATE

#######################################

# inputFileName must a valid path to file of entity references

# No default value, this parameter must be specified

**inputFileName=???**

# delimiter the field sparator character, must be comma,

# semi-colon, colon, pipe character, or tab character.

# Default value is comma

**delimiter=???**

# hasHeader must be True or False

# If True, the first line of the file is not processed

# If False, all lines of the file are processed as references

# Default value is True

**hasHeader=???**

# tokenizerType must be 'Compress' or 'Splitter'

# Compress replaces non-word characters with null

# Splitter replaces non-word characters with space

# Default value is "Splitter"

**tokenizerType=???**

# truthFileName Optional Parameter

# if provided, must be an existing text file in valid format

# Default value null

**truthFileName=???**

# runIterationProfile Optional Parameter

# if True, outputs cluster profile & F-meas for each iteration

# Default value False

**runIterationProfile=???**

# addRefsToLinkIndex Optional Parameter

# If True, the complete reference is included in the link index

# If False, the link index only includes the refID and clusterID

# Default value False

**addRefsToLinkIndex=???**

########################################

# Global Correction Parameters (OPTIONAL)

# runGlobalCorrection must True or False

# If True, global correction will run prior to blocking

# Default value False

**runGlobalCorrection=???**

# Global Correction Details (OPTIONAL)

# globalCorrectionDetail must True or False

# If True, global corrections are printed to logFile

# Default value False

**globalCorrectionDetail=???**

# minFreqStdToken must integer value > 0

# specifies the lowest freq of correction token

# Default value 5

**minFreqStdToken=???**

# minLenStdToken must integer value > 0

# specifies the minimum length of a correction token

# Default value 3

**minLenStdToken=???**

# maxFreqErrToken must integer value > 0 and < minFreqStdToken

# specifies the minimum length of a correction token

# Default value 3

**maxFreqErrToken=???**

########################################

# Block Correction Parameters (OPTIONAL}

# blockCorrection must be True or False

# indicates if block corrections will occur

# Default value is False

**blockCorrection=???**

# blockCorrectionDetail must be True or False

# indicates if details of block corrections will be logged

# Default value is False

**blockCorrectionDetail=???**

########################################

# Blocking Parameters

# beta must be >= 2 and < sigma

# indicates the maximum frequency of a blocking token

# Default value is 2

**beta=???**

# blockByPairs must be True or False

# If True, 2 refs must share same 2 tokens to be in same block

# If False, only have to share 1 token to be in same block

# Default value = True

**blockByPairs=???**

# minBlkTokenLen must be integer value > 0

# min length of a token to use for blocking

# Default value 4

**minBlkTokenLen=???**

# excludeNumericBlocks must be True or False

# If True, all digit tokens are not used for blocking

# Default value True

**excludeNumericBlocks=???**

########################################

# Stop Word Parameters

# sigma must be an integer value > beta

# all tokens with freq > sigma are removed before matching

# Default value 12

**sigma=???**

# removeDuplicateTokens must be True or False

# If True, duplicate tokens in same reference are removed

# Default value False

**removeDuplicateTokens=???**

# removeExcludedBlkTokens must be True or False

# If True, tokens excluded from blocking are removed from ref

# Default value True

**removeExcludedBlkTokens=???**

########################################

# Linking Paramters

# mu must be decimal value between 0.0 and 1.0

# starting value for match threshold to link two references

# Default value 0.5

**mu=???**

# muIterate must be decimal value between 0.0 and 1.0

# value added to mu at the end of each iteration

# Default value 0.10

**muIterate=???**

# comparator must be the name of a valid comparator

# 'Cosine','MongeElkan','ScoringMatrixStd', 'ScoringMatrixKris'

# Default value ScoringMatrixKris

**comparator=???**

# matrixNumTokenRule must be True of False

# If True, requires exact match between two numeric tokens

# applies only to ScoringMatrixStd and ScoringMatrixKris

# Default value False

**matrixNumTokenRule=???**

# matrixInitialRule must be True of False

# If True, requires exact match if either token has length 1

# applies only to ScoringMatrixStd and ScoringMatrixKris

# Default value False

**matrixInitialRule=???**

############################

# Cluster Quality Parameters

# epsilon must be decimal value between 0.0 and 1.0

# only clusters with quality >= epsilon are kept each iteration

# Default value

**epsilon=???**

# epsilonIterate must be decimal value between 0.0 and 1.0

# value added to epsilon at the end of each iteration

# Default value 0.0

**epsilonIterate=???**

#############################