A Method for Implementing Probabilistic Entity Resolution

Awaad Alsarkhi, John R. Talburt
Department of Information Science
University of Arkansas at Little Rock
Little Rock Arkansas, 72204, USA

***Abstract*—Deterministic and probabilistic are two approaches to matching commonly used in Entity Resolution (ER) systems. While many users are familiar with writing and using Boolean rules for deterministic matching, fewer are as familiar with the scoring rule configuration used to support probabilistic matching. This paper describes a method using deterministic matching to "bootstrap" probabilistic matching. It also examines the effectiveness three commonly used strategies to mitigate the effect of missing values when using probabilistic matching. The results based on experiment using different sets of synthetically generated data processed using the OYSTER open source entity resolution system.**

Keywords—Entity resolution; probabilistic matching; deterministic matching; boolean rules; scoring rule; missing values

I. Introduction

The OYSTER open source entity resolution system **(https://bitbucket.org/oysterer/oyster/)** was designed to support both deterministic and probabilistic matching. Probabilistic matching is performed using a scoring rule based on agreement and disagreement weights [18]. Generating the weights typically requires more effort and analysis than using Boolean rules to implement deterministic matching. The scoring rule is similar to the Boolean rule which specifies a similarity function and optional data preparation function for comparing the values of identity attributes between two entity references [2], [3], [30]- [34]. The linking and review decisions for each pair of entity references is determined by setting up a score threshold. Pairs scoring at or above the threshold are linked, and those below are not linked. Moreover, pairs scoring below the threshold, but above a user defined "review score" are written to an exceptions report for clerical review and possible remediation.

Each identity attribute has at least one agreement weight and at least one disagreement weight. In some cases, there can also be a third weight called a "missing weight" to be used instead, of the agreement or disagreement weight in the case both values of the identified attribute are missing.

Basically, the power of the probabilistic matching is the ability to assign weights to particular values of identity attributes. Typically, individual weights are only assigned when the identity attribute has an uneven distribution of value frequencies, and then, only to the values with the highest frequencies, e.g. the top 10% of frequencies. Value-frequency weights are less effective for identity attributes with an even distribution of value frequencies low frequencies such as dates-of-birth or telephone numbers.

For example, in the context of a school system, if many different students have the first name "Mary," then the weight of agreement for "Mary" should be smaller than the agreement weight for a name used by only a few students, perhaps even one student. In general, the magnitude of the weight is proportionate to its ability to discriminate between references to different students. Weight calculations are discussed in detail in a later section of the paper.

Another advantage of the scoring rule is its natural support for the "clerical review" or "link remediation" process. This is a process by which pairs of references with minimal support for linking are reported for manual review by domain experts. This is easily done when using the scoring rule by simply flagging any pair of references with a total score very close to the match threshold, i.e., "near matches" for review. Although very labor intensive, the clerical review is often implemented in high-risk applications where high-levels of linking accuracy are required, such as patient medical records.

The organization of this paper is as follows:

- Section II discusses prior work related to this study.
- Section III describes the "bootstrap" method for creating a scoring rule.
- Section IV describes the calculation of the weights and three missing value treatments approaches based on that database by applying run of OYSTER is configured invariants algorithm designed for feature extraction.
- Section V gives the experimental results.
- Section VI compares results from Boolean Rules and Scoring Rule
- Section VII and VIII summarizes the conclusions of this work.

II. E Literature Review

Numerous systems have been used to solve some data quality issue can problems weaken the effectiveness of probabilistic matching [4], [5], [6]. One of the influential methods is OYSTER algorithms. While OYSTER does include the capability to implement probabilistic matching, many users are not familiar with its setup and operation. The purpose of

this paper is to provide a step-by-step guide for users interested in experimenting with probabilistic matching using OYSTER. Particularly, the matching outcome of the scoring rule is lower when the sources have a high level of inconsistent value representations, such as misspellings, aliases, especially missing values. It should be noted here, while probabilistic matching has the ability to significantly improve record linking accuracy, it is not absolute. Certain data quality problems in the reference data can impair the effectiveness of probabilistic matching [10], [11], [19], [20]- [24]. In particular, the problems of inconsistent representation of identity attribute values and missing values have the most impact.

```
<ScoringRule Ident="Example1" MatchScore ="12" ReviewScore="11">
    <Term Item = "Fname" SIMILARITY="SoundexOrNickName"
AgreeWgt="11.27"
WgtTable="/Users/Alassad/Desktop/Oyster/LScoringIndex/Input/WgtTabl
eFirst.txt" DisagreeWgt="-8.30" />
    <Term Item ="Lname" SIMILARITY="Exact" DataPrep="Scan(LR,
Letter, 0, ToUpper, SameOrder)" AgreeWgt="4.20"
WgtTable="/Users/Alassad/Desktop/Oyster/LScoringIndex/Input/WgtTabl
eLast.txt" DisagreeWgt="-19.93"/>
    <Term Item="Nbr" SIMILARITY="Exact" DataPrep="Scan(LR,
DIGIT, 0, KeepCase, SameOrder)" AgreeWgt="10.42" DisagreeWgt="-
2.96"/>
    <Term Item="DOB" SIMILARITY="Exact" DataPrep="Scan(LR,
DIGIT, 0, KeepCase, SameOrder)" AgreeWgt="5.94" DisagreeWgt="-
1.31"/>
</ScoringRule>
```

Fig. 1. Example Attributes Script for Scoring Rule.

Figure 1 shows an example Attributes Script defining three identity attributes “Fname” (student first name), “Lname” (student last name), DOB (Date of Birth), and “Nbr” (street number of student address). Each run of OYSTER is configured by three XML Scripts: Run Script, Source Descriptor, and Attributes Script. The Run Script sets the overall purpose of the run called the “Run Mode.” The Run Modes for OYSTER are

- Merge-Purge
- Identity Resolution
- Identity Capture
- Identity Update
- Reference-to-Reference Assertion
- Structure-to-Structure Assertion
- Reference-to-Structure Assertion
- Structure Split Assertion

The two Run Mode configuration relevant to this paper are Merge-Purge and Identity Capture [3]. Both read a set of entity references and link the references together according to the match rule (either Boolean or Scoring) specified in the Attributes Script and transitive closure. The only difference is that Identity Capture also creates an Identity Knowledgebase whereas Merge-Purge does not. Identity Capture is the first step in using OYSTER for identity management applications such as MDM [5], [9], [13], [16].

The Source Descriptor Script is used to define the source and layout of the entity references to be linked by the system. The Source Descriptor can describe several different file sources including comma separated value (CSV) and fixed-width field formats. Because the focus of this paper is on the configuration of the Scoring in the Attributes Script, we will always assume the Run Mode is Identity Capture and the reference source is CSV.

III. Generating Weights and Configuring the Scoring Rule

A. Scoring Rule Configuration

The Attributes Script has two primary functions. The first is to designate the identity attributes in the entity references being processed. The second is to define how the identity attributes are to be matched. OYSTER currently supports two types of matching definitions, Boolean rules for deterministic matching and the scoring rule for probabilistic matching [7], [8]. The Scoring Rule configuration defines the Match Score to be 5 and the Review Score to be 4. All three of the identity attributes are used in the scoring rule. The first <Term> of the rule defines the agreement (“Similarity”) as “SoundexOrNickName”. This means that two first names will be in agreement if they generate the same SOUNDEX code (e.g. “Philip” and “Phillip”) or if they are nicknames of each other (e.g. “Robert” and “Bob”).

The first <Term> also defines a weight table for frequently occurring first name values named “WgtTableFirst.txt”. The weight table contains a set of key-value pairs. The key is the attribute value, e.g. “Mary”, and the value is the corresponding agreement weight, e.g. “4. In the weight table, the key and value must be separated from each other by a single tab character. When a weight table is given, the OYSTER logic is as follows. If the values agree, then the table is searched for both attribute values. Note: the lookup operation for attribute values in the table is not letter case sensitive. If both attribute values are found in the table, then it uses the larger of the two agreement weights. If only one value is found, it uses the one agreement weight found. If neither value is found, then it uses the default agreement weight given in the <Term> element. In this case, the default agreement weight is “9.08”. Of course, if the attribute values disagree, then in the <Term> element the disagreement weight of “2.45” is used.

The second <Term> element for “Lname” operates in much the same way except that it uses a “DataPrep” function in addition to a Similarity function. The DataPrep function is a data transformation that is applied to the raw attribute value and the result of the DataPrep transformation is passed to the Similarity function. The DataPrep function used here is the “SCAN” function. The parameters of the SCAN direct it to search the characters of the attribute value from left-to-right (LR), extract only letters (Letter), extract all letters found (0), change all letters to upper case (ToUpper), and keep the letters in the same order asthey were found (SameOrder). For example, “O’Malley” and “OMalley” would both transform to “OMALLEY”.

An important note about the use of weight tables. If a DataPrep function is used in the <Term>, then the lookup key

used for the weight table is the output of the DataPrep transformation function. If a DataPrep function is not used, then the lookup key is the raw, untransformed attribute value from the input file, but ignoring case. It is important to understand this logic to properly construct an agreement weight table. For example, if a <Term> were to define the DataPrep function as "SOUNDEX", then the keys defined in the weight table for this <Term> should be SOUNDEX output hash codes, not the input names. So, if the input value is "John" and the DataPrep is SOUNDEX, then the system will try to lookup for "J500" the SOUNDEX hash of "John" in the table. On the other hand, if DataPrep is SCAN (Letters to uppercase), then the system would try to lookup "JOHN" in the weight table [12], [15], [29].

B. Selecting Identity Attributes

The first step in selecting identity attributes is to profile the data. The primary candidates for identity attributes are those that have a high completeness and high uniqueness score. Attributes with low completeness or low uniqueness can be used as a secondary identity attribute to support the primary attributes.

The accuracy of the scoring rule is lower when the sources have high levels of inconsistent value representations (misspellings, aliases, etc.) and missing values. Also be sure to watch for placeholder values. These can bias the weight calculations.

The best candidates for value-level weights are identity attributes with an uneven distribution of the value frequencies, e.g. like names where several have high frequencies and many other have low frequencies. Attributes with evenly distributed values are best assigned only attribute-level weights

C. Linking the Records Using Another Process

The calculation of the agreement and disagreement weights follows the method developed by Fellegi and Sunter (1969). In their method, the agreement weight is based on the ratio of two conditional probabilities. The probability equivalent references will agree on the attribute or attribute value divided by the probability non-equivalent references will agree on the attribute or attribute value. The problem is that we don't know which references are equivalent, i.e. are referencing the same entity. In fact, if it were already known which references were equivalent and non-equivalent then there would be no purpose in setting up the matching rules.

One solution to this problem it to manually determine the correct linking for some subset of the data, i.e. create a "truth set". However, this can be exact costly in terms of time and effort. A second solution is to approximate the true linking with another ER process. For this paper, we follow the second solution by running OYSTER with a set of Boolean (deterministic) rules. The clusters created by this process are then assumed to be the equivalent references for the purpose of calculating the agreement and disagreement weights.

D. Calculating Weights

From cluster profile of the Boolean linking process (found in the OYSTER .log file), calculate the total number of equivalent pairs and total number of non-equivalent pairs.

This is done by taking each line of the cluster profile and calculating the number of pairs based on the cluster size. For example, if the profile has 8 clusters of size 5, then each cluster of size 5 represents (5x (5-1)/2 = 10 equivalent pairs

Next multiple the number of pairs in each cluster by the number of clusters. For example, if there are 10 cluster of size 5, then 8 x 10 = 80 equivalent pairs

Sum these count for each line in the cluster profile to get the total number of equivalent pairs. Let E represent this number.

Calculate the total number of possible pairs by taking the file size (total number of records) N and calculating N x (N-1)/2. Let T represent this number. For example, if the file contains 100 records, the T = 100x99/2 = 4,950 pairs

Calculate the total number of non-equivalent pairs by subtracting the total number of equivalent pairs found from the cluster profile from the total number of possible pairs. Let U represent this number, i.e.

U = T – E.

Next, join the Link File output from the OYSTER Boolean rule output (. link file) back to the source data by record identifier. This will create a new file called the weight analysis table. Each row of the weight analysis table should have a column for the

- Record identifier
- OYSTER identifier
- Value of each identity attribute in the record

Next, select the first identity attribute to process. Create a new column in the weight analysis table to hold the standardized value of the first attribute. Standardization is the process of removing variation from the values of the selected attribute. At a minimum you should convert all names to upper case letters so that "ANN" and "Ann" will come together.

If you want to remove even more variation, you could replace the name with the SOUNDEX code, NYSIIS, or other phonetic hash code. If you are planning to use a weight table, please read the previous section on Weight Table Logic. The setup of your weight table depends upon whether you use SOUNDEX for DataPrep or for Similarity. For example, SOUNDEX would make "ANN" and "ANNE" group together. If you used SOUNDEX for the Similarity, then you still have to keep the original name value together with its standardized or hashed value to populate the weight table. In the current implementation of the Scoring rule, if you use DataPrep, it looks up the DataPrep value in the Table, but if you don't use DataPrep it looks up the source value while ignoring case. So if you are calculating value-level weights using SOUNDEX or some other hash, it will be simpler if you use the hash function as the DataPrep so that the keys in the weight table are also hash values.

After creating the column of standardized values for the attribute in the weight analysis table, do a primary sort of the table by the standardized value of the attribute, and a secondary sort by the OYSTER identifier. This will give you groups of

consecutive rows with the same standardized value, and at least one subgroups with the same OYSTER identifier. If it only has one subgroup, then the group and subgroup are the same.

For the first subgroup, count the number of records having the same standardized value and same OYSTER identifier. Call this number A1V1N1 for Column 1, Value 1, Number of rows in Subgroup 1.

Now, calculate the number of equivalent pairs agreeing on standardized value V1 for this subgroup. Call this number C1V1E1 meaning Column 1, Value 1, Equivalent pairs in Subgroup 1. A1V1E1 is calculated by

A1V1E1 = A1V1N1 x (A1V1N1-1)/2

If there is more than one subgroup for standardized value V1, repeat this calculation for each subgroup of V1. If there is a second subgroup, then C1V1N2 would represent Value 1, Number of rows in Subgroup 2. The second subgroup would contribute another

C1V1E2 = C1V1N2 x (C1V1N2-1)/2

Equivalent pairs agreeing on V1.

Calculate the total number of equivalent pairs agreeing on V1 (call this number C1V1E) by summing the equivalent pairs for all subgroups, i.e.

C1V1E = C1V1E1 + C1V1E2 + ….

Next calculate the total number of pairs agreeing on the first standardized value V1 in Column 1 by counting the number of rows having the same standardized value V1. Call this value C1V1N. Note V1N should be the same as the sum of count of each subgroup, i.e.

C1V1N = C1V1N1 + C1V1N2 + …

The total pairs agreeing on V1 (call this C1V1A) is calculated by

C1V1A = C1V1N x (C1V1N-1)/2.

Next calculate the number of non-equivalent pairs agreeing on value V1 (call this value C1V1U). This number is calculated by

C1V1U = C1V1A – C1V1E.

For example, suppose C1 is the first name attribute, and if C1V1 is the standardized value “JOHN” and there are 5 records with this values that standardize to “JOHN” such as “John” or other variations. Suppose there are 2 subgroups of this value. the first attribute C1 has been selected for attribute-level or value-level weights, we need to make some other calculations for the entire attribute.

The first is the attribute’s overall disagreement weight D. This is done by repeating the previous steps for all of the values of the attributes. In particular, we need to Sum all pairs agreeing on the values of C1, as

C1A = C1V1A + C1V2A + …

Sum all equivalent pairs agreeing on the values of C1, as

C1E = C1V1E + C1V2E + …

Calculate the number of non-equivalent pairs agreeing on the values of C1, as

C1U = C1A – C1E

Then the

Disagree Weight C1

= log2{(1 – (C1E/E)) x (U/(1 - C1U))}

The agreement weight for C1 will depend upon whether it will have value-level weights or only a single attribute-level agreement weights. If there is only one agreement weight for the entire attribute C1, then it is calculated as

Agree Weight C1 = log2{(C1E/E) x (U/(C1U)}

using the same numbers as described above.

However, in the case of value-level weights, then formula is the same, but the values of C1E and C1U should only be the sum of the pair counts for the values not used for individual weight calculations. In other words,

C1A = C1VxA + C1VyA + …

where Vx is a value of C1 not given an individual weight, and Vy is a value of C1 not given an individual weight and so on. Another way to look at this is to think of removing the frequent values from the weight analysis table and calculating the overall agreement weight for the remaining values.

The same process described above is repeated for every identity attribute.

E. Constructing the OYSTER Scoring Rule

Following the same example, as shown in Figure 1, each one of the attributes from the weight analysis table will be used to define a <Term> in the Scoring Rule. The attribute name will be the value for the Item in the <Term>

For the <Term>, define a DataPrep and Similarity function in alignment with the DataPrep and Similarity functions used in the Boolean rules. Even though the scoring rule will allow you to use the same attribute in more than one scoring rule <Term>, we recommend only using each attribute once to define a <Term> in the Scoring Rule with one Similarity Function and one optional DataPrep function.

If a scoring rule <Term> defines only attribute-level weights, then the agreement weight (AgreeWgt value) should be set to the overall agreement weight as calculated in the previous section. Similarly, the disagreement weight (DisagreeWgt value) should be set to the overall disagreement weight for the attribute.

A <Term> can also define an optional Missing weight. The Missing weight is added to the overall score instead, of the AgreeWgt or DisagreeWgt values whenever both of the values being compared are missing. Although the missing weight is often set to zero, it is possible to calculate a weight by treating “missing” as a value just as you would a normal name value.

While more formal research is needed to determine the best strategy for scoring missing values, there is some preliminary evidence the best strategy is to not assign a special value for

missing value agreements. If no Missing weight value is given, then the DisagreeWgt value will be used if either or both values of the attribute are missing.

If a scoring rule term uses value-level weights, then the AgreeWgt value should be set to the overall agreement weight for only the rows in weight analysis not used to calculate individual value weights. On the other hand, the disagreement weight is still calculated as the overall disagreement weight for all values of the attribute. In addition, a <Term> using value-level weights must use the value of WgtTable to define a path to the text file containing the agreement weights to be used for the frequently occurring values. The weight table is simple. Each row should only have an attribute value followed by only ONE-tab character followed by the agreement weight. Any rows starting with two exclamation marks (!!) are ignored as comments.

In addition to setting the parameters for each <Term> item, the match score (MatchScore value) must be set at the <ScoringRule> level. An optional review score (ReviewScore value) value can also be set. The review score is a value less than match score. When a pair of references generate a score less than the match score, but greater than or equal to the review score, the two references will be written to a Clerical Review report. The purpose of the clerical review is to allow the user to see the pairs of references scoring just below the match score and determine if they are true negative pairs or false negative pairs.

If the review score is not given, then no clerical review report will be produced. If a review score is given, it should not be set too far below the match score. If too many pairs are written to the Clerical Review Report, it may significantly degrade the performance of the system. The goal is to only produce and review pairs of records scoring close to the value of the match score.

On exception to this rule is when using the Clerical Report for analysis and quality assurance. By running the scoring rule with a small test file, setting the match threshold actual high, and the review score low, every pair of references will be written to Clerical Review Report. This allows the user to analyze and check the scores being generated for every pair of references in the test file.

F. Setting the Optimal Match Score

The final parameters needed to create a complete scoring rule are setting an optimal match score. The value assigned to the MatchScore attribute in the <ScoringRule> term determines the linking performance of the system.

As the match score increases, the scoring rule will link fewer references resulting in higher precision and lower recall. Conversely, decreasing the match score will link more references resulting in lower precision and higher recall. At the extremes, if the match score is set so high that no pair of records can be linked, the precision will be 100% but the recall will be 0%. Conversely, if the match score is set so low that all records are linked, the precision will be 0% and the recall will be 100%. In both cases, the F-measure will be 0.0.

The balance between the precision and recall is the F-measure calculated as the harmonic mean of the precision and recall values. For some applications, the precision may be more important than recall, and the optimal threshold may not be the value producing the highest F-measure, but instead, the necessary level of precision. Other applications may favor recall over precision. For research described here, the threshold value producing the highest F-measure is considered optimal.

The review score and Clerical Review Report can be important tools in adjusting the value of the match score. It allows the user to examine pairs of records scoring just below the match score and possibly provide evidence for decreasing or increasing the match score value.

There is no formula for setting the Match Score. It is typically adjusted by trial and error. To start, review the output of the Boolean rule process and select a cluster containing several records, then extract these records from the source file to create a new test file containing only the records from one cluster. As described earlier, set the match score high and the review score low. Also, omit the Index so that every pair of records in the input will be compared, scored, and written to the Clerical Review Report.

The scores from the report can be used as the starting point for setting the match score. The reasoning is as follows: If the Boolean rules linked these records into a single cluster, then the scoring rule should link these records as well. Estimate the starting match score as the average of all the scores generated by the cluster. To improve the starting estimate, repeat the same process for other clusters linked by the Boolean rules, and include these scores in the average.

The reason for starting with the average score and not the minimum score is that not every pair of records in a cluster necessarily match. Some records come into the cluster through transitive closure. So even if a cluster contains records A, B, and C. It may be because A matched with B, and B matched with C, while it may not be true that A matched with C. Setting the match score low enough for A and C match in the scoring rule may create a large number of false positive links. A better estimate is the average score.

G. Indexing the Scoring Rule

The scoring rule must have an index (inverted index blocking) when the reference source contains more than a few hundred records. Otherwise, the runtime will be unacceptable. The problem is that aligning the indices to a scoring rule is more complicated than aligning indices to a Boolean Rule. For Boolean rules, you can inspect each rule (AND clause), and as long as the rule uses at least one hash function comparator, then it is possible to design an index in alignment with the rule.

In general, indices for Scoring Rules are less precise than for Boolean Rules. Rather than a single index on First-Name + Last-Name+ Street-Number, you might instead, have three separate indices, one for First-Name, another index for Last-Name, and an another for Street-Number. The downside to the strategy of indexing each attribute separately is that the list of match candidates will be larger for each input reference, causing more comparisons, and slower performance.

Another method for designing the index is a more complex analysis. If the scoring rule uses N attributes, then generate all possible combinations of agreement and disagreement weights as N-dimensional vectors of ordered pairs. The ordered pair for each dimension would represent a combination of Agree or Disagree together with the corresponding weight. For example, if there are three attributes, the vectors would look like [(A, 120), (D, -30), (A, 50)] where (A, 120) represents agreement on the first attribute with a score of 120, (D, -30) represents a disagreement on the second attribute with a score of -30, and (A, 50) agreement on the third attribute with a score of 50.

As you can see, there can be a great many of these vectors when there are value-level weights. For example, if the first <Term> has a table of 20 value agreement weights, plus the default agreement, and default disagreement. This would be a total of 22 possible weights for the first attribute. If these second attribute has 32 weights, and the third attribute has 3 weights, then there are 22 x 32 x 3 = 2,112 possible weight vectors.

Next, compute the total score (sum of the weights) for each vector and classify the vectors into 2N categories according to agreement/disagreement patterns similar to those used by Fellegi and Sunter [1]. As in the previous example where N=3, then there would be 8 (23) pattern categories: “DDD”, “DDA”, “DAD”, “DAA”, “ADD”, “ADA”, “AAD”, and “AAA”. Next, create an analysis file by generating an ordered pair for each vector where the first element is the agreement pattern and the second element the total score. Using the previous example for the vector [(A, 120), (D, -30), (A, 50)] the total score is 140. This vector would produce the pair (ADA, 140).

Finally, sort all of the ordered pairs according to their match scores. For any given choice of a match score, such as the starting estimate, you can now see all of the agreement patterns above and below that score. Suppose in this example that for a selected match score, all of the patterns above this score are “AAD”, “DAA”, or “AAA”. From this information you can design two indices to support the scoring rule for this match score. One index is a combination of the first and second attributes and the second index a combination of the second and third attributes. From an indexing standpoint, the pattern “AAA” is redundant to the first two patterns of “AAD” and “DAA”, i.e. any candidate returned by indexing on all three attributes would also be returned by indexing on two elements. However, the index solution just described will only provide alignment when all of the attributes a hash function for comparison, such as SOUNDEX or SCAN(), which can also be used for indexing.

H. Assessing Performance & Adjusting Match Score

You can use the Review Score setting to generate the Clerical Review Report. This will let you examine “near” matches and decide if you need to lower the Match Score. However, this approach does not show you possible false positive linking of pairs above the Match Score.

IV. EXAMPLE

The following example gives more detail on the calculation of the weights used for the Scoring rule given in Figure 1 as well as the performance of its linking with various MatchScore settings as shown in Figure 4. The weight calculations and performance measures are based on a test file of 141,745 synthetic person references known as ListB. The ListB reference were created in previous research [24]- [28] in a way that the correct linking is known. This allows the precision and recall to be immediately calculated for any given ER process.

Even though the true linking for ListB is known, it was not used to calculate the probabilistic weights for this example. The true linking was only used to evaluate the precision and recall of the linking results. The weights were calculated by bootstrapping the scoring rule from Boolean linking as described earlier. Given a set of entity references in the real world, exactly which pairs of references are equivalent is not known. The process starts by selecting the identity attributes and linking the references with a reasonable set of Boolean (deterministic) rules to serve as a “surrogate truth set” for generating probabilistic weights. The OYSTER Attributes script used to define the starting Boolean rules is shown in Figure 2.

```
<OysterAttributes System = "IndexTestWith">
        <Attribute Algo="none" Item="Fname"/>
        <Attribute Algo="none" Item="Lname"/>
        <Attribute Algo="none" Item="Nbr"/>
  <Attribute Algo="none" Item="DOB"/>
<!-- -->
<Indices>
          <Index Ident="X1">
                  <Segment                                  Item="Lname"
Hash="SCAN(LR,LETTER,0,ToUpper,SameOrder)"/>
                  <Segment                                    Item="Nbr"
Hash="SCAN(LR,DIGIT,0,ToUpper,SameOrder)"/>
          </Index>
     <Index Ident="X2">
       <Segment                                  Item="Lname"
Hash="SCAN(LR,LETTER,0,ToUpper,SameOrder)"/>
       <Segment                                    Item="DOB"
Hash="SCAN(LR,DIGIT,0,KeepCase,SameOrder)"/>
     </Index>
</Indices>
<IdentityRules>
          <Rule Ident="1">
                  <Term                                  Item="Fname"
SIMILARITY="SOUNDEXORNICKNAME"/>
                  <Term                                  Item="Lname"
SIMILARITY="SCAN(LR,LETTER,0,ToUpper,SameOrder)"/>
                  <Term                                    Item="Nbr"
SIMILARITY="SCAN(LR,DIGIT,0,ToUpper,SameOrder)"/>
          </Rule>
          <Rule Ident="2">
                  <Term                                  Item="Fname"
SIMILARITY="SOUNDEXORNICKNAME"/>
                  <Term                                  Item="Lname"
SIMILARITY="SCAN(LR,LETTER,0,ToUpper,SameOrder)"/>
                <Term                                    Item="DOB"
SIMILARITY="SCAN(LR,DIGIT,0,ToUpper,SameOrder)"/>
          </Rule>
        </IdentityRules>
</OysterAttributes
```

Fig. 2. Boolean Rule Configuration for Processing ListB.

The Boolean rules shown in Figure 2 use First Name, Last Name, Street Number, and Date-of-Birth. The rules as configured here produced a linking results with a Precision of 0.9800, Recall of 0.5064, and F-Measure of 0.6677.

Figure 2 also shows the inverted index blocking used for this rule set. Note that the first index (X1) is in alignment with both Boolean rules (1 and 2). Any pair of references matching by either rule would also produce the same match key by this index. The two terms of the index correspond exactly to the second and third terms of both the rules. This is possible because hash functions such as SCAN () can be used as both a comparator (True if both hash values agree) and a match key generator (hashing one string into another string). On the other hand, the comparator function NICKNAME used in the second Boolean rule (Ident=2) is a true similarity function requiring two strings as input while producing a Boolean True or False output. Because similarity functions do not produce a hash output, they cannot be used for inverted indexing using hash keys.

Following the weight calculation steps outlined in the previous section, ListB was profiled and individual agreement weights were calculated for the most frequently occurring Fname and Lname values. In this case, the top 10% of the high frequency names were selected. Figure 3 shows some of the entries in the weight table for Lname. Note that many of the names were only given as initials as illustrated by the first entry "A", and the file also contained many typographical errors, e.g. "ADASM" a typing transpose of "ADAMS".

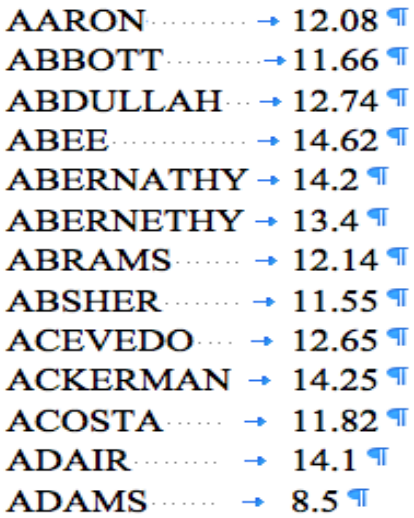

Fig. 3. Lname Agreement Weights.

Figure 4 also shows the performance of its linking with various MatchScore settings . After removing the top 10% of the most frequent first name values, the agreement weight for the remaining names was calculated as 11.27. The disagreement weight for all first name values was -8.30. Similarly, after removing the top 10% of last name values, the agreement weight for the remaining last names was calculated as 4.20, and the disagreement for all Last Name values was -19.93. Because the street number values and date-of-birth values were somewhat evenly distributed, only the overall agreement and disagreement weights were calculated these attributes. For street number the values were 10.42 and -2.96, respectively, and for date-of-birth the values were 5.94 and -1.31, respectively.

```
<OysterAttributes System="School">
  <Attribute Algo="none" Item="Fname"/>
  <Attribute Algo="none" Item="Lname"/>
  <Attribute Algo="none" Item="Nbr"/>
  <Attribute Algo="none" Item="DOB"/>
<!-- -->
<ScoringRule Ident="Example1" MatchScore ="12"
ReviewScore="11">
    <Term Item = "Fname" SIMILARITY="SoundexOrNickName"
AgreeWgt="11.27"
WgtTable="/Users/Alassad/Desktop/Oyster/LScoringIndex/Input/Wg
tTableFirst.txt" DisagreeWgt="-8.30" />
    <Term Item ="Lname" SIMILARITY="Exact"
DataPrep="Scan(LR, Letter, 0, ToUpper, SameOrder)"
AgreeWgt="4.20"
WgtTable="/Users/Alassad/Desktop/Oyster/LScoringIndex/Input/Wg
tTableLast.txt" DisagreeWgt="-19.93"/>
    <Term Item="Nbr" SIMILARITY="Exact"
DataPrep="Scan(LR, DIGIT, 0, KeepCase, SameOrder)"
AgreeWgt="10.42" DisagreeWgt="-2.96"/>
    <Term Item="DOB" SIMILARITY="Exact"
DataPrep="Scan(LR, DIGIT, 0, KeepCase, SameOrder)"
AgreeWgt="5.94" DisagreeWgt="-1.31"/>
</ScoringRule> <Indices>
    <Index Ident="X1">
      <Segment Item="Fname" Hash="SOUNDEX"/>
      <Segment Item="Lname"
Hash="SCAN(LR,LETTER,0,ToUpper,SameOrder)"/>
      <Segment Item="Nbr"
Hash="SCAN(LR,DIGIT,0,KeepCase,SameOrder)"/>
    </Index>
    <Index Ident="X2">
      <Segment Item="Fname" Hash="SOUNDEX"/>
      <Segment Item="Lname"
Hash="SCAN(LR,LETTER,0,ToUpper,SameOrder)"/>
      <Segment Item="DOB"
Hash="SCAN(LR,DIGIT,0,KeepCase,SameOrder)"/>
    </Index>
    <Index Ident="X3">
      <Segment Item="Fname" Hash="SOUNDEX"/>
      <Segment Item="Nbr"
Hash="SCAN(LR,DIGIT,0,KeepCase,SameOrder)"/>
      <Segment Item="DOB"
Hash="SCAN(LR,DIGIT,0,KeepCase,SameOrder)"/>
    </Index>
    <Index Ident="X4">
      <Segment Item="Lname"
Hash="SCAN(LR,LETTER,0,ToUpper,SameOrder)"/>
      <Segment Item="Nbr"
Hash="SCAN(LR,DIGIT,0,KeepCase,SameOrder)"/>
      <Segment Item="DOB"
Hash="SCAN(LR,DIGIT,0,KeepCase,SameOrder)"/>
    </Index>
  </Indices>
</OysterAttributes>
```

Fig. 4. Final Scoring Rule with Blocking.

V. Testing and Results

Table 1 shows the details of precision, recall and F-measure comparison between scoring rule results. Figure 5 is the line chart specifically compares the F-measure performance.

TABLE I. COMPARE THRESHOLD SCORE RULE RESULTS

	Score Rule		
Threshold	Precision	Recall	F-Measure
-20	0.79	0.72	0.75
-15	0.9	0.72	0.8
-10	0.91	0.72	0.8
-5	0.91	0.72	0.8
0	0.93	0.71	0.81
5	0.93	0.64	0.76
10	0.95	0.59	0.73
15	0.98	0.56	0.71
20	0.98	0.48	0.65

VI. COMPARISON OF RESULTS

Table 2 shows the scoring rule using weights computing from Boolean rule linking can outperform the initial set of Boolean rules. In this example, a 20% improvement in F-measure with significantly higher precision

TABLE II. SUMMERY OF RESULTS

	Rule- Generation		
	Precision	Recall	F-Measure
Boolean Rule	0.6677	0.5064	0.6677
Scoring Rule	0.93	0.71	0.8014

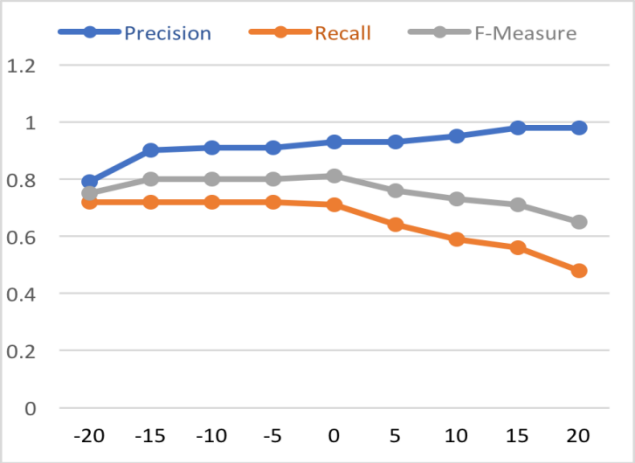

Fig. 5. Scoring Rule Performance for Different Thresholds.

VII. CONCLUSION AND FUTURE WORK

The bootstrap process described in this paper allows researchers to configure a probabilistic entity resolution engine using OYSTER by taking advantage of its ability execute both Boolean rules and the scoring rule with frequency-base weights. It also shows the amount of tuning required to get good results from probabilistic matching due to the large number of variable including the

- Choice of identity attributes
- Comparators used for the identity attributes
- Choice of identity attributes for frequency-based value weights
- Number of values selected for individual weights
- Match score selection
- Impact and treatment of missing values
- Impact and treatment of inconsistent values

At the same time, these variables create the opportunity for additional research including the use of both agreement and disagreement weights at the value level, analysis tools for setting optimal match scores, the treatments to mitigate the effect of missing values, and the optimal frequency cutoff for frequency-based value weights.

Another avenue of research is based on repeating the bootstrapping process. Just as the Boolean linking results can be used as a proxy for equivalence to generate weights for a scoring rule, the linking of the scoring rule itself can be used to generate a second set of weights, and so on. Will the weight generated from on scoring rules output allow the construction of another scoring rule that will always outperform its predecessor? [14], [17], [18].

REFERENCES

[1] Fellegi, I. P., & Sunter, A. B. (1969). A theory for record linkage. Journal of the American Statistical Association, 64(328), 1183-1210.

[2] Zhou, Y., Talburt, J.R., Kobayashi F., and Nelson E. (2012). Implementing Boolean matching rules in an entity resolution system using XML scripts. *The 2012 International Conference on Information and Knowledge Engineering (IKE'12)*, Las Vegas, Nevada, July 16-29, 2012 (pp. 332-337).

[3] Agichtein, E., & Ganti, V. (2004). Mining reference tables for automatic text segmentation. *Proceedings of the Tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 20-29). ACM.

[4] Christen, P. (2012). Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection. Springer Science & Business Media.

[5] Benjelloun, O., Garcia-Molina, H., Kawai, H., Larson, T. E., Menestrina, D., Su, Q., ... & Widom, J. (2006). Generic entity resolution in the serf project. Stanford InfoLab.

[6] Kobayashi, F., Eram, A., & Talburt, J. (2018). Entity resolution using logistic regression as an extension to the rule-based oyster system. *Proceedings 2018 IEEE Conference on Multimedia Information Processing and Retrieval* (MIPR) (pp. 146-151).

[7] Wang, P., Pullen, D., Talburt, J. R., & Wu, N. (2014). Probabilistic Matching Compared to Deterministic Matching for Student Enrollment Records. *Proceedings Information Technology: New Generations* (ITNG), (pp. 355-359).

[8] Kobayashi, F. & Talburt, J.R. (2013) Probabilistic Scoring Methods to Assist Entity Resolution Systems Using Boolean Rules, *The 2013 International Conference on Information and Knowledge Engineering (IKE'13)*, Las Vegas, Nevada, July 22-25, 2013, CSREA Press (pp. 101-107).

[9] Whang, S. E., & Garcia-Molina, H. (2010). Entity resolution with evolving rules. *Proceedings of the VLDB Endowment*, 3(1-2), 1326-1337.

[10] Bilenko, M., Kamath, B., & Mooney, R. J. (2006). Adaptive blocking: Learning to scale up record linkage. In Data Mining, 2006. *Proceedings Sixth International Conference on Data Mining* ICDM'06. (pp. 87-96).

[11] Elmagarmid, A. K., Ipeirotis, P. G., & Verykios, V. S. (2007). Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1), 1-16.

[12] Koybayashi, F. & Talburt, J.R., Oyster Version 3.6 Reference Guide, https://bitbucket.org/oysterer/oyster/downloads/

[13] Talburt, J. R., Zhou, Y., & Shivaiah, S. Y. (2009). SOG: A Synthetic Occupancy Generator to Support Entity Resolution Instruction and Research. *Proceedings International Conference on Data Quality* (ICIQ-09), pp. 91-105.

[14] Wang, P., Pullen, D., Talburt, J., & Wu, N. (2014) Iterative approach to weight calculation in probabilistic entity resolution. *19th MIT International Conference of Information Quality*, August 1-3, 2014, Xi'an, China, (pp. 244-258).

[15] Zhou, Y. & Talburt, J. (2011) The role of asserted resolution in entity identity information management, *Proceedings: 2011 Information and Knowledge Engineering Conference (IKE 2011)*, Las Vegas, NV, July 18-20, 2011 (pp. 291-296).

[16] Hashemi, R., Ford, C., Vanprooyan, T., & Talburt, J. (2002). Extraction of features with unstructured representation from HTML documents. In P. Isaias (Ed.), *International Association for Development of Information Society (IADIS) International Conference on WWW/Internet 2002* (pp. 47- 53). Lisbon, Portugal.

[17] Herzog, T. N., Scheuren, F. J., & Winkler, W. E. (2007). *Data quality and record linkage techniques*. Springer Science & Business Media.

[18] Hernández, M. A., & Stolfo, S. J. (1995). The merge/purge problem for large databases. *ACM Sigmod Record* (Vol. 24, No. 2, pp. 127-138).

[19] Pushkarev, V., Neumann, H., Varol, C., & Talburt, J. R. (2010). An Overview of Open Source Data Quality Tools. *Proceedings of the International Conference on Information and Knowledge Engineering* IKE (pp. 370-376).

[20] Talburt, J., Wang, R., Hess, K., & Kuo, E. (2007). An algebraic approach to data quality metrics for entity resolution over large datasets. In *Information quality management: Theory and applications* (pp. 1-22). IGI Global.

[21] Dong, X., Halevy, A., & Madhavan, J. (2005). Reference reconciliation in complex information spaces. *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* (pp. 85-96).

[22] J. Talburt, "Oyster v3.3 Reference Guide 2013 03 18: http://downloads.sourceforge.net/project/oysterer/OYSTER_3.3/. [Accessed 16 July 2015].

[23] Talburt, J. R. (2011). *Entity resolution and information quality*. Morgan Kaufmann.

[24] F. Kobayashi and J. R. Talburt, Decoupling Identity Resolution from the Maintenance of Identity Information. *Proceedings: International Conference on Information Technology* (ITNG), Las Vegas, NV, 2014. (pp. 349-354).

[25] Newcombe, H. B., Kennedy, J. M., Axford, S. J., & James, A. P. (1959). Automatic linkage of vital records. *Science*, 130(3381), 954-959.

[26] Zhou, Y. (2012). Modeling and design of entity identity information in entity resolution systems. *Doctral Dissertation*. University of Arkansas at Little Rock.

[27] Syed, H., Talburt, J., Liu, F., Pullen, D., & Wu, N. (2012, January). Developing and refining matching rules for entity resolution. *Proceedings of the International Conference on Information and Knowledge Engineering* (IKE) (p. 1).

[28] Talburt, J. R., & Zhou, Y. (2015). Entity information life cycle for big data: Master data management and information integration. Morgan Kaufmann.

[29] Pullen, D., Wang, P., Talburt, J. & Wu, N. (2014) Mitigating data quality impairment on entity resolution errors in student enrollment data. *11th International Conference on Information and Knowledge Engineering*, July 21-24, 2014, Las Vegas, NV, (pp. 96-100).

[30] Wang, R. Y., & Strong, D. M. (1996). Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, 12(4), 5-33.

[31] Talburt, J., Wu, N., Pierce, E., & Hashemi, R. (2007). Entity identification using indexed entity catalogs. In H.R. Arabnia & R.R. Hashemi (Eds.), *The 2007 International Conference on Information and Knowledge Engineering* (pp. 338-342). Las Vegas, NV: CSREA Press.

[32] Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S. E., & Widom, J. (2009). Swoosh: a generic approach to entity resolution. *The VLDB Journal—The International Journal on Very Large Data Bases*, 18(1), 255-276.

[33] Zhou, Y., Talburt, J. R., Su, Y., & Yin, L. (2010). OYSTER: A tool for entity resolution in health information exchange. *Proceedings of the 5th International Conference on Cooperation and Promotion of Information Resources in Science and Technology* (pp. 358-364).

[34] Wang, R.Y., 1998. A product perspective on total data quality management. *Communications of the ACM* 41 (2), 58-65.