The ChoiceMaker 2 Record Matching System

Andrew Borthwick
ChoiceMaker Technologies, Inc.
646-336-4442
aborthwick@choicemaker.com

ABSTRACT

This paper describes the key features of an innovative record matching system called ChoiceMaker 2 developed by ChoiceMaker Technologies (CMT). We begin with an overview of the stages that a record matching system goes through to find an incoming "query record" in a database. We then consider the stages one by one: We sketch out our patent-pending process for identifying possible matches to the query record, which is known as "blocking". We describe the process by which we use a machine learning technique known as maximum entropy modeling to tune the system to the problem at hand. Next we describe the ClueMaker® programming language that CMT has developed for describing record matching characteristics. We describe our method for testing record matching models and describe how our IDE facilitates this process. We describe the process by which we develop record matching models. Finally, we discuss systems integration issues and the interfaces that ChoiceMaker offers for deployment.

1. APPROXIMATE RECORD MATCHING

Approximate record matching is employed when information is not always identified by a reliable unique key. Record-matching tasks can be broken down into three main categories:

- Duplicate record removal or linkage: The same person, business, or thing is present more than once in a database. Duplicate records are removed or linked together.
- Database linkage: Two databases are linked or merged. This might occur, for instance, because of a corporate merger, to build a data warehouse, or to prevent duplication of effort by storing information common to multiple databases in a single enterprise-wide database.
- Approximate database search: Search a database for records similar to an input record. Similarly, prevent users from adding duplicate records to the database by providing a real-time check of whether a record entered on a user entry screen is present in the database.

In all of these cases the basic problem is essentially the same. Given an input or query record, search a target database for record(s) that denote the same thing (e.g., the same person, product, or company).

1.1 Matching Process Overview

Advanced approximate record matching systems generally perform matching as a two-step process, as illustrated in Figure 1:

1. *Query record.* A query record is sent to the matching engine.
2. *Blocking*. The engine searches the target database for records that are possible matches to the query record. The objective at this stage is to retrieve all possible matches and not too many non-matches.
3. *Many possible matches.* This is the set of records returned by blocking which are possible matches to the query record.
4. *Decision making*. For each possible match, the matching engine determines the probability that the record denotes the same thing as the query record. Possible matches are sorted into matches, potential matches, and non-matches based on two user-defined thresholds. I.e., any record matching the query record with a probability higher than the "match threshold" is declared a "match".

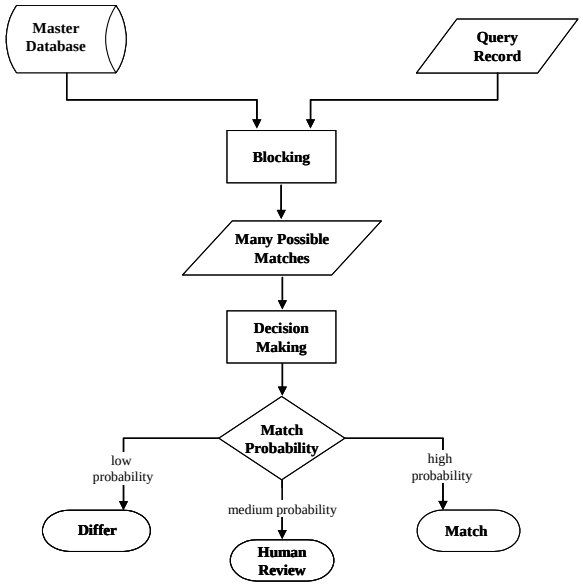

Figure 1: ChoiceMaker Matching Overview

2. BLOCKING

In the field of approximate record matching, a "blocking" step refers to a fast matching algorithm primarily used as the first step of a larger record matching system. The goal of a blocking step is to attempt to find all *possible* matches to an input query record rather than to aim for precision in determining which record is the correct match. Blocking thus aims for maximum "recall" or "sensitivity", possibly at the expense of achieving high "precision" or "specificity". In effect, blocking is a sort of "is it in the ballpark?" test that can be used to narrow down the number of records that need to be processed by the higher precision, but more computationally intensive "scoring" stage. In this second stage, ChoiceMaker uses the computationally intensive maximum entropy (ME) based model discussed below. The ME model tests every record returned by the blocking algorithm against the query record to see if they match.

© November 2004. ChoiceMaker Technologies, Inc. All Rights Reserved.

Blocking provides a huge performance improvement, because when deduplicating a database of n records, there are approximately $n^2/2$ pairs of records in the database. Hence, when n is of any magnitude, it would be time-prohibitive for the system to attempt to examine all pairs of records.

Traditional blocking methods are generally based on an *ad hoc* judgment of the usefulness of different fields in a matching decision. For instance, a healthcare site might use Medicaid and medical record number matches as blocking characteristics—meaning that any records matching on those two fields would be passed on to the second-stage matching algorithm. Also commonly used are matches on birthday and first name, birthday and last name, birthday and Soundex code of last name, etc.

This traditional approach can work reasonably well, but its *ad hoc* nature places a limitation on the portability of any system built around it. It also has a problem of generating too many false negative responses (i.e., records that should have been linked, but were not). The quality of the blocking routine is important to the ability of the system to minimize the number of false negatives since pairs that are not seen as possible matches in the blocking phase will be missed even if the second-stage matching algorithm's decision-making engine would have assigned them a high probability of match. At the same time, the system has to carefully manage tradeoffs between false negatives and run-time performance. If the blocking algorithm is too liberal in passing along hypothetical matches, system run-time may exceed the user's tolerance.

ChoiceMaker has developed a patent-pending approach to the blocking problem. This technique automatically builds a query that is as liberal as possible in efficiently retrieving records that match on individual fields or sets of fields while avoiding selection criteria that are predicted to return more than the maximum number of records. The ability to do blocking without extensive manual setup greatly simplifies the deployment of a record matching system. In addition, the algorithm has the advantage that response time is relatively predictable and independent of the data in the query record.

In contrast to the traditional approaches discussed above which block on fixed combinations of keys, ChoiceMaker's automated blocking algorithm (ABA) dynamically constructs a set of database SELECT statements which we call "blocking sets." Each blocking set is built to be as liberal as possible in retrieving records that match the query record on individual fields or sets of fields while not containing SELECT statements that are predicted to return more than a user-defined maximum number of records, *m*. The records retrieved are those retrieved by the union of the SELECT statements.

Let's consider two examples. If the query record contains the data "FirstName = Asa", "LastName = Segur", both of which are rare names, we would construct a query which selected records where "FirstName = Asa" OR "LastName = Segur". On the other hand, if the query record contains the data "FirstName = Jim" AND LastName = Smith", both of which are common names, then we would construct a query which selected records where "FirstName = Jim" AND "LastName = Smith".

Our algorithm assumes that, for any query record we might receive, we are able to determine the frequency of any value entered in any column used in blocking. This is accomplished by means of a "Counts" table which, for each blocking column, holds the count of every value occurring in that column with a frequency greater than *m*. This Counts table is updated periodically (perhaps every day or every few hours), but it does not need to be kept exactly accurate with respect the database because, if one assumes that the database will be gradually growing over time, we will expect the value counts on the Counts table to tend to be slightly low. This will make values look rarer than they really are, which will lead the algorithm to append fewer additional fields onto blocking sets containing these slightly underreported values. This will lead the algorithm to retrieve slightly too many records for the second matching stage, which will cause the algorithm to perform slightly more slowly than it otherwise would, but it will not decrease the algorithm's accuracy.

Given this Counts table, the list of blocking columns, B, and *m*, the maximum number of records that we want to retrieve with any given SELECT statement, the algorithm is as follows:

1. Receive a query record, *q*
2. Obtain from Counts, frequency counts for every value in q which is in a column in B
3. Create blocking sets
    a. Find all combinations of blocking values from q which are predicted to return fewer than *m* records:
        i. Compute the expected frequency of combinations of fields by combining field frequencies while assuming that values in different fields are independent
    b. Don't return any redundant blocking sets
4. Retrieve the union of the records that satisfy the query and return them for second stage scoring.

As an illustration, suppose we had the following query record:

Field	**Value**	**Field-value frequency**
FirstName	Keanu	[Not in Counts]
LastName	Reeves	20,000
City	Beverly Hills	30,000
Street	Locust Ln.	[Not in Counts]
Age	40	1,000,000

Table 1: Example query record with frequency of values

If we assume that the value of *m* is 100 and that the database has 200,000,000 records, then we end up with the following SELECT statements:

SELECT	**Expected frequency**	**Comment**
FirstName = "Keanu"	< 100	Not in Counts
Street = "Locust Ln."	< 100	Not in Counts
LastName = "Reeves" AND City = "Beverly Hills"	3	
LastName = "Reeves" AND Age = 40	100	

Table 2: SELECT statements generated by ABA

© November 2004. ChoiceMaker Technologies, Inc. All Rights Reserved.

Note that we do not select on Age = 40 AND City = "Beverly Hills" because it is expected to return 150 records. When we see that this two-field blocking set is not sufficiently selective, we attempt to add a third field. However, any field that we add would be redundant with a blocking set we have already created. FirstName and Street are already being used as single-field blocking sets. On the other hand, if we added LastName to the blocking set, then this blocking set would be redundant with other two two-field blocking sets because both already will return a superset of the records that this hypothetical three-field blocking set would return.

ChoiceMaker has two patent-pending versions of this algorithm. The real-time version described above is designed for situations where sub-second response time is required. It is not suitable for large databases. The batch version of this algorithm, which is the subject of a second patent application, performs a similar process to the real time and achieves the same level of accuracy, but gains great efficiency by blocking all the query records simultaneously. The batch version is used for deduplicating a database and matching a large dataset against an already deduplicated database. ChoiceMaker has used the batch blocking algorithm to deduplicate a database at a rate of over 800,000 records per hour on a typical high-end single-processor desktop with 1.5 GB of RAM.

3. MACHINE LEARNING AND MODEL DEVELOPMENT

ChoiceMaker's record matching models are built around a set of 50 – 200 "clues" (more commonly known in the AI community as "features"), which indicate a "match" or "differ" decision. Each clue reads the pair of records in question—generally the query record and a record from the database and determines whether a particular indicator of a match or non-match decision is present in this pair of records. If the clue's criteria for predicting a match or non-match are met, then we say that the clue is "active" on the pair of records. Some sample clues include:

- Are the first names the same and are they common, uncommon, or very rare?
- Do the last names have the same phoneticization according to Soundex [6] or similar techniques?
- Is the date of birth different?

These clues are written in ChoiceMaker's ClueMaker® programming language described in section 4.

Our ML approach [3] constructs a record matching model that outputs the probability that a pair of records represents the same entity. The model is trained on a set of pairs of records that have been tagged as a "match", "differ", or "hold (unsure)". Although we have experimented with other ML techniques, ChoiceMaker currently uses maximum entropy modeling (ME) [1] [2] in all of its production models. ME requires that each clue predict a decision (a "future" in the AI literature), in this case the predictions are "match" and "differ". The ME training process then assigns each clue a weight, which is a positive real number indicating the relative predictive strength of the clue. At run time, the weights of all active clues are combined to form a probability of match by the following formula:

$$\text{Probability} = \frac{\text{MatchProduct}}{\text{MatchProduct} + \text{DifferProduct}}$$

where MatchProduct is defined as the product of the weights of all clues predicting "match" for the pair and DifferProduct is the product of all clues predicting "differ" for the pair.

To give a simple example, consider the example in Table 3 where we are matching the search record "Jim Conner" to the potential match record "Jim Connor." Note here that "C560" is the phonetic code for both Conner and Connor according to Soundex. In this example, clues 1 and 3 activate predicting a "match" decision whereas clue 2 activates predicting a "differ" decision.

	Field	Query Record	Match Record	Clue name	Prediction	Weight
1	First name	Jim	Jim	First names match	Match	1.5
2	Last name	Conner	Connor	Last names differ	Differ	2.2
3	Soundex last name	C560	C560	Soundex last names match	Match	5.5

Table 3: Maximum entropy example

Putting this into the above formula gives us a probability of 79% that "Jim Conner" and "Jim Connor" are the same person:

MatchProduct = 1.5 * 5.5 = 8.25

DifferProduct = 2.2 = 2.2

Probability = 8.25 / (8.25 + 2.2) = 79%

Remember that the weights 1.5, 2.2, and 5.5 were determined during training by the maximum-entropy engine in an attempt to produce for as many examples as possible a decision that is consistent with the human marking. Note that the model most likely contains many additional clues which did not fire on this pair of records, such as "last names match," but which would have fired if their conditions were met.

Relative to other machine learning techniques, maximum entropy has proven to be a good choice for this problem because it is fast at run time, is relatively easy to explain to clients, and can assign accurate weights to multiple overlapping and interacting clues.

4. CLUEMAKER® PROGRAMMING LANGUAGE

4.1 Why a new programming language?

The introduction of a new programming language requires a strong justification as IT departments are understandably hesitant to support additional languages. We created ClueMaker [4] for the following reasons:

- A set of clues written in ClueMaker is roughly ten times shorter than the same set of clues written in Java.

© November 2004. ChoiceMaker Technologies, Inc. All Rights Reserved.

- Clues written in ClueMaker are more easily understood by customers
- Since ClueMaker contains many constructs specific to record matching it is less error prone than repetitious boilerplate code in Java.
- ClueMaker permits many code optimizations (such as common subexpression elimination and loop invariant code motion) that cannot be applied to Java programs because of side effects. The performance gain of these optimizations can often reach a factor of ten.

These benefits, coupled with the fact that ClueMaker gets compiled into Java should mitigate IT concerns. Note also that for clients who leave model development in the hands of CMT, IT staff needs little or no understanding of ClueMaker. For clients who want to build or maintain their own record matching models, only a small team of model developers needs to learn ClueMaker.

4.2 ChoiceMaker Schemas

The records to be matched are described by a ChoiceMaker schema, which is defined in an XML document. A schema defines the name and type of each field. It may also define a field's 'validity predicate', which is a Java expression that identifies valid values. For instance, a first name of "Boy" would not be valid, and a social security number would never begin "000". A schema can also define derived fields, which are computed from other values in the record. For example, the fields "city", "state", and "zip code" can be derived from an unparsed address.

4.3 ClueMaker and Java

ClueMaker extends Java with some powerful constructs that make it easy to write record matching clues. The ClueMaker compiler compiles ClueMaker code into Java.

An example clue is mFirstName, which is shown in the first 4 lines in Figure 2. The keyword **match** indicates that the clue predicts "match". The keyword **valid** takes a field-name argument and calls the field's validity predicate in the schema, which returns true if the field's value is valid. The two records being compared are always referred to as q (the input, or "query" record) and m (the possibly matching record retrieved from the database).

The expression q.firstName == m.firstName is simply a Java expression. These expressions can be arbitrarily complex. For instance, in dLastNameSoundex, we see a call to the Java method "soundex" in the Soundex class.

```
clue mFirstName {
     match  valid(q.firstName) && valid(m.firstName)
               && q.firstName == m.firstName;
}

clue dLastNameSoundex {
     differ valid(q.lastName) && valid(m.lastName)  &&
          Soundex.soundex(q.lastName) !=
          Soundex.soundex(m.lastName);
}
```

Figure 2: Example ClueMaker clues

Consequently, ClueMaker is a very easy language for a Java programmer to learn. The core of each clue consists of actual Java code. Complicated methods such as Soundex can be defined in standard Java. The ClueMaker language manual devotes just 25 pages to describing the novel features of the language.

4.4 ClueMaker features

ClueMaker does, however, contain many constructs that greatly facilitate writing clues. These include:

- The shorthand forms "same" and "different" which simultaneously check validity and whether the fields in question are the same or not
- A "swap" construct which checks whether a pair of fields have been swapped (e.g., first name in the last name field).
- An ability to work with "stacked data", schemas which hold multiple values for the same field. For instance, many databases allow multiple values for address, names, etc.

ClueMaker, along with ChoiceMaker's machine learning technology, allows the efficient construction of record matching models which are carefully optimized to solve the challenges of each client's database.

4.5 Complex Clues in ClueMaker

A key advantage to ClueMaker is that it allows model developers to concisely describe complex record matching criteria. For instance, when matching a nationwide database of names, addresses, and birthdays, we built a record matching clue which looked for "snowbirds", retirees who lived in the Northeast or Midwest in the summer and in Florida or Arizona in the winter. This is a strong indicator of a match condition which requires examining multiple fields simultaneously. This clue is illustrated in Figure 3, where we use a mapping from state to region of the country: Northeast, Midwest, South, and West are 1, 2, 3, and 4, respectively.

```
1  clue mSnowbirds {
2     match exists(i,j;
3       and(valid(r.info[i,j].state)) &&
4       and(r.info[i, j].age>=60) &&
5       xor(r.info[i, j].state == "FL" || r.info[I, j].state == "AZ") &&
6       xor(Maps.lookupInt("stateRegions", r.info[i,j].state) <=2));
7  }
```

Figure 3: "Snowbird" clue

Hence, ClueMaker allows the developer to represent this sophisticated piece of human intuition in a record matching model with just seven lines of code.

5. TESTING

After developing clues in ClueMaker for our record matching model and training it on hand-marked data using ME, we are ready to test the model on a separate corpus of hand-marked data

© November 2004. ChoiceMaker Technologies, Inc. All Rights Reserved.

on which the model has not been trained. This testing on unseen data is performed on data tagged by ChoiceMaker Technologies during the development process and then is performed by the client on data tagged by the client for a final test prior to deployment. This final test is designed to be a scientific evaluation of the system, to ensure that the record matching model has not been tuned to the peculiarities of the training and development test data. Our development process is illustrated in Figure 4.

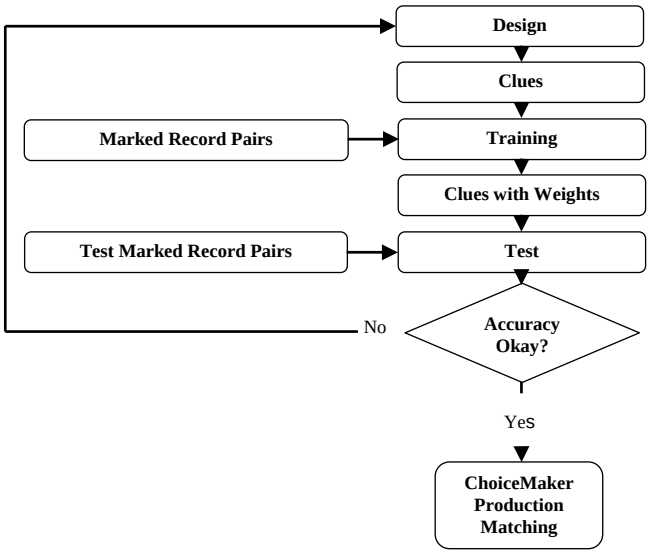

Figure 4: ChoiceMaker's model development process

ChoiceMaker recommends that its clients evaluate the system by first determining what false-positive and false-negative error rates they can accept, where a false-positive is a record-pair identified as a match by ChoiceMaker which is human tagged as "differ" and a false-negative is a pair human tagged as "match" which ChoiceMaker identifies as "differ" (see Table 4)..

		Clerical decision	
		Differ	Match
ChoiceMaker decision	Differ	Correct	False-negative
	Match	False-positive	Correct

Table 4: Types of errors

Different clients may have different relative tolerances for false-positives and false-negatives. For instance, in a children's immunization database, falsely identifying two individuals as the same person could be very serious because if, say, "John" was vaccinated but "Jon" was not, then John's vaccine would appear on Jon's record and Jon would not be vaccinated. On the other hand, in a counter-terrorism situation, it is more important not to miss any possibly matching suspects, even at the expense of falsely identifying non-terrorists as requiring investigative follow-up.

Given the client's stated tolerance for false-positive and false-negative responses, the client can test the model using our ChoiceMaker Analyzer tool, which determines the "match" and "differ" thresholds (see section 1.1) which will yield no more than the desired error rates. The client can then evaluate the quality of the model based on the percentage of record-pairs that fall between the match and differ thresholds. Since the client will need to evaluate these records by hand, lower percentages are clearly desired.

If the percentage of record-pairs in the "false positive", "false negative", and "needs human review" buckets are acceptable to the client, then the model is ready to be deployed in production. On the other hand, if the accuracy is unsatisfactory, the client can use ChoiceMaker Analyzer to better understand the record-pairs that the system is classifying incorrectly. For example, if there are patterns of errors which suggest the need for additional clues or changes to existing clues, then the client can report these patterns to CMT to help CMT's developers improve the model.

6. SYSTEM INTEGRATION

ChoiceMaker has a flexible architecture which is easily integrated into almost any common system architecture.

ChoiceMaker is a 100% Java application, allowing it to run on any platform supporting Java 1.4, including Windows, Unix, Linux and all other major operating systems.

ChoiceMaker currently runs against Oracle and MS SQL-Server databases, with other database implementations planned when required. As noted below, when operating in batch mode, ChoiceMaker also takes XML or comma separated (.csv) files as input.

Since it is designed from the ground up to be integrated into a larger system. ChoiceMaker offers both Enterprise Java Bean (EJB) and Web Services (SOAP) interfaces. For clients preferring a batch mode, command line interface, this is available too.

As mentioned above in Section 2, "Blocking", ChoiceMaker offers two logical interfaces: real-time and batch. In the real-time version of the algorithm, ChoiceMaker's inputs and outputs are as follows:

> Input: A query record, comprising the identifying information about the person or entity the user is seeking in the database
>
> Output: A list of record ID's in the database which are possible matches to the query record. For each ID, we return ChoiceMaker's decision (either "match" or "hold/possible match") and the probability that the ID matches the query record. In addition, depending on user requirements, we can also return a parsed and standardized version of the input record. For instance, we can parse a person's name into first, middle, and last names and we can parse the address into house number, street name, apartment number, city, state, zip code, and country.

ChoiceMaker's batch interface inputs a large number of query records all at once. These records may be input as one of the following:

1. An Oracle or MS SQL Server database
2. An XML file

© November 2004. ChoiceMaker Technologies, Inc. All Rights Reserved.

3. A delimited flat file, such as a comma separated value (.csv) file which can be exported from Microsoft Excel

For each record in the batch, ChoiceMaker outputs the same information as the real-time version. Note that the batch interface detects matches both among the query records and between the query records and the database records. If there is no database, then ChoiceMaker simply deduplicates the query records.

As noted above, the batch interface provides very high speed throughput, but does not provide a real-time response. It does, however, provide a Web Services or EJB API allowing the client's system to monitor and control the progress of the batch process.

The batch interface also offers a "transitivity engine" which transforms ChoiceMaker's pairwise match or hold decisions into user-defined database actions for each input record. These database actions, for a given record *X*, might include "Load *X* as a new record", "Link *X* with record *Y*", or "Place *X* in the queue of records to be human reviewed".

A simple example of the transitivity engine's usefulness is that it allows the user to specify what they want to do when ChoiceMaker detects that records A and B match and records B and C match, but ChoiceMaker cannot verify that A and C match. Another example is the following: assume that records 1 and 2 are records in a database presumed to be free of duplicates. What should be done when ChoiceMaker matches record *A* to both *1* and *2*? These matches would seem to imply that *1* and *2* are duplicates, which contradicts the assumption that the database is free of duplicates. In each of these cases, the user might specify to ChoiceMaker's transitivity engine that the records in question should be linked, that they should be assumed to be non-matches, or that they should be held for human review. The user also has the option of specifying different actions for the transitivity engine based on the strength of ChoiceMaker's predictions. For instance, if A is a high probability match to both B and C, but B and C are just low probability matches to each other, the user can specify to the transitivity engine that B and C should be declared a match rather than held for human review.

7. RESULTS

The algorithms described in this document have enabled ChoiceMaker to tackle a broad variety of data quality issues. Some representative projects include:

- New York State Education Department: New York State Student Identification System
- Iowa Education Department: A statewide student identification system
- Agency for Toxic Substance and Disease Registry: World Trade Center Registry Project [5]--a database seeking to track everyone in the vicinity of the World Trade Center on 9/11/2001 for epidemiological studies on health issues such as lung cancer, depression, etc.
- A major pharmaceuticals firm: A system to provide real-time matching of a nationwide database of over five million healthcare professionals

Here we will describe a case study of the use of ChoiceMaker on a major public health project at the New York City Department of Health and Mental Hygiene (DOHMH). The DOHMH used ChoiceMaker 2 in a Master Child Index to link the Citywide Immunization Registry (CIR), which tracks children's vaccinations citywide, and LeadQuest, a system which tracks children's blood lead levels for the same citywide population of children. A fuller description of this project can be found in [7].

A children's immunization database proved to be a highly challenging problem because of the inherent noisiness and variability of the data. Children often receive their first vaccination (for Hepatitis B) the day they are born, at which point they often have not been given a first name. Furthermore, children's names change frequently due to nicknaming, Americanization, a mother marrying and changing her name along with the child's, or any one of a number of reasons. Finally, the diversity and unfamiliarity of NYC's multiethnic population, where over 52% of New York City births were to foreign-born mothers, led to an increased number of spelling errors [7].

One factor facilitating the successful completion of this task was that ChoiceMaker was able to leverage a broad range of data fields containing different types of identifying information, including mother's maiden names, Medicaid numbers, and medical record numbers. On the other hand, every one of these fields was present on only a minority of records and was problematic in a different way. For instance, when asked to fill in their maiden name, married women (particularly foreign-born women) will often misunderstand the question and fill in their married name. A Medicaid number should be a completely reliable identifier, but frequently a mother will put down her own Medicaid number for all of her children, creating the possibility that siblings might be incorrectly matched together. Hospital medical record numbers are good identifiers, but are subject to typographical errors. However, we learned that sequential medical record numbers are a strong indicator that the two records represent siblings who are twins, because when twins are born, they are typically entered into the hospital's database sequentially and are assigned consecutive medical record numbers. ChoiceMaker's ClueMaker programming language proved to be adept at describing these complex match or non-match clues while our maximum entropy machine learning technology handled the task of combining these clues into an integrated probabilistic model.

ChoiceMaker built a record matching model containing 193 different clues to handle this complex record matching problem. Out of a total of 4.6 million records in the CIR and LeadQuest, "more than 1.6 million records within and between the two systems were merged, at a very high level of accuracy" [7]. One measure of the success of the MCI project is that the DOHMH is planning to leverage the MCI department-wide. A Communicable Disease Surveillance System will be the first system to join, but down the road, other programs such as Sexually Transmitted Diseases and HIV are planning to join. The DOHMH also reported that "indications are that the matching system is more accurate in merging records than humans" [7].

8. CONCLUSIONS

ChoiceMaker combines several innovative technologies which make for a system which can be rapidly customized to accurately match any kind of data. These technologies include a Java-based

© November 2004. ChoiceMaker Technologies, Inc. All Rights Reserved.

programming language for describing the matching characteristics (called “clues”) of pairs of records in a database and machine learning technology for combining these clues into an overall probabilistic model by learning the relative importance of the different clues from human-marked pairs of records.

ChoiceMaker also exhibits high speed due to its innovative use of patent-pending algorithms for identifying possibly matching records in the first stage of matching known as “blocking”. The Java-based system is available for virtually any operating system and runs against Oracle, MS SQL Server, and XML databases. It provides web services, EJB, and batch interfaces.

ChoiceMaker has proved to be an effective commercial solution for a wide range of approximate matching problems, including a noisy, complicated New York City medical records database.

9. ACKNOWLEDGMENTS

This material is based in part upon work supported by Small Business Innovation Research (SBIR) grants from the National Science Foundation under Awards Number DMI-0060675 and DMI-026213. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

10. REFERENCES

[1] Berger, A., Della Pietra, S. A., and Della Pietra, V. J., A Maximum Entropy Approach to Natural Language Processing *Computational Linguistics*, vol. 22, pp. 39-71, 1996.

[2] Borthwick, A., A Maximum Entropy Approach to Named Entity Recognition 1999. New York University.

[3] Borthwick, A., inventor. ChoiceMaker Technologies, assignee. A Probabilistic Record Linkage Model Derived from Training Data. no. 6,523,019, Feb 18, 2003.

[4] Buechi, M., Borthwick, A., Winkel, A., and Goldberg, A., "ClueMaker: A Language for Approximate Record Matching," Eighth International Conference on Information Quality, Cambridge, Massachusetts, Aug. 2003.

[5] ChoiceMaker Technologies, Inc. ChoiceMaker Technologies provides matching engine in World Trade Center Health Registry. Available at http://www.choicemaker.com/press%20releases/wtc_reg.htm

[6] Knuth, D. *The Art of Computer Programming*, Reading, MA: Addison-Wesley, 1998.

[7] Papadouka, V., Schaeffer, P., Metroka, A., Borthwick, A., Tehranifar, P., Leighton, J., Aponte, A., Liao, R., Ternier, A., Friedman, S., and Arzt, N., Integrating the New York Citywide Immunization Registry and the Childhood Blood Lead Registry *Journal of Public Health Management and Practice*, vol. Nov, 2004.

© November 2004. ChoiceMaker Technologies, Inc. All Rights Reserved.