Evaluating Genetic Algorithms for selection of similarity functions for record linkage

Faraz Shaikh and Chaiyong Ragkhitwetsagul
Carnegie Mellon University
School of Computer Science
ISRI - Institute for Software Research International
{fshaikh,cragkhit}@andrew.cmu.edu

Abstract

Machine learning algorithms have been successfully employed in solving the record linkage problem. Machine learning casts the record linkage problem as a classification problem by training a classifier that classifies 2 records as duplicates or unique. Irrespective of the machine learning algorithm used, the initial step in training a classifier involves selecting a set of similarity functions to be applied to each attribute to get a similarity measure. Usually this is done manually with input from a domain expert. We evaluate an approach in which the optimal combination of similarity function for a given type of input data records is searched using Genetic Algorithms.

1 Introduction

Record Linkage is an important pre-processing step on raw data integrated from multiple sources. The problem is aggravated when records from the two sources do not share a unique key based. Absence of a shared unique key mandates use of complex methods for purging duplicate records from the final integrated data set [3].

A trivial method for doing record linkage is to form a set of rules for detecting duplicates. Such rules are then tuned over time to accommodate exceptions. The solution may or not be accurate but is definitely not elegant. Firstly this method needs creating the initial rule set by a domain expert; second this method requires constant monitoring and human intervention of some type for detecting and accommodating exceptions [8, 9].

Machine learning algorithms alleviate the two problems of creation of the initial rule set and constant monitoring and adapting the rule set to exceptions. A machine learning approach to create the initial rule set is described in [14]. This method is

based on computing similarity scores between records and then learning the mapping rule. The method requires sufficiently large amount of labeled data. This labeled data is created manually and sometimes is infeasible to generate because the labeling has to perform and exhaustive search over the dot product of the input records. As the number of records N increase the search space for labeling increasing as a quadratic equation [N*N]. In [16] the author tries to employ active learning for selecting the most uncertain record pair for a teacher to label. This significantly reduces the number of labeled record pairs required for the learner to converge to optimal mapping rules. The authors in [16] also suggest methods like grouping, indexing and sampling for managing the quadratic size [N*N] of the search space generated in the labeling stage.

One important class of machine learning algorithms employed in record linking is Probabilistic Record Linkage. PRL uses a large labeled datasets to generate a weight vector for similarity each attribute. The weights are generated according to statistical measures over the training data. Each weight can be seen as a measure of how strongly/weakly a match for a particular attribute affects the probability of the record pair being a duplicate. To classify new records each agreement /disagreement on individual attributes of a records are considered in proportion to the weight vector and sum is generated. This sum is then used in determining the class of the record pair and duplicate or unique. Usually 2 thresholds values are chosen for the sum called the 'positive and negative threshold'. All records pairs having a sum of weighted agreement greater than the positive threshold are considered to be sure duplicates. Similarly all pairs below having the weighted agreement less than the negative threshold are considered to be surely unique. Record pairs whose weighted agreement values fall between the positive and the negative threshold fall in the confusion area of the classifier and the classifier may decide to ask a teacher for help in classifying such records. In summary PRL can be viewed as the Naïve Bayes algorithm for record classification.

In the above discussion we have *assumed* that we **have** a set of similarity functions to be applied for matching individual attributes of the record. Usually this is true if the attributes are small in number and the attribute domain is well understood. For example one can fairly conclude that *soundex()* is a better similarity match function for a field like names of a person. But, the *soundex()* may or may not perform better on an attribute where similar sounding attributes are actually unique. Similarly a numeric attribute cannot be subjected similarity functions that are designed to work on strings. Also depending upon the quality of data soundex() and stringcmp() may perform differently as similarity functions. Soundex and stringcmp() are just 2 representative similarity functions discussed here for the sake of an example. A complete list of similarity matching functions is discussed by W. Cohen in [2, 4, 5, 6, 7].

We wish to evaluate the effectiveness of Genetic Algorithms to search the optimal set of similarity functions to be used for training a classifier given a type of input records [1, 11, 12, 16].

2 Related Work

The task of finding the optimum set of similarity functions is in our view considered to be a trivial task. Not much has been done in automatically tuning the selections functions. This to our understanding is because the record linking problem domain already has way too many problems to solve. Finding the optimum set of similarity functions is can also be treated a pre-processing step before performing the actual record linking. We feel this is because the record linking hasn't been tried out on records with large number of fields. Also since most of the input records on which the linking is to be performed have a well-understood attribute domain, no one really looks into the performance of similarity functions. In a way there is an implicit hidden assumption based on empirical evidence that certain similarity functions are the best functions for a given attribute domain. Nevertheless, we feel that using GA to find the optimum set of similarity functions can give output similarity function sets that are not quite obvious just by looking at the domain of the attribute [10, 15].

The closest research work done on similar line is done by Marco Modesto et.al in [14]. Our work is based on evaluating the effectiveness of Marco's methods on real world data. The *original idea is completely attributed* to Marco Modesto. Nevertheless Marco's presentation of the idea in [17] is just an introduction to the idea. We apply the author's idea on records collected from 2 real world data sources. The data source is made available to us from [17]. In the original experiment the author used some of this own personal data as input and used the Febryl framework to learn the optimal similarity function set. The similarity functions that took part in each generation of the GA iteration were standard inbuilt functions provided by Febryl in [5].

A comprehensive listing of similarity functions available for names and records is presented by W.Cohen in [4]. Finally, we have chosen the Weka library to provide us with standard similarity function like Jaccard, Jaro, Jaro-Winkler, Levenstein, Monge-Elkan and Smith-Waterman. The selection of these similarity functions is based on the criterion of best capturing the notion of similarity of attributes in the restaurant test data set. We believe the above selected similarity functions in the entirety are enough to capture the notion of similarity in the given attribute set of {Name, Street address, City, Type of food}.

Also some of work in coming up with an optimum set similarity functions as well as learning the parameters (viz. predicate threshold) associated with these similarity functions comes under the banner of "learnable similarity functions". Learnable similarity functions are researched extensively by Bilenko in [12, 13]. Bilenko concludes that similarity functions are learnable and provide performance improvement is applications requiring them like record linkage, semi-supervised clustering and blocking. Bilenko's doctoral thesis [13] is a very comprehensive study of use different learning approaches to learn similarity functions. The paper also suggests that similarity functions can be learnt at both the field level and the record level. A mix and match of similarity functions at both record and attribute levels along with theirs parameters defines the search space of our learning problem. Belenko's research experimented with most of the prominent learning algorithms (viz SVM, HMM) for learning similarity functions. Albeit, genetic algorithms are not covered in Belenko's extensive work, we feel that Genetic algorithms in essence are covered by others algorithms researched by Belenko.

3 GA for selecting optimal similarity function set

In this section we describe the approach to find the set of optimal similarity functions to be used to a given set of input attributes.

Inputs:

a) A set of labeled records pairs of the form

$$Dtrain = \{R1, R2 \text{-> } Distinct\}$$

Where are R1 belongs to data-set 1 and R2 belongs to data-set.

b) A set of available candidate similarity functions of the form.

$$SimFunc_{candidate} = \{Sim\ (R1, R2) \text{ -> } (0 \text{ to } 1)\}$$

Where each similarity outputs the real similarity value of two text inputs R1, R2 on a scale of 0 to 1. 1 Signifying a perfect match

Output:

a) A set of candidate similarity functions to be applied to the attributes of the records to be linked.

b) A decision tree (or other classifiers) based that matched the based on the selected candidate similarity functions. This tree (classifier) is the best tree that survives after the selected number of generations.

Method:

1. Partition the available labeled data into 2 set D_{train} and D_{test}

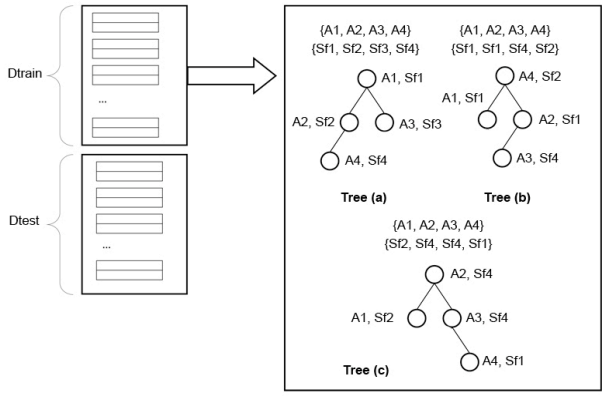

Figure 1 Using Dtrain to create decision trees according to the sets of similarity

2. Create and initial population composed of a set of randomly selected similarity function from $SimFunc_{candidate}$ (see Figure 1).

3. Using the training data D_{train} generate a decision tree for each element in the population created in step 2. The nodes of the tree are attributes and each node we selected use the similarity function assigned to the particular attribute

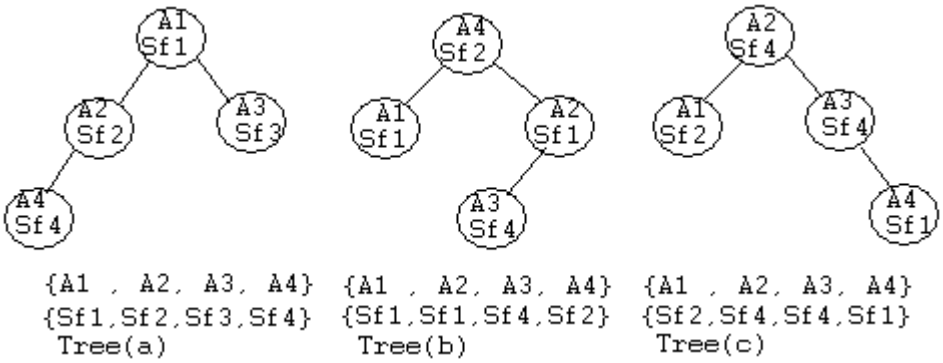

Figure 2 Represent decision trees formed by sets of similarity functions

The above figure depicts 3 such trees that are created as part of step3. Each tree is generated using a set of similarity functions. Tree (b) for example was generated using the similarity function set {Sf1, Sf1, Sf4, Sf2}. In this example we have 4 attributes {A1, A2, A3, A4}. Tree (b) thus uses Sf2 similarity function at the root node with attribute A4.

4. For each tree calculate created in step 2 calculate its fitness using the fitness function (described ahead). The fitness function is a measure of how well the tree performs on D_{test}. This is the selection step in a Genetic Algorithm (GA).

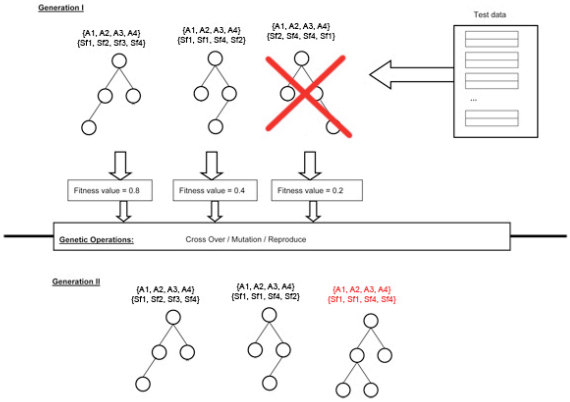

Figure 3 Using Genetic Algorithm to remove and reproduce the sets of similarity functions

5. From the trees M available select the fittest N fit trees for reproduction and discard the (M-N) trees.

6. Using the M most fit trees selected in step 5 create a new set of population using function crossover, mutation.

7. Using this new population create trees for the next generation.

8. Repeat above steps for a fixed number of generations.

9. Output the selection function vector of the fit tree and the tree itself.

As you can see, the above-mentioned algorithm tries to find the set of optimal selection functions that maximize the fitness function. The fitness function in turn is defined as how well a tree performs on the test data. The steps of cross over and mutations are stands step in GA to avoid common pitfall like converging to a local maxima for the fitness function [10, 11].

3.1 Fitness function

The fitness function to be used should be such that favors tress, which classifies more, test examples correctly and penalizes the trees in proportions to the miscalculations they output. We used the same fitness function as mentioned in [17]. The formula to find fitness score for each individual (set of similarity functions) is

$$Fitness() = (RI * 10) - (WI * 2)$$

Where RI means the number of right classification, and WI means the number of incorrect classification.

In each evolution, there are some individuals in the population, which are not strong enough (low fitness score). These individuals cannot survive to the next generation. Thus, we remove them from the population.

3.2 Reproduction

The fittest individuals have chances to reproduce new members of the population. The reproduction process can be done by using “cross over” between two individuals or by “mutation” of an existing individual. The number of new members might be varying.

3.3 Selection

The selection process mainly depends on the fitness score. The most n fittest individuals are selected to be in the next generation. On the opposite way, we can also remove the n weakest individual out from the population.

3.4 Cross Over

The cross over process happens in the reproduction process. To produce a new member to the population, the new member usually comes from selecting features from a pair of existing members. This can happen randomly or by some selecting criteria.

3.5 Mutation

Mutation causes some changes in the genetic of an individual. By the way, the changes are very small, and don’t make big difference. It can happen by a biologically transformation of genes. It our project, it means changing some features in a set of similarity function.

4 Evaluation Framework

We evaluate our claim by comparing the result from Genetic Algorithm to the result from baseline experiment. The baseline experiment composes of training a classifier using 6 similarity functions (Levenstein, Jaccard, Jaro, Jaro-Winkler, Monge-Elkan, and Smith-Waterman) one at a time [4, 5, 6, 7]. This classifier is used to classify the test data. The baseline result comes from the best result of these 6 similarity functions.

4.1 Baseline result

Table 1 Baseline results

Similarity functions	Number of correctly classified records	Number of misclassified records	F-measure
Jaccard	241	10	0.86111111112
Jaro	250	1	0.98360655737
Jaro-Winkler	240	11	0.81967213114
Levenstein	243	8	0.88571428571
Monge-Elkan	184	67	0.42735042735
Smith-Waterman	240	11	0.81967213114

As we can see that some similarity functions perform better than the others. This is because different similarity functions can work well with different kind of string. Our project aims to solve this problem by determining the most suitable similarity function for each attribute in a provided dataset.

4.2 Experimental results

Our experiment is setup using the following configurations:

1. Number of evolution: 20
2. Using 6 similarity functions (as in the baseline): Levenstein, Jaccard, Jaro, Jaro-Winkler, Monge-Elkan, and Smith-Waterman
3. Population size: 1296
4. Number of evolutions: 20
5. Number of selection: 1276
6. Number of crossover: 20/evolution

Average f-measure score as in figure 4 measures the result for each evolution. We can see that the average f-measure score increases every evolution.

The result from the best individual in the last evolution is as follows:

Correctly Classified Instances	250	99.6015%
Incorrectly Classified Instances	1	0.3985%
f-measure	0.9841269841269841	

Table 2 Confusion Matrix

	Non-duplicate	Duplicate
Non-duplicate	219	1
Duplicate	0	31

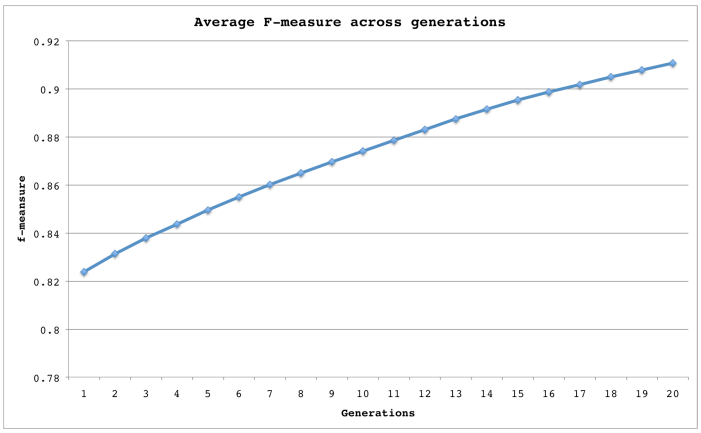

Figure 4 Average f-measure for each evolution

As a result, we obtained the six best sets of similarity functions, which can produce the same result, and have the same fitness score.

Table 3 Best sets of similarity functions

	Name	Street Address	City	Kind of Food
Set 1	Levenstein	Jaro	Jaro	Levenstein
Set 2	Levenstein	Jaro	Jaro	Jaro
Set 3	Levenstein	Jaro	Jaro	Monge-Elkan
Set 4	Levenstein	Jaro	Jaro	Jaccard
Set 5	Levenstein	Jaro	Jaro	Smith-Waterman
Set 6	Levenstein	Jaro	Jaro	Jaro-Winkler

5 Conclusion

The results clearly show that we can learn similarity functions using genetic algorithms. Across our experiments the F-measure across generations always shows improvements as we training each advancing generation. We have seen that given a small selection of similarity functions viz 2 to 3 the genetic algorithm quickly (9^{th} To 10^{th} generation) converges to optimum selection of similarity functions. Although we would like to mentions the convergence is highly sensitive to number of choices available as similarity functions. In our final experiment we used 6 similarity functions and the algorithm converged in the late 100'Th of generation. From our results its fair to assume that without any optimizations, finding the

optimum set of similarity function using genetic algorithm can quickly becomes and computable intractable as the choice for similarity functions increases.

6 Acknowledgments

Thanks to kind Professors, Tom Mitchell, and William Cohen. Also thanks for Andrew Arnold and Mary McGlohon for contributing as our TAs in this course.

References

[1] Banzhaf, W., Nordin, P., E, R. E. K. R., and Francone, F. D. Genetic Programming - An Introduction: On the Automatic Evolution of Computer Programs and Its Applications. Morgan Kaufmann Publishers, 1998.

[2] Bilenko M., Mooney R. J., Cohen W. W. , Ravikumar P., and Fienberg S. Adaptive name matching in information integration. In IEEE Intelligent Systems 18. 5 (September/October 2003), 16-23.

[3] Carvalho M. G., Silva A. S. Learning to Deduplicate. In JCDL'06. (2006) 41-50.

[4] Cohen, W. W. Data integration using similarity joins and a word-based information representation language. ACM TOIS 18, 3 (2000), 288–321.

[5] Cohen, W. W.; Ravikumar, P.; and Fienberg, S. E. A comparison of string distance metrics for name-matching tasks. In Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03). 2003.

[6] Cohen W.W., Ravikumar P., Fienberg S. E. A Comparison of String Metrics for Matching Names and Records. In American Association for Artificial Intelligence. 2003.

[7] Cohen, W. W., and Ravikumar, P. 2003. Secondstring: An open-source java toolkit of approximate string-matching techniques.

Project web page, http://secondstring.sourceforge.net.ternal Revenue Service Publication R99/04. Available from http://www.census.gov/srd/www/byname.html; [accessed 10 April 2008.]

[8] Fellegi I. P., Sunter A. B. A theory for record linkage. Journal of American Statistical Association 66, 1 (1969), 1183-1210.

[9] Joachims, T. Learning to Classify Text Using Support Vector Machines. Kluwer. 2002.

[10] Jones G. Genetic and Evolutionary Algorithms. Available from http://www.wiley.co.uk/ecc/samples/sample10.pdf; accessed 12 April 2008.

[11] Koza J. R. Genetic Programming: on the programming of computers by means of natural selection. MIT Press, 1992.

[12] Mikhail Bilenko .Learnable Similarity Functions and Their Applications to Record Linkage, Proceedings of the Ninth AAAI/SIGART Doctoral Consortium

[13] Mikhail Y. Bilenko Phd Thesis Learnable Similarity Functions and Their Application to Record Linkage and Clustering. Available from: http://www.cs.utexas.edu/~ml/papers/marlin-dissertation-06.pdf [accessed 4 April 2008]

[14] Modesto M., Carvalho M. G. Record Deduplication By Evolutionary Means.

Available from

http://homepages.dcc.ufmg.br/~nivio/cursos/pa06/seminarios/seminario16/seminario16.p

df [accessed 4 April 2008]

[15] Sarawagi S., Bhamidipaty A. Interactive deduplication using active learning. In

Conference on Knowledge Discovery in Data (Edmonton, Alberta, Canada). ACM, New

York, 2002. pp. 269-278.

[16] Tejada S., Knoblock1 C. A., and Minton S. Learning Object Identification Rules for Information Integration. In Information Systems Vol. 26, No. 8, pp. 607–633, 200.

[17] Whitley D. A. Genetic Algorithm Tutorial. Available from: http://www.cs.iastate.edu/~honavar/ga_tutorial.pdf; accessed 10 April 2008.