Frequency-aware Similarity Measures

Why Arnold Schwarzenegger is Always a Duplicate

Dustin Lange Felix Naumann
Hasso Plattner Institute, Potsdam, Germany
firstname.lastname@hpi.uni-potsdam.de

ABSTRACT

Measuring the similarity of two records is a challenging problem, but necessary for fundamental tasks, such as duplicate detection and similarity search. By exploiting frequencies of attribute values, many similarity measures can be improved: In a person table with U.S. citizens, Arnold Schwarzenegger is a very rare name. If we find several Arnold Schwarzeneggers in it, it is very likely that these are duplicates. We are then less strict when comparing other attribute values, such as birth date or address.

We put this intuition to use by partitioning compared record pairs according to frequencies of attribute values. For example, we could create three partitions from our data: Partition 1 contains all pairs with rare names, Partition 2 all pairs with medium frequent names, and Partition 3 all pairs with frequent names. For each partition, we learn a different similarity measure: we apply machine learning techniques to combine a set of base similarity measures into an overall measure. To determine a good partitioning, we compare different partitioning strategies. We achieved best results with a novel algorithm inspired by genetic programming.

We evaluate our approach on real-world data sets from a large credit rating agency and from a bibliography database. We show that our learning approach works well for logistic regression, SVM, and decision trees with significant improvements over (i) learning models that ignore frequencies and (ii) frequency-enriched models without partitioning.

Categories and Subject Descriptors

I.5 [**Pattern Recognition**]: Clustering—*Similarity Measures*

General Terms

Algorithms

Keywords

similarity measures, duplicate detection, similarity search

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

1. COMPARING RECORDS

The problem of determining the similarity (or distance) of two records in a database is a well-known, but challenging problem. Suitable similarity measures help to find duplicates and thus cleanse a data set, or they can help finding nearest neighbors to answer search queries. The problem comprises two main difficulties: First, the representations of same real-world objects might differ due to typos, outdated values, and sloppy data or query entries. Second, the amount of data might be very large, thus prohibiting exhaustive comparisons. The first problem is overcome by devising sophisticated similarity measures (the focus of this paper); the second problem is alleviated by efficient algorithms and indexes that avoid comparing each entry or query with all other entries.

Our original motivation stems from a cooperation with Schufa, a credit rating agency that stores data of about 66 million citizens, which are in turn reported by banks, insurance agencies, etc. To ensure the quality of the data, it is necessary to detect and fuse duplicates therein [14]. Also, queries about the rating of an individual must be responded to as precisely as possible.

We propose a novel comparison method that partitions the data using value frequency information and then automatically determines similarity measures for each individual partition. We show that this method indeed finds a different configuration for each partition and that we achieve an overall better precision. Note that while we motivate our ideas using a person data use case, our approach is general and can be used with any composed similarity measure. We report on experiments with the person data set, but also with a bibliographic data set from DBLP.

Example: Two Arnold Schwarzeneggers. Assume a table in which each row represents a person. We know a person's name, birth date, and address. We need to decide whether two different records represent the same person. Usually, we use some similarity measure to answer this question. A common approach is to use the same, carefully crafted similarity measure across all records. We argue that it makes sense to use different similarity measures for different groups of records.

We explain the intuition of our idea by means of Arnold Schwarzenegger. In a person table containing US citizens, we cannot find many Arnold Schwarzeneggers. In fact, it is highly unlikely that there are two Arnold Schwarzeneggers stored in our table. Arnold Schwarzenegger is our representative for a rare name. From this rarity, we conclude that

another record with this same name is already quite likely to be a duplicate of the earlier. We are less rigorous about the similarity of birth date and address: For two rows with rare names, we argue that address and date-of-birth similarity are less important than for rows with frequent names. □

With this paper, we make the following contributions:
Exploiting frequencies: We propose two approaches to exploit frequencies in learnable similarity measures: The first directly incorporates frequencies in machine learning models. The second partitions the data according to value frequencies and learns a composite similarity measure for each partition.
Partitioning strategies: We propose different partitioning strategies: a greedy partitioning algorithm, equi-depth partitioning, random partitioning, as well as a novel partitioning algorithm inspired by genetic programming.
Evaluation on two real-world data sets: The first is a person data set from Schufa, a large credit rating agency. The data set contains ca. 66m person records and 2m search queries, from which we evaluate 10,000 of the most difficult ones. The second data set is from the DBLP bibliography, from which we created a data set that contains 10,000 paper reference pairs (available from our website, see Sec. 5).

2. RELATED WORK

In the last decades, many researchers have worked on duplicate detection. For a general overview on duplicate detection, we refer the reader to excellent surveys [4, 15].

Bilenko et al. use machine learning techniques to learn similarity measures [3]. They learn base similarity measures using SVMs and apply another SVM to combine the base similarity measures. Sarawagi and Bhamidipaty use active learning for interactive duplication detection [11]. They also employ machine learning techniques (decision trees, Naive Bayes, and SVMs) to combine base learners. We adapt and greatly extend this approach by creating a set of learners for different partitions of the data. Shen et al. [12] propose to create a set of different matchers for different portions of the data. They focus on the task of integrating a set of data sources and create different similarity measures for comparing entities from the different sources. In contrast to their work, we partition data according to frequencies and not based on different sources of the data. Moreover, we employ a set of similar matchers, i.e., we learn one similarity function for each of the partitions – but all of them with the same machine learning technique. Another idea is to use actual attribute values for partitioning (e.g., using a Country attribute, we learn one measure for US citizens and one measure for Austrian citizens). While this idea depends on the availability of attributes values that are suitable for partitioning, our approach is more generally applicable as it only exploits meta-information of the values (frequencies).

There are also approaches to duplicate detection/entity resolution that exploit knowledge about frequencies. Bhattacharya and Getoor apply collective entity resolution; by incorporating knowledge about references between entities, entity resolution can be improved [2]. Their approach starts by resolving entities with least frequent names, since "two references with the name 'A. Ansari' are more likely to be the same, because 'Ansari' is an uncommon name." Torvik and Smalheiser propose a probabilistic approach to author name disambiguation [13]. In their model, they include the frequency of names to predict the probability that two names match. They also recognize that "if the name is very unusual (e.g., D. Gajdusek), the chances are better that any two randomly chosen articles with that name are written by the same individual than if the name is very common (e.g., J. Smith)." We also exploit this insight, but with an entirely different approach.

3. COMPOSING SIMILARITY

In this section we describe the similarity model used throughout the paper. We follow the common and proven notion of defining individual similarity measures for different attributes and attribute types; for instance, dates are compared differently than names or addresses. These individual similarities are subsequently combined to define some global similarity of two records.

Base Similarity Measures. We first split the problem of measuring the similarity of two records into smaller subproblems. We define a set of **base similarity measures** $sim_p(r_1, r_2)$, each responsible for calculating the similarity of a specific attribute p of the compared records r_1 and r_2 from a set R of records. In our use case, we have three functions sim_{Name}, $sim_{\mathsf{BirthDate}}$, and sim_{Address}. All base similarity measures can be chosen independently. For example, we could use Jaro-Winkler distance for sim_{Name} [15], the relative distance between dates for $sim_{\mathsf{BirthDate}}$, and Euclidean distance for sim_{Address}. The base similarity measures can also test for equality (e.g., for email addresses) or boolean values (e.g., for gender).

Each base similarity measure is a function

$$sim_p : (R \times R) \rightarrow [0, 1] \subset \mathbb{R} \quad (1)$$

In the following, we assume the domain of the similarity measures to be between 0 and 1, with 1 representing identity and 0 dissimilarity of the compared record parts.

Composition of Base Similarity Measures. To calculate the overall similarity of two records, we integrate the base similarity measures into an overall judgement. We consider the task of judging whether two records are similar a classification task: the classes are *isSimilar* and *isDissimilar*; the features are the results of the base similarity measures. To derive a general model, we employ machine learning techniques. We have enough training data for supervised learning methods. In case of lacking training data, active learning methods could be chosen [11].

Due to their popularity and strong performance in different classification tasks, we selected the following three classification methods for further analysis: logistic regression, decision trees, and support vector machines.

4. EXPLOITING FREQUENCIES

In our use case, we have data about a large set of persons. We need to define a similarity measure for comparing persons, but our findings are equally applicable to other use cases, such as products, bills, etc. We want to answer the question: Can information about data distribution improve our similarity measurement?

In this section, we propose two approaches to exploit frequencies in learning similarity measures. A requirement for both approaches is a frequency function that is introduced in

Sec. 4.1. In Sec. 4.2, we describe how to adapt the machine learning techniques to exploit frequencies. As an alternative approach, a partitioning idea is discussed in detail in Sec. 4.3.

4.1 Frequency Function

First of all, we need to select attributes for frequency evaluation (in our use case, we select the attributes FirstName and LastName). For two compared records, we then determine the value frequencies of the selected attributes. We define a frequency function f that takes as arguments two records and determines a frequency:

$$f : R \times R \to \mathbb{N} \tag{2}$$

Modeling the frequency function is a domain-specific task. In the following, we describe how we modeled this function in the Schufa use case. Our goal is to partition the data according to the name frequencies. We have two name attributes in our data model (FirstName and LastName) and need to handle several data quality problems: swapping of first and last name, typos, and combining two attributes (so that one frequency value is calculated). First and last name may be switched (e. g., FirstName=`Arnold`, LastName=`Schwarzenegger`; FirstName=`Schwarzenegger`, LastName=`Arnold`). We take this possible switch into account by calculating the attribute similarities for both attribute value combinations (switched and non-switched) and proceeding with the combination that results in a larger similarity value. Next, typos may occur in attribute values (e. g., `Arnold; Arnnold`). We assume that at least one of the spellings is correct and that a typo leads to a less frequent name. Although this is not always true, this heuristic works in most cases. Thus, we take the larger frequency value of both spellings. Lastly, we combine the frequency values of first and last name. We argue that the less frequent name is more distinguishing and helpful (e. g., `Schwarzenegger` is more distinguishing than `Arnold`). Thus, we take the smaller frequency of the different attribute values as the result of our frequency function. Experiments with alternatives, namely using the maximum or the average instead of the minimum, showed that minimum is in fact the best accumulation function in our use case.

This description reflects the characteristics of our person data use case; for other data sets, the individual frequency function must be adjusted accordingly. For the DBLP data set, our frequency function works similar to the Schufa function explained above, except that there is no check for switched first and last names, since the name parts are not split in this data set.

4.2 Frequency-enriched Models

A first idea to exploit frequency distributions is to alter the models that we learned with the machine learning techniques explained in Sec. 3. One could manually add rules to the models, e.g., for logistic regression, we could say "if the frequency of the name value is below 10, then increase the weight of the name similarity by 10% and appropriately decrease the weights of the other similarity functions". Manually defining such rules is cumbersome and error-prone.

Another idea is to integrate the frequencies directly into the machine learning models. We add the frequencies of the compared entities' attribute values as an additional attribute to the discussed models. Some machine learning techniques can only handle normalized feature values (e. g., logistic regression and SVMs). Since all similarity values are required to lie in the range $[0, 1]$, we need to scale the frequency values accordingly. We apply the following scaling function:

$$scaled\_f(r_1, r_2) = \frac{f(r_1, r_2)}{M}, \tag{3}$$

where M is the maximum frequency in the data set.

Adding attributes to the learned model means adding complexity and mixing information. The models become "polluted" with frequency information. The comprehensibility of the created models is lower, since each model contains mixed decisions based on similarity values and frequencies. As our experiments in Sec. 5 show, this idea is outperformed by the partitioning approach that we describe in the following.

4.3 Partitioning

We propose to partition compared record pairs based on frequencies and create different models for the different partitions. These models are equal to the ones learned in Sec. 3, we just create several of them. The models still decide only on the basis of similarities. Since the models do not contain any additional information about the frequencies, they are still as easy to interpret and adjust as before.

We now partition compared record pairs into n partitions using the determined frequencies. The number of partitions is an important factor for partitioning. A too large number of partitions results in small partitions that can cause overfitting. A too small number of partitions leads to partitions too large for discovering frequency-specific differences. To determine a good number of partitions as well as a good partitioning, we need a partitioning strategy. We introduce several strategies in Sec. 4.4.

In the following, we formally define partitions. The entire frequency space is divided into non-overlapping, continuous partitions by a set of thresholds:

$$\Theta = \{\theta_i \mid i \in \{0, \ldots, n\} \wedge \theta_0 < \ldots < \theta_n\} \tag{4}$$

with $\theta_0 = 0$ and $\theta_n = M + 1$, where M is the maximum frequency in the data set. The partitions are then defined as frequency ranges I_i:

$$I_i = [\theta_i, \theta_{i+1}) \tag{5}$$

A partitioning I is then a set of partitions that covers the entire frequency space:

$$I = \{I_j \mid j \in \{0, \ldots, n-1\}\} \tag{6}$$

A partition covers a set of record pairs. A record pair (r_1, r_2) falls into a partition $[\theta_i, \theta_{i+1})$ iff the frequency function value for this pair lies in the partition's range:

$$\theta_i \leq f(r_1, r_2) < \theta_{i+1} \tag{7}$$

For each partition, we learn a composite similarity measure using the learning techniques presented in Sec. 3. It is now the task of the learning method to derive an appropriate model for each partition. In other words, we shift the problem of appropriately handling rare or frequent values to the learning method. If the learning method cannot exploit frequencies, then we expect the method to generate the same model for each partition (assuming each partition is large enough for creating appropriate models).

With the set of composite similarity measures for all partitions at hand, we can judge new record pairs. For a record pair, we first determine the partition that the pair belongs to with Formula (7). We then apply only the composite similarity measure that has been learned for this partition. By retaining frequency lists for all frequency-relevant attributes, the frequency lookup is quite fast. Thus, our approach hardly affects the query time. This is quite important for our use case as each query needs to be answered in less than a second.

4.4 Partitioning Strategies

An important problem in partitioning our data is to determine the number of partitions as well as the thresholds that separate the partitions. Since the number of partitions is determined by the number of thresholds, we only need to determine a set of thresholds.

A complete search on the threshold space is too expensive. For a maximum frequency M, we could define up to $M-1$ thresholds, resulting in a separate partition for each possible frequency. Overall, there are 2^{M-1} possibilities to choose threshold sets, since each distinct frequency $f \in \{0,\ldots,M\}$ can be either contained or not contained in a threshold set. For a reasonable maximum frequency of $M = 1,000,000$, there are obviously too many threshold combinations to consider. Even if we consider only the number of distinct frequencies that actually occur as possible thresholds in the data set, the computation costs are too high. In our use case data set, the frequency of the most frequent last name is 616,381. The number of distinct frequencies is 4629, which results in 2^{4628} theoretically possible different partitionings.

To efficiently determine a good partitioning, we suggest the following partitioning strategies, which we empirically compare in Sec. 5.1 and Sec. 5.2:

Random partitioning: To create a random partitioning, we randomly pick several thresholds $\theta_i \in \{0,\ldots,M+1\}$ from the set of actually occurring frequencies of the considered attribute values. The number of thresholds in each partitioning is also randomly chosen. The maximum number of partitions in one partitioning as well as the total number of generated initial partitionings are fixed. For our use case, we define a maximum of 20 partitions in one partitioning.

Equi-depth partitioning: We divide the frequency space into e partitions. Each partition contains the same number of tuples from the original data set R. For our use case, we create partitionings for $e \in \{2,\ldots,20\}$.

Greedy partitioning: We define a list of threshold candidates $C = \{\theta_0,\ldots,\theta_n\}$ by dividing the frequency space into segments with the same number of tuples (similar to equi-depth partitioning, but with fixed, large e, in our case $e = 50$). We then begin learning a partition for the first candidate thresholds $[\theta_0,\theta_1)$. Then we learn a second partition that extends the current partition by moving its upper threshold to the next threshold candidate: $[\theta_0,\theta_2)$. We compare both partitions using F-measure. If the extended partition achieves better performance, the process is repeated for the next threshold slot. If not, the smaller partition is kept and a new partitioning is started at its upper threshold; another iteration starts with this new partition. This process is repeated until all threshold candidates have been processed.

This greedy partitioning algorithm stops after encountering a worse F-measure. A worse F-measure is an indicator for a more difficult set of tuples in the analyzed partition; thus, it makes sense to create a new partition for this set.

Genetic partitioning: Another partitioning approach is inspired by genetic algorithms. Since we have achieved overall best results with this approach in our experiments, we dedicate the following section to genetic partitioning.

4.5 Genetic Partitioning

To determine a good partitioning, but keep the number of generated and evaluated partitions low, we apply ideas inspired by genetic algorithms [1, 8]. Genetic algorithms in turn are inspired by biological evolution. From an initial population, the fittest individuals are selected. These individuals "breed" offspring that are assumed to be fitter than their parents. Random genetic mutation and crossover produce even fitter offspring.

Indeed, genetic algorithms are a good fit to our problem. A specific partitioning contains several thresholds, some of which are a good choice, while others should be moved by a small portion and again others should be replaced or removed completely. Deciding whether a threshold is a good choice is difficult, as we can only evaluate performance of partitions, and these are defined by *two* thresholds. In addition, we do not know the optimal number of partitions in advance. Due to their intelligent selection algorithms and random events, genetic algorithms can efficiently handle these choices and are a promising choice for partitioning (as we also empirically show in Sec. 5.1 and 5.2).

Genetic Partitioning Algorithm. The detailed steps of the genetic partitioning algorithm are the following:

Initialization: We first create an initial population consisting of several random partitionings. These partitionings are created as described above with the random partitioning approach.

Growth: Each individual is grown. We learn one composite similarity function for each partition in the current set of partitionings using the techniques presented in Sec. 3.

Selection: We then select some individuals from our population for creating new individuals. A fitness function determines which partitionings to select. In our case, we choose F-measure as fitness function using our training data. For each partition, we determine the maximum F-measure that can be achieved by choosing an appropriate threshold for the similarity function. We weight the partitions' F-measure values according to the compared record pairs in each to calculate an overall F-measure value for the entire partitioning. We then select the partitionings with highest weighted F-measure. For our use case, we select the top five partitionings. Note that we use a test set for evaluation that is different from the training set used for learning the composite similarity measure for each partition.

Reproduction: We build pairs of the selected best individuals (during all iterations) and combine them to create new individuals. Two techniques are applied to each new partitioning. *Recombination:* We first create the union of the thresholds of both partitionings. For each threshold, we randomly decide whether to keep it in the result partition or not. Both decisions have equal chances. *Mutation:* We randomly decide whether to add another new (also randomly picked) threshold *and* whether to delete a (randomly picked) threshold from the current threshold list. All possi-

bilities have equal chances. If a partition is too small, then we have too little training and test data available. To avoid too small partitions (and thus overfitting), we define a minimum partition size (we set this value to 20 record pairs in our use case). Randomly created partitionings with too small partitions are discarded. In our use case, we create two new partitionings for each partitioning pair processed in the reproduction phase.

Termination: The resulting partitions are evaluated and added to the set of evaluated partitions. The selection/reproduction phases are repeated until a certain number of iterations is reached or until no significant improvement can be measured. In our use case, we require a minimum F-measure improvement of 0.001 after 5 iterations. The algorithm returns the partitioning that has been evaluated best during all iterations.

5. EVALUATION

In this section, we provide a detailed evaluation of our approach. We describe the Schufa data set in Sec. 5.1. We compare different partitioning strategies and show improvements achieved by our approach. In Sec. 5.2, we show results of experiments on a data set extracted from DBLP.

We performed all tests on a workstation PC. Our test machine runs Windows XP with an Intel Core2 Quad 2.5 GHz CPU and 8 GB RAM. We used Weka [6] as implementation of the machine learning techniques.

5.1 Evaluation on Schufa Data Set

We evaluated our approach on real-world data from Schufa, a large credit agency. The Schufa database contains information about the credit history of about 66m people.

Preparation. Our data set consists of two parts: a person data set and a query data set. The person data set contains about 66 million records. The most relevant fields for our search problem are name, date and place of birth, and address data (street, city, zip). The query data set consists of 2 million queries to this database. For each query, we know the exact search parameters (most record fields are mandatory), and the result obtained by Schufa's current system. This result contains up to five candidate records.

The Schufa system automatically evaluates its performance. A confidence value is assigned to each result. If only one result could be found and if its confidence is above a pre-determined high threshold, then the result is automatically accepted. Results with a confidence below a predetermined low threshold are automatically discarded. For all other results, Schufa is particularly careful: An expert needs to determine whether one of the results can be accepted or not.

Thus, there are many manually evaluated queries (the least confident cases) that we can use for evaluating our approach. We randomly selected 10,000 of these most difficult queries for evaluating our system: we built record pairs of the form *(query, correct result)* or *(query, incorrect result)*, depending on how the result has been evaluated by the expert. We can compare with the results of Schufa's current system and we can show whether our system would allow Schufa to save some of the (expensive) manual decisions without losing precision.

The automatically evaluated cases are not interesting for us: by comparing with Schufa's current system, we could only determine whether our system would act in the same way, without knowing whether these decisions are correct.

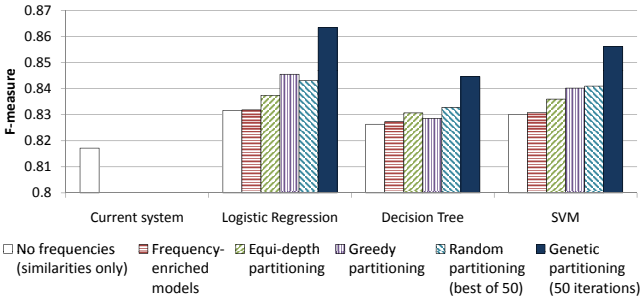

Figure 1: Comparison of frequency approaches and partitioning strategies for Schufa data set

Results. In this experiment, we compare the results of the learned similarity measures when applying all partitioning strategies described in Sec. 4.4. All results are determined using 10-fold CV on the 10k query data set containing difficult queries. We show the results in Fig. 1.

We observe improvements with partitioning for all learning methods and almost any partitioning strategy. The partitioning approaches also achieve better results than the frequency-enriched models.

For equi-depth partitioning, greedy partitioning, and random partitioning, we can see no clear leader across all learning methods. The achieved results are specific to each method. For all learning methods, genetic partitioning achieves the overall best results.

When comparing learning methods for genetic partitioning, logistic regression achieves better results than SVM, while decision trees are relatively far behind. Logistic regression can gain the largest profit from genetic partitioning. Thus, this method can even exceed SVM results. Decision trees profit less from genetic partitioning than SVMs or logistic regression.

We further show the result of Schufa's manually developed current system. In total, we can improve this system's F-measure by approx. 6%. This improvement translates to 600 of the 10,000 evaluated most difficult queries (with least confident results) and is a significant and valuable improvement for this use case.

5.2 Evaluation on DBLP Data Set

To investigate whether our approach also works for other data sets, we prepared another data set from DBLP [9], a bibliographic database for computer sciences. The main problem in DBLP is the assignment of papers to author entities. As described by Ley, DBLP is not perfect and needs to handle joins and splits: Author entries that falsely summarize publications from several authors need to be split into distinct author entries; author entries that represent the same entity, but have different names, need to be joined [9]. In contrast to other work that focuses on artificially injected errors [10] or specific groups of names [5, 7], we want to handle the actual DBLP problems on a larger scale. We constructed our data set from cleaned parts of DBLP, where different aliases for a person are known or ambiguous names have been resolved. In DBLP, cleaning is done manually (due to author requests) and automatically (using fine-tuned heuristics) [9].

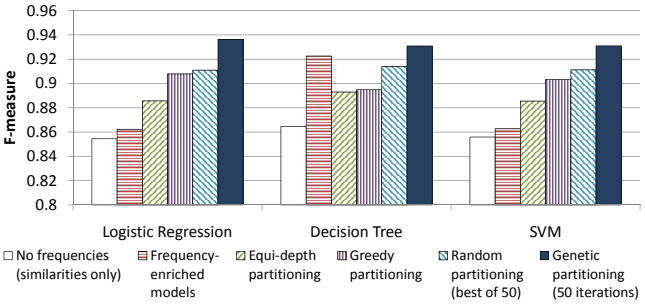

Figure 2: Comparison of frequency approaches and partitioning strategies for DBLP data set

Preparation. We created a new data set consisting of paper reference pairs that can be assigned to the following categories: (1) Two papers from the same author, (2) Two papers from the same author with different name aliases (e. g., with/without middle initial), (3) Two papers from different authors with the same name, (4) Two papers from different authors with different names.

For each paper pair, the matching task is to decide whether the two papers were written by the same author. The data set contains 2,500 paper pairs per category (10k in total). This does not represent the original distribution of ambiguous or alias names in DBLP (where about 99.2 % of the author names are non-ambiguous), but makes the matching task more difficult and interesting. We provide this data set on our website[1].

Results. Similar to the previous experiment on the Schufa data set, we applied 10-fold cross validation to compare the partitioning strategies from Sec. 4.4. The results in Fig. 2 look similar to the results of the Schufa data set. Without partitioning, all three machine learning techniques achieve similar F-measure results of about 0.86.

Incorporating only frequencies into these models (without partitioning) improves performance only by a small amount. An exception are decision trees, for which we can measure a significant improvement. For this data set, the decision tree algorithm could determine that the frequency attribute is relevant for the classification task.

From the partitioning strategies, genetic partitioning clearly outperforms all other strategies with overall F-measure results of about 0.93. The results show that random partitioning is not enough to achieve best results. The equi-depth and greedy partitioning strategies perform worse than the other partitioning strategies, but still better than the no-partitioning approach without frequencies. For this data set, too, partitioning always improves the results.

Overall, we can measure a significant improvement by genetic partitioning of about 9 %, which is larger than the improvement on the Schufa data set. Our experiments show that our approach works on different data sets, but that the actual amount of improvement depends on the data set.

6. CONCLUSION

With this paper, we introduced a novel approach for improving composite similarity measures. We divide a data set consisting of record pairs into partitions according to frequencies of selected attributes. We learn optimal similarity measures for each partition. Experiments on different real-world data sets showed that partitioning the data can improve learning results and that genetic partitioning performs better than several other partitioning strategies.

[1] `http://www.hpi-web.de/naumann/data`

Acknowledgments

We thank Schufa Holding AG for supporting this work. We especially thank Stephan Springob, Boris Zerban, and Michael Stolz for their valuable input. In addition, we are grateful to Tobias Vogel for fruitful discussions and Arvid Heise for his help on preparing the DBLP data set.

7. REFERENCES

[1] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA, Jan. 1998.

[2] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1, March 2007.

[3] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proc. of the ACM Intl. Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, pages 39–48, Washington, DC, USA, 2003.

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

[5] A. A. Ferreira, A. Veloso, M. A. Gonçalves, and A. H. Laender. Effective self-training author name disambiguation in scholarly digital libraries. In *Proc. of the Joint Conference on Digital Libraries (JCDL)*, pages 39–48, Gold Coast, Australia, 2010.

[6] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11, 2009.

[7] H. Han, L. Giles, H. Zha, C. Li, and K. Tsioutsiouliklis. Two supervised learning approaches for name disambiguation in author citations. In *Proc. of the Joint Conference on Digital Libraries (JCDL)*, pages 296–305, Tuscon, AZ, USA, 2004.

[8] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[9] M. Ley. DBLP – some lessons learned. *Proc. of the VLDB Endowment*, 2(2):1493–1500, 2009.

[10] V. Rastogi, N. N. Dalvi, and M. N. Garofalakis. Large-scale collective entity matching. *Proc. of the VLDB Endowment*, 4(4):208–218, 2011.

[11] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proc. of the ACM Intl. Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, pages 269–278, Edmonton, Alberta, Canada, 2002.

[12] W. Shen, P. DeRose, L. Vu, A. Doan, and R. Ramakrishnan. Source-aware entity matching: A compositional approach. In *Proc. of the Intl. Conf. on Data Engineering (ICDE)*, pages 196–205, Istanbul, Turkey, 2007.

[13] V. I. Torvik and N. R. Smalheiser. Author name disambiguation in MEDLINE. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3:11:1–11:29, July 2009.

[14] M. Weis, F. Naumann, U. Jehle, J. Lufter, and H. Schuster. Industry-scale duplicate detection. *Proc. of the VLDB Endowment*, 1(2):1253–1264, 2008.

[15] W. E. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau, 1999.