

Classification with Pedigree and its Applicability to Record Linkage

Evan S. Gamble, Sofus A. Macskassy, and Steve Minton

Fetch Technologies, 2041 Rosecrans Ave, El Segundo, CA 90245
{egamble, sofmac, minton}@fetch.com

Abstract

Real-world data is virtually never noise-free. Current methods for handling noise do so either by removing noisy instances or by trying to clean noisy attributes. Neither of these deal directly with the issue of noise and in fact removing a noisy instance is not a viable option in many real systems. In this paper, we consider the problem of noise in the context of record linkage, a frequent problem in text mining. We present a new method for dealing with data sources that have noisy attributes which reflect the pedigree of that source. Our method, which assumes that training data is clean and that noise is only present in the test set, is an extension of decision trees which directly handles noise at classification time by changing how it walks through the tree at the various nodes, similar to how current trees handle missing values. We test the efficacy of our method on the IMDb movie database where we classify whether pairs of records refer to the same person. Our results clearly show that we dramatically improve performance by handling pedigree directly at classification time.

1 Motivation

Real-world data is virtually never perfect and studies have shown that field level error rates around 5% or more are not uncommon [9]. In this paper, we consider the problem of noise in the context of *record linkage* applications that identify matching records that refer to the same entity but come from different data sources. In this scenario, we are given pairs of data records from one or more sources (generally databases) and are tasked with deciding whether they refer to the same entity (e.g., given two database person records, classify whether they refer to the same person). Machine learning can be used to create rules for classifying matching entities in the face of spelling mistakes, nicknames, abbreviation, missing words, etc. For example, a record linkage application should indicate that “Fetch Technologies” at “2041 Rosecrans Avenue, Suite 245” in “El Segundo” is highly likely to be the same company as “Fetch” at “2041 Rosecrans Av” in “El Segundo, CA”. However, some cases, the individual attributes may be completely incorrect or out of date. This may occur, for example, if a company moves. In our experience, this type of noise, where one attribute is completely incorrect, is likely to occur in text processing applications, because the problem of noise is aggravated by natural language extraction mistakes. In this paper we consider problems that occur when individual attributes are completely incorrect.

There is a large body of literature studying the problem of learning in the presence of noise [10, 7, 3, 1]. Data pre-processing has been proposed as a way to handle such noisy data, where two recently proposed ways of pre-processing such noisy instances

have been to either remove them [17] or clean them [5, 16]. There are problems with both of these. In the former, we are losing potentially good data as presumably most of the instance is good. In the latter we will still have a noisy instance unless we have an oracle that can clean up the instance. This paper proposes a new way of handling noisy attributes by dealing with them directly in the classification under the assumption that the training phase consists of clean data and that noisy data is only an issue during classification time. This is a reasonable assumption when the training set is small enough to clean by human inspection but there is too much data at classification time to clean by inspection. Attribute noise can be quantified during cleaning of the training set.

As mentioned above, there are many problems in the real world where it is to be expected that some attributes in fact have been misrecorded. We refer to the accuracy of the records for a given data source with respect to accurate recording of the data as the *pedigree* of that source. Some databases may have been populated from sources that were unreliable. If we know the nature of these source problems, then we could incorporate this knowledge into the classification process. For example, we may know that some of the data records come from mining a text source that is not very good at recording the birth date and that this field is wrong every so often—i.e., the source pedigree is less than 100%. Current ways of handling such a case involves identifying when the attribute is wrong and then either cleaning the attribute by “correcting” it, or ignoring the instance all together. Another possibility is to completely ignore this attribute during training [13]. None of these deal directly with the attribute, however. The method proposed in this paper is to recognize the fact that the birth date is wrong every so often and use the uncertainty of the correctness during classification time. We show, in a case study on record linkage, that our method improves performance over not taking this uncertainty into account.

This paper formally presents the problem of *source pedigree* (attribute noise) and enumerates the ways in which a classification system handles attribute noise in a more direct manner than what has previously been suggested. We expand on one of these methods, namely having a learned classifier model change its classification based on the likelihood that each attribute may be wrongly reported. We make the simplifying assumption that data is clean during training and that attribute noise is present only at classification time.

We next describe related work, followed by the formal description of the source pedigree problem. We then describe the plausible solutions and our specific method, followed by an empirical case study in the domain of record linkage, and conclude with a discussion of the results.

2 Related Work

The literature on learning in the presence of noise is quite extensive, both in the standard propositional setting [11, 1] as well as in relational settings [3]). As is usual in machine learning, all of these cases assume that the training data is drawn from the same distribution as the test set and that the both exhibit the same type of noise. In these cases, the best performance an optimal learner can achieve is to match the class

noise. Various techniques and proofs have shown under what circumstances the learner can get arbitrarily close to this optimum [2].

A different view of attribute noise is that the attribute carries little or no information—i.e., that it is irrelevant, but through random chance that it may appear relevant to the prediction task [7, 6]. In this case, the task is to identify and then ignore these irrelevant attributes.

One alternative approach to learning in the presence of noise is to pre-process the data and clean it up before learning from it [5, 4, 17]. A recent comprehensive study on how to handle attribute noise is the most directly relevant piece of work to this paper [16]. In this comprehensive work, Zhu and Wu perform an empirical study on the effects of attribute noise as well as the effect of cleaning up either the training data and/or the test data. It was found that having a dirty test set universally did worse than having a clean test data set, where cleaning the test set would perform better with a dirty training set than vice versa. One main focus of Zhu and Wu is to propose ways of handling noise. In this particular work, they focus on handling attribute noise and describe an overall design framework in which this can be done. This design is geared towards predicting whether an attribute is noisy and then “correcting” its value by inference or other means. Recent work, however, seem to indicate that when you have noisy attributes, it may be better to ignore them altogether over trying to “correct” them [13].

3 Classification with Pedigree

Formally, our domain is the general classification problem, modified by the addition of pedigree information.

As in standard supervised learning, we are given a training set of records, each of which is a vector of attribute values, either numeric or literal. For each record $x = (x_1, \dots, x_n)$ in the training set we are given a label $y \in \{-1, 1\}$. After training, for a record x the classifier will produce a value $s \in [0, 1]$ which is the probability that $y(x) = 1$.

In addition to the normally available information, with *source pedigree* each record $x = (x_1, \dots, x_n)$ for both training and classification also has an associated vector of *pedigrees* $p = (p_1, \dots, p_n)$ where $p_i \in [0, 1]$ is the pedigree of the i th attribute value. A pedigree p_i for an attribute value means we believe the attribute value to be accurate with probability p_i , otherwise it is inaccurate. *Accurate* means that it is the correct value for that attribute in that record and *inaccurate* means the value is essentially randomly drawn from the set of possible values for that attribute. We assume that the distribution of inaccurate values will be approximated by a uniform distribution over all the values for that attribute in the training set.

If all pedigree vectors are $(1, \dots, 1)$ the problem reduces to the usual classification problem. If all pedigree vectors are $(0, \dots, 0)$ the classifier cannot trust any of the values and will have to predict based on the class prior.

Although our setting allows each record to have a different pedigree vector for generality, in practice we anticipate that blocks of records would share the same pedigree vector. The records in a typical classification problem would derive from different data sources, with pedigree specified per data source. In our set up of the problem we do not

explicitly model different data sources, but records from a particular data source may be thought of as a block of records in the training or classification sets.

The goal is to find a way to use partially reliable attribute values at classification time to the extent permitted by the pedigrees.

4 Plausible Solutions to Handling Pedigree

We assume that for training we have available a set of records with perfect reliability, meaning the associated pedigree vectors are $(1, \dots, 1)$. We have not yet explored formal solutions using training data with imperfect reliability, but we plan to address this in a future study.

In this setting, we need to have the classifier be able to handle noisy test data at classification time. Plausible solutions fall into two types: those where the records are modified by the pedigrees before the classifier runs and those where the records and pedigrees are combined within the classifier.

4.1 Pre-processed Pedigrees

Pre-processing the records with pedigree information allows the classifier to be treated as a “black box”, so any existing classifier could be used without modification. In this scenario, we tell the classifier what the source pedigree is and let the classifier itself learn the appropriate model. A disadvantage of this is that the pedigree vector for each block of records (equivalently, each data source) must be known at training time and represented in the training set such that the classifier can learn from them. One added complexity is that we also need to introduce noise into the training set according to the pedigree vector (remember that we assume our training set is clean).

For completeness, we present here one possible pre-processing method, although this is not the method of choice nor what we will be using in our case study. For each unique pedigree vector p we assign a unique literal identifier P . Given a record x in either the training or classification sets with associated pedigree vector p we augment the record with an additional attribute with value P . The augmented record becomes (x_1, \dots, x_n, P) . The additional attribute lets the classifier learn a different sub-model (distribution, tree, etc.) for each unique pedigree.

We explicitly incorporate pedigree information into the training set as follows. We oversample the training data by some multiple that depends on the precision of pedigree training desired. For each record with pedigree vector p in the training set, we modify the augmented similarity vector (x_1, \dots, x_n, P) as follows. For each attribute i , with probability $1 - p_i$ we replace x_i with a random value from the set of possible values for that attribute. Ideally, the random value should be chosen from all of the i th values in the training set to match as well as possible the distribution of all correct values for the i th attribute.

The modified records, incorporating random substitutions according to pedigree and augmented with pedigree identifiers, are used to train the classifier. The label y provided for training has the same value for a given record whether or not random substitutions have been made in the attribute values—i.e., we do not consider noisy labels.

4.2 Pedigrees Used at Classification Time

Designing a classifier to use pedigrees only at classification time, rather than as a pre-processing step for training and classification, has a couple potential advantages. First, the new data sources with new pedigrees could be used at classification time without retraining. Second, oversampling of the training data is not required, with potentially large savings of time and space.

We chose to design our own decision tree classifier (DTP) based on C4.5 [12] and J48 [15] that uses pedigree vectors at classification time. Because training data is assumed to have perfect reliability, and pedigree vectors are not used during training, the decision trees produced during training are the same as produced by the original C4.5 and J48 programs.

At classification time, DTP uses the input records and the associated pedigree vectors to determine the paths to take through the decision tree. During the search of the decision tree, when a node condition refers to an attribute with partial reliability, each of the direct branches from the node are traversed. The final values produced for each branch are combined in a proportion dependent on the pedigree of the attribute value. The following section will describe the details of this.

5 Decision Tree Classifier Using Pedigree

The Decision Tree classifier using pedigree (DTP) algorithm based on C4.5 and J48 works as follows. The decision tree is generated from the training set of records without using pedigrees, exactly following C4.5 and J48.

At classification time, each input record x and associated pedigree vector p is given to the classifier.

Classification proceeds as a normal decision tree with a search of the tree from the root. A decision is made at each node as to which of the branches to follow based on the result of comparisons of an attribute value with one or more constants. For a given node with a condition involving x_i , if $p_i = 1$ then DTP does as C4.5 and follows only one branch, eventually producing s , the final probability that $y(x) = 1$. However, if $p_i < 1$, all m direct branches are taken, giving m final probabilities, $\{s_1, \dots, s_m\}$. In that case, we want to combine $\{s_1, \dots, s_m\}$ together in some way to produce s .

As a by-product of the generation of the tree during training, the m direct branches have weights $\{w_1, \dots, w_m\}$ with $\sum w_j = 1$, where w_j is the ratio of training examples at that node that went down branch j .

Let's consider the case where $p_i < 1$ and the k th branch is the "correct" branch, meaning the branch for which the condition involving x_i is true—i.e., based on the given value of x_i , that is the branch that should be followed. Then we combine $\{s_1, \dots, s_m\}$ according to their weights w_j , weighing the correct branch more heavily in proportion to p_i :

$$s = s_k(p_i(1 - w_k) + w_k) + (1 - p_i) \sum_{j \neq k} s_j w_j \quad (1)$$

As mentioned above, if the k th branch is correct and $p_i = 1$, the equation for s reduces to $s = s_k$ which is the normal evaluation for the node. Similarly, if $p_i = 0$ then $s =$

$\sum s_j w_j$ which is the expected value for s given that nothing is known about the true value of x_i .

This procedure is repeated at every node during the partial traversal of the decision tree. If $p_i < 1$ for all i , then every node in the tree will be traversed.

6 Empirical Study

We here present an empirical study on the efficacy of our method. The study is in the domain of record linkage as this method was developed in order to handle source pedigree in this setting. We will therefore present enough details about the record linkage problem such that the reader can appreciate the results presented.

6.1 Source Data

Our empirical study of the DTP algorithm is based on a record linkage problem involving the Internet Movie Database (IMDb).¹

We have a set of records of people from the IMDb: actors, directors, etc. Each person record is a vector of ten fields, although only three were ultimately used by the classifier: (*Name*, *DateOfBirth*, *Spouse*). For simplicity, we will describe the records as if they were composed of only those three fields.

We simulated imperfect source pedigree by introducing noise into the *Spouse* field. In 25% of the records used for testing (not training) the *Spouse* value was replaced by the *Spouse* value of another random record. *Spouse* is normally a strongly identifying field, meaning two records with the same *Spouse* value are very likely to refer to the same person.

Some people in the IMDb have multiple *Name* values (nicknames, screen names, etc.) and some have multiple *Spouse* values (presumably from consecutive marriages). A person with n names and m spouses is represented by $n \cdot m$ records in our set.

Not all people in the IMDb had *DateOfBirth* or *Spouse* values, in which case those fields have a null value.

6.2 Classifier Input Vectors

Given two records from a set of person records, our record linkage system generates a vector of attribute similarity scores using a learned similarity metric [8]. The similarity metric involves several types of transformations between the tokenized field values including Equal, Misspelling, Synonym, Concatenation, Abbreviation, Prefix, Suffix, Acronym, Soundex, and Missing. The numbers of each type of transformation are combined according to learned weights.

The vectors of attribute similarity scores for pairs of records become inputs to the DTP classifier to classify the pair as referring to the same person or not. Since considering all possible pairs of records is not tractable in a real setting, during both training and testing, pairs of records first pass through a simple blocking stage to eliminate likely mismatches, as in [14]. Once pairs have made it past blocking, their similarity vectors are produced. These then make up the training and test sets.

¹ <http://www.imdb.com>

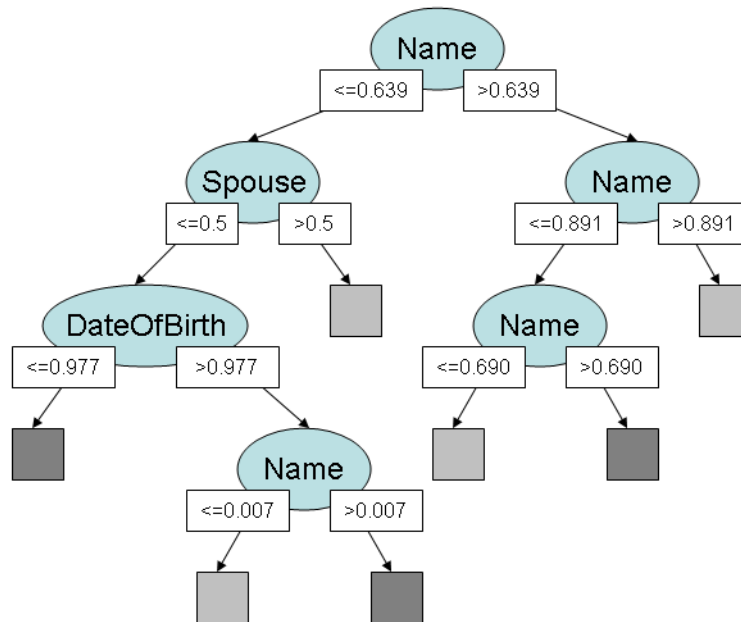


Fig. 1. Decision tree built on training set. The darker boxes denote the non-match classification and the lighter boxes a match classification.

6.3 Training

For training we chose a set of 1111 person records, randomly divided into two subsets of 550 and 561 records. Pairs of records composed of one record from each subset were drawn randomly. Of those pairs that made it through blocking, 7607 were negative examples, i.e., pairs of records referring to different people, and 1079 were positive examples. Training produced a decision tree with 6 nodes which correctly classified all but 60 negative and 10 positive examples. The resulting tree is shown in Figure 1.

6.4 Testing

Our test set consisted of 476 person records, representing about 86 people. Noise was added to the *Spouse* field for 25% of the records by substituting the *Spouse* value of another random record.

Our record linkage system progressively builds a “person table” by adding each test record as they are presented to the system. The person table is initially empty. We ordered the 476 records randomly and added them one at a time. For each such record r , all pairs consisting of r and one of the records in the growing person table are sent through blocking. Similarity score vectors for those record pairs that survive blocking are sent to the classifier to determine the probability that the pair is a match, i.e. refers to the same person. If the best fitting record q in the person table has likelihood $> 90\%$ of being a match with r , then r is added to the person table as an outer join on the person

ID of q . Otherwise, r is added to the person table with a new, unique person ID. This is a subtle issue, but if we have a match on an existing record, then this results in multiple records being added to the person table as each possible combination of attributes need to be recorded. This is done so that subsequent incoming records of the same person will have a better chance of finding a highly similar record in the person table.

6.5 Results

We ran the record linkage system twice on the test set, once with a pedigree vector of $(1, 1, 1)$ for each similarity score vector, and once with a pedigree vector of $(1, 1, 0.75)$, meaning the *Spouse* similarity score had a pedigree of 0.75, corresponding to the 25% noise we added to the *Spouse* field of the test set.

For the purpose of illuminating the behavior of the DTP classifier, we examine the likelihood scores produced by the classifier on all record pairs passing through blocking, rather than examining the final person tables. In both cases, pedigree=1 and 0.75, over 21000 record pairs passed through blocking to be evaluated by the classifier.

We report the results of classification using a precision-recall curve. The precision-recall curve is generated by considering, for each potential likelihood score produced by the classifier, how well using that score as a definitive threshold would do with respect to precision and recall. For example, if we consider the likelihood score of 0.8, we can consider all the record pairs that got a score above 0.8 and ask how many of those were true positives—this is the precision at score=0.8. We can also ask out of all the record pairs that are true positives how many of those got a score over 0.8—this is the recall at score=0.8. If we plot the precision and recall at every distinct score and join the dots we get a precision-recall curve.

Figure 2 shows the precision-recall curve for the two runs—one with pedigree=1 and one with pedigree=0.75. As we can see, their performance is very different even up to a high level of recall (0.95), where using pedigree we get a much higher precision than the run that does not use pedigree. It is obvious that they will both have the same precision at recall=0 and that they will converge to the same point at recall=1, but their performance in between is the interesting part. Their differences are the highest at recall = 0.2486 (about 25% of all the true positives have been accounted for). In this case, using pedigree we get a precision near 1 whereas not using pedigree has a precision of 70%. This difference is very large, when we consider that we had over 21000 total pairs of records, where about 15% of them were true matches. At a recall of 25% (800 records), using pedigree only about 10 extra records were incorrectly classified, whereas not using pedigree about 1150 records were incorrectly classified.

To shed some light on records where not using pedigree hurt and using pedigree generated proper scores, we identified the pairs of records where the two runs had different predictions. Table 1 shows two such pairs. Remember in Figure 1 that one node split on the *Spouse* attribute with one leaf being a match and another a non-match. In the pairs shown in the Table, we see that they wrongly shared the same spouse and in both cases they would have gone down the wrong branch from the *Spouse* node in the tree. Using pedigree, however, was enough to cast into doubt which branch to take and DTP therefore combined scores from both branches, enough to push the match score low.

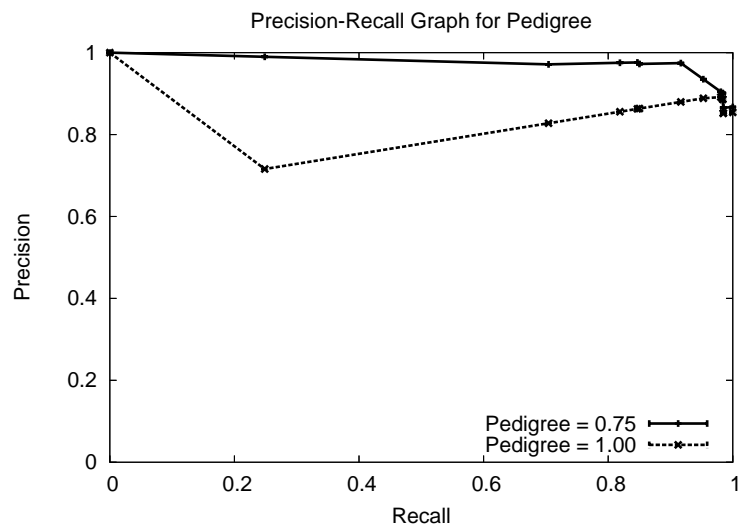


Fig. 2. Precision-recall curve comparing record linkage with and without pedigree. As we can see, using pedigree can dramatically improve performance, especially at low levels of recall.

7 Discussion and Limitations

We presented a new methodology for handling attribute noise directly rather than through the current standard of either removing noisy instances or cleaning noisy attributes, neither of which are viable options in a real world system.

Our proposed method uses a decision tree which changes its classification score based on a specified source pedigree. We tested DTP (Decision Tree with Pedigree) in a real world scenario to perform record linkage on people from the online movie database IMDb. We deliberately introduced noise to test the efficacy of DTP and the results clearly show that using pedigree information dramatically improved the results when viewed on a precision-recall curve.

We do note that our current method has two significant limitations. First, we assume that the training data is clean. This is appropriate for some applications, but for those applications where only noisy training data is available new methods are needed, which we will address in a future study. Second, we assume that some outside oracle will provide us with pedigree scores either for a complete data source or on a per-record basis. Again, this assumption is not likely to be the case although one could assume that the user has some idea about the noise of the data sources. Both of these limitations define clear open research questions which we hope will be addressed by the community.

References

1. Aha, D.W., Kibler, D.: Detecting and removing noisy instances from concept descriptions. Technical Report 88-12, University of California, Irvine (1989)

Name	DoB	Spouse		Name	DoB	Spouse
Julie Ege	11/12/1943	Rob Johnson	≠	Debbi Lee Carrington	12/14/1959	Rob Johnson
Jeff Gordin	8/4/1971	Jennifer Sealy	≠	Jeff Glenn Bennet	N/A	Jennifer Sealy

Table 1. Two example pairs of records where using pedigree correctly predicted that they are not referring to the same person whereas not using pedigree wrongly identified the people to refer to the same person (i.e., that Julie Ege and Debbi Carrington were the same and that Jeff Gordon and Jeff Glenn Bennet were the same). This is due to the fact that the noisy *Spouse* value had them share (wrongly) the same spouse, which the no-pedigree classifier used wrongly, whereas the pedigree classifier took the uncertainty of that information into account.

- Aha, D.W., Kibler, D., Albert, M.K.: Instance-based learning algorithms. *Machine Learning* (1991) 37–66
- Brunk, C.A., Pazzani, M.J.: An investigation of noise-tolerant relational concept learning algorithms. In: *Proceedings of the 8th International Workshop on Machine Learning*, Morgan Kaufmann (1991) 389–393
- Elnahrawy, E., Nath, B.: Cleaning and querying noisy sensors. In: *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*. (2003)
- Gamberger, D., Lavrac, N., Dzeroski, S.: Noise detection and elimination in data preprocessing: experiments in medical domains. *Applied Artificial Intelligence* **14** (2000) 205–223
- Khoshgoftaar, T.M., Hulse, J.D.V.: Empirical case studies in attribute noise detection. In: *IEEE International Conference on Information Reuse and Integration*. (2005) 211–216
- Littlestone, N.: Redundant noisy attributes, attribute errors, and linear threshold learning using winnow. In: *Proceedings of the fourth Annual Workshop on Computational Learning Theory (COLT)*. (1991)
- Minton, S., Nanjo, C., Knoblock, C.A., Michalowski, M., Michelson, M.: A heterogeneous field matching method for record linkage. In: *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM)*. (2005) 314–321
- Orr, K.: Data quality and systems theory. *CACM* **41**(2) (February 1998) 66–71
- Quinlan, J.R.: Learning efficient classification procedures and their application to chess endgames. In Michalski, R.S., Carbonell, J.G., Mitchell, T.M., eds.: *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann, Los Altos, CA (1983) 463–482
- Quinlan, J.R.: Decision trees as probabilistic classifiers. In: *Proceedings of the Fourth International Machine Learning Workshop*, Irvine, CA (June 1987) 31–37
- Quinlan, J.R.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA (1993)
- Saar-Tsechansky, M., Provost, F.: Handling missing features when applying classification trees. Technical Report CeDER Working Paper CeDER-05-19, Stern School of Business, New York University (2005)
- Tejada, S., Knoblock, C.A., Minton, S.: Learning domain-independent string transformation weights for high accuracy object identification. In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. (2002)
- Witten, I.H., Frank, E. In: *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco (2000)
- Zhu, X., Wu, X.: Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review* **22**(3) (November 2004) 177–210
- Zhu, X., Wu, X., Chen, S.: Eliminating class noise in large datasets. In: *Proceedings of the 20th International Conference on Machine Learning (ICML)*. (2003) 920–927