Comparative evaluation of entity resolution approaches with FEVER

Hanna Köpcke, Andreas Thor, Erhard Rahm
Department of Computer Science, University of Leipzig
04009 Leipzig, Germany
{koepcke, thor, rahm}@informatik.uni-leipzig.de

ABSTRACT

We present FEVER, a new evaluation platform for entity resolution approaches. The modular structure of the FEVER framework supports the incorporation or reconstruction of many previously proposed approaches for entity resolution. A distinctive feature of FEVER is that it not only evaluates traditional measures such as precision and recall but also the effort for configuring (e.g., parameter tuning, training) a good entity resolution approach. FEVER thus strives for a fair comparative evaluation of different approaches by considering both the effectiveness and configuration effort.

1. INTRODUCTION

Entity resolution (also referred to as object or entity matching, duplicate identification, record linkage or reference reconciliation) is a fundamental problem for data integration and data cleaning [11]. It is the task of identifying entities referring to the same real-world object. The high importance and difficulty of the entity resolution problem has triggered a huge amount of research on different variations of the problem and numerous approaches have been proposed especially for structured data. For a recent overview survey and a tutorial see [5] and [8], respectively.

Due to the high number and diversity of different entity resolution approaches we see a strong need for comparative evaluations of different schemes. To date most entity resolution approaches have been evaluated individually using diverse methodologies, configurations, and test problems making it difficult to assess the overall quality of each approach, let alone their comparative effectiveness and efficiency. Only few attempts for comparative evaluations of some sub-approaches have been made, e.g., evaluation of different string similarity metrics [4] and of blocking approaches [1]. Some benchmark proposals for entity resolution have been made [10], [13] but they have not yet been implemented or applied.

In this demo paper we present FEVER – a framework for evaluating entity resolution. FEVER is not yet another entity resolution approach but serves as a flexible evaluation platform for a large spectrum of entity resolution algorithms and strategies building upon and extending our previous prototypes MOMA [12] and STEM [7]. FEVER offers the following key features:

- FEVER supports the flexible construction and comparative evaluation of many different entity resolution workflows based on so-called operator trees. Operator trees support the combined application of different blocking and match algorithms in order to achieve a high effectiveness (precision, recall). Individual operator implementations can be based on virtually any previously proposed algorithm, e.g., for blocking or entity matching, so that these can be evaluated with FEVER. Entity resolution methods may be based on machine learning techniques utilizing training data. Hence, FEVER can be used to compare non-learner and learner-based approaches for entity resolution.
- FEVER allows each match approach to be automatically executed and evaluated under different parameter configurations. We first use this feature to determine the necessary effort for training and parameter tuning (e.g., finding suitable similarity thresholds) to obtain a reasonable match quality. This way we can conduct a fair comparison of different entity resolution algorithms under comparable tuning effort. Hence, an approach A with better effectiveness than approach B is only superior if it does not incur a substantially higher configuration effort.
- FEVER can also be used to fine-tune match approaches by letting the system automatically evaluate a large number of parameter settings for test data. The best performing configuration can then be used for subsequent match tasks on similar and larger input data.

In the following we give an overview of FEVER and present the operator library. We then describe FEVER's effort-based configuration strategies that facilitate a fair algorithm comparison. We conclude with a description on what will be demonstrated.

2. OVERVIEW OF FEVER

Figure 1 illustrates the architecture of FEVER consisting of modules for the definition and execution of entity resolution approaches or so-called match workflows. A GUI-based workflow editor allows to interactively specify match workflows utilizing operator trees. The FEVER runtime executes a specified workflow and, thus, generates evaluation data.

A match workflow specification has three parts: input data definition, operator tree, and configuration specification. All parts can be edited in the workflow editor. Figure 2 shows a screenshot of the FEVER's main window with portions of the workflow editor on the left side. The right side shows a sample evaluation result for different match workflows.

VLDB '09, August 24-28, 2009, Lyon, France.

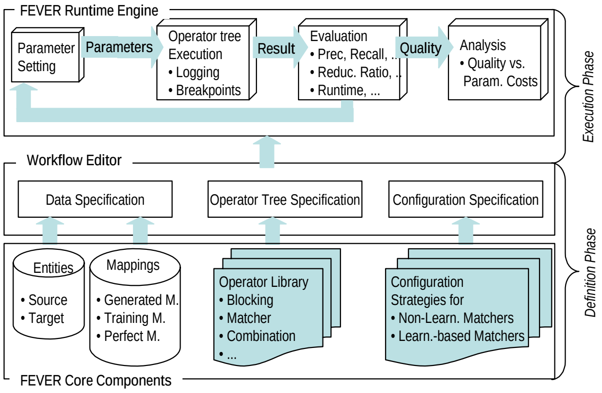


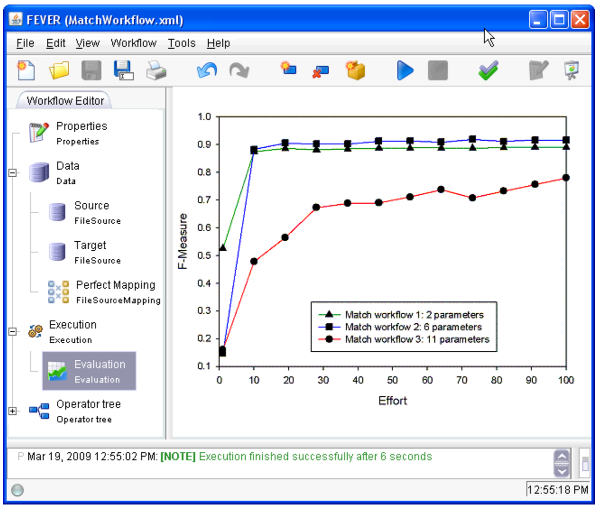
Figure 1: Architecture of FEVER

The input data includes the entities of one or two data sources to be resolved, and the perfect match result for evaluating the effectiveness of the entity matching workflow. For training-based match approaches, the training data also needs to be provided. The actual match workflow is declaratively described in terms of an **operator tree**. The input data sources form the leaves of the tree and non-leaf nodes are operators. The output of the root operator is the final match result specifying the identified entity correspondences within a so-called mapping (see next section). FEVER provides an extensible **operator library** with a variety of operators for training selection, blocking, matching, and mapping combination. Operators define their expected input and delivered output as well as their mandatory and optional parameters. The shape of the operator tree determines the order in which operators are applied. An operator tree is well formed if all operators are nested correctly, provided with their necessary input and mandatory parameters are set.

The operator tree concept provides a high flexibility to specify a tailored workflow for a given match task and supports its comparative evaluation with other approaches. In particular, it allows the selection and combination of several match approaches. Developed operator trees can be saved as building blocks and can then be reused as a sub-tree in other workflow definitions. The operator tree concept and the supported operators are described in the following section.

The third match workflow component specifies the execution of the operator tree according to a **configuration strategy** defining the parameter settings of all operators of the tree. For evaluation purposes the operator tree is usually executed multiple times with different parameter settings. This way the effectiveness of the match workflow for different settings can be evaluated and the effort to find an effective configuration can be determined. We present configuration strategies in more detail in Section 4.

Specified workflows are stored in a repository and can be executed by the FEVER runtime (upper part of Figure 1). The runtime supports an iterative execution model to execute an operator tree several times for different parameter settings. The match result of each operator tree execution is automatically evaluated with the help of the given perfect result and, thus, evaluation measures such as precision, recall, and F-measure can be determined. In addition, performance indicators are recorded, e.g., execution runtime and used main memory. The calculated quality measures are stored and can be compared to the effort of configuring the parameter setting. Hence, FEVER not only allows an evaluation of the match quality, but also facilitates an analysis of the effort expended to reach the respective match quality. As a consequence, FEVER supports a fair comparison of match algorithms by comparing the match quality reached under the same effort.



Mar 19, 2009 12:55:02 PM: **[NOTE]** Execution finished successfully after 6 seconds

Figure 2: GUI of FEVER

For smaller datasets the execution of a match workflow can be started from the GUI and the results interactively inspected via the build-in plotter. This feature will be used during the demonstration (see Section 5). For larger datasets the match workflow can be modeled within the GUI and then executed in batch mode. The FEVER implementation is written in Java and uses the Rapid Miner [9][1] library of machine learners.

3. OPERATOR TREES

FEVER allows the definition of entity matching algorithms by operator trees. Operator trees are a common modeling concept for numerous database problems (e.g., query optimization) and have previously also been used to model entity resolution approaches [3]. We build on those previous approaches and extend the idea to a flexible method for defining match workflows. The tree is executed in a post-order traversal sequence and the results of the child operators are input to the father operator.

FEVER's operator library offers a variety of operators. The main operator types for blocking, matching and training selection generate **mappings** as operator output. A mapping *m* between two sets of entities $A \subseteq S_A$ and $B \subseteq S_B$ from two sources S_A and S_B consists of a set of match correspondences, i.e., $m = \{(a, b, s) | a \in A, b \in B, s \in [0,1]\}$. The similarity value *s* indicates the strength of the similarity between two entities $a \in A$ and $b \in B$. The uniform mapping data structure is the foundation for the flexible combination of operators within trees. Some previous match approaches represent their results as clusters of entities that are considered to be the same. FEVER can also support such methods by interpreting clusters as a mapping containing pairwise correspondences between all entities of the cluster.

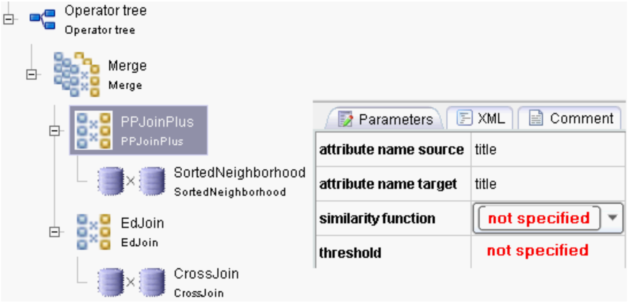
[1] http://www.rapidminer.com

Blocking operators expect as input one or two sets of entities and return a mapping. Blocking is needed for large inputs to reduce the search space for entity resolution from the Cartesian product to a small subset of the most likely matching entity pairs. Besides the *CrossJoin* (which retains all possible entity pairs of the Cartesian product) FEVER supports disjoint (e.g., (multi-pass) sorted neighborhood) as well as overlapping blocking methods (e.g., bigram indexing, canopy clustering). Disjoint methods build mutually exclusive blocks, whereas overlapping methods may result in overlapping blocks of entities. All blocking operators have to specify a blocking key. Additional parameters such as the window size for the sorted neighborhood operator might be required.

The **match operators** form an integral part of the operator library. They require as input a mapping resulting either from the previous application of a blocking operator or another match operator. The input mapping specifies the pairs of entities that should be compared by the match approach and, thus, limits the match complexity. FEVER supports non-learning matchers as well as learning-based matchers. For **non-learning matchers (NLM)**, our current implementation incorporates a variety of previously proposed attribute matcher or similarity join algorithms (e.g., EdJoin and PPJoinPlus [14], PartEnum [2]). The similarity join operators match entities based on the similarity of a single attribute pair. Further parameters include the (string) similarity measure to be applied (e.g., edit distance, cosine, n-gram, TF/IDF) and the threshold above which entities are considered to match. A multi-attribute matcher is also supported which directly evaluates and combines the similarity for multiple attribute pairs.

Learning-based matchers (LM) are complex match approaches utilizing supervised learning algorithms such as SVM (support vector machine), decision trees and logistic regression to automatically find an effective combination of non-learning matchers such as attribute matchers. In addition to an input mapping, LM require a training mapping that contains manually labeled correspondences representing examples for matching (similarity value equals 1) and non-matching (0) entities. Furthermore, LM expect as a parameter a list of attribute matcher specifications indicating which similarity measure should be evaluated on which entity attributes. The LM operator applies the specified similarity measures to the attribute pairs in the training mapping. The learner uses the resulting similarity values to automatically determine a combination of the specified similarity measures for a match decision. The learned combination is then applied during the LM execution to the input mapping to determine the match correspondences.

Training mappings are provided by **TrainSelect operators**. They select correspondences from an input mapping and prompt users to label them interactively as match or non-match. By doing so the correspondences are annotated with similarity 0 (non-match) or 1 (match) and, thus, a training mapping is compiled. Labeled correspondences are additionally stored in a repository to avoid repeated labeling of the same correspondence. The input mapping may be the result of a previous blocking operator execution, e.g., *CrossJoin* for smaller datasets. However, an approximate match result may also be used as input to make sure that only "interesting" correspondences are labeled, e.g., correspondences whose entities fulfill a minimal similarity. The chosen TrainSelect operator determines the resulting training mapping of a specified size. Currently, FEVER offers two operators for training selection: *Random* and *Ratio*. *Random* randomly selects correspondences from the input for labeling. *Ratio* reduces a randomly selected set of correspondences so that a defined ratio between match and non-match correspondences is guaranteed. This approach is helpful to avoid an over-fitting of the learning-based match approach. The TrainSelect operators support a fair comparison of learning-based matchers by ensuring that learners are provided with training data of the same size and quality.



attribute name source	title
attribute name target	title
similarity function	not specified
threshold	not specified

Figure 3: Graphical representation of an operator tree (left) incl. parameters for selected similarity join operator (right)

Besides the operators for blocking, matching and training selection FEVER also offers auxiliary operators for combining and filtering entities and mappings. Figure 3 shows a sample operator tree merging the results of two independently executed match approaches. The first approach applies a PPJoinPlus similarity join [14] on the output of a Sorted Neighborhood blocking method. The second approach computes an EdJoin on the whole Cartesian product (cross join) of entity pairs.

4. CONFIGURATION STRATEGIES

An operator tree typically comprises several operators each having several parameters that need to be specified in order to apply the operator tree to a match problem. Typical operator parameters have been discussed above. Permissible parameter values can mostly be defined by a set of possible values, or by a range of real or integer numbers. For example, the similarity join operator *PPJoinPlus* (see Figure 3) has a set parameter to specify the similarity measure to be applied (Cosine or Jaccard in this case) and a range parameter for the similarity threshold.

We further distinguish between bounded and free parameters. A bounded parameter is already assigned a value through the user in the operator tree definition. Parameters that are not bounded are called free parameters. They are treated as parameters of the operator tree and, thus, have to be assigned a value dynamically according to a configuration strategy. For the example in Figure 3, the names of the attributes to be compared are bound parameters (manually provided) while the similarity function and threshold are free parameters.

An assignment of all free parameters of an operator tree with a valid value is called a **parameter setting**. For operator trees without learning-based match operators the parameter setting is sufficient for the configuration. Training-based operator trees additionally require a training mapping for the configuration. To allow for comparative evaluations, we take an effort-based approach. We evaluate the quality of an operator tree against the effort spent to determine the match configuration. We consider both, the **parameterization** and the **labeling effort**. The parameterization effort can be represented as the number of parameter settings that have been evaluated to identify the best setting, i.e., the setting for which the operator tree result has the best quality (e.g., F-measure). For training-based workflows the labeling effort regards the number of correspondences that have to be labeled by the user. We can thus ensure equal labeling effort for a comparative comparison of different training-based approaches. Parameterization and labeling effort are considered independently and are not set off into a single effort measure. This facilitates an analysis of the reached match quality measured in terms of a quality measure (e.g. precision, recall, f-measure) against the effort spent for parameterization as well as for labeling. In future work, we will investigate how to automatically determine a good trade-off between the expended effort and the reached quality.

The **evaluation** of an operator tree is subject to a specified maximal labeling effort *M* and maximal parameterization effort *N*. Hence, for training-based operator trees we restrict the user to label *M* training pairs. Additionally, we generate *N* different parameter settings according to a specified configuration strategy and determine from all obtained evaluation results the best ones. A configuration strategy takes as input the free parameters of an operator tree and the maximal parameterization effort *N*. To generate *N* different parameter settings FEVER currently supports the following configuration strategies. In a **user defined** strategy the parameter settings are completely specified by the user. This manual strategy can be applied if a defined set of parameter values should be evaluated, e.g., a threshold parameter value varies from 0 to 1 in 0.05 steps. The **random** strategy is a straightforward way that assigns parameter values out of their possible values randomly, i.e., *N* parameter settings are selected by random assignment of parameter values. The **grid** strategy realizes a simple grid search, dividing the multidimensional parameter space into a uniform grid. The coarseness of the grid is controlled by the number of parameter settings *N* and, thus determines the search efficiency and the quality of the solution. The sophisticated **gradient descent** strategy is more goal-oriented and iteratively refines a parameter setting by considering the quality of previously generated settings. We have adapted the Hooke-Jeeves method [6] for this strategy. Note that all (except user-defined) strategies are independent from the match approach modeled by the operator tree. Thus the configuration strategies can be applied to different match workflows.

5. DEMONSTRATION DESCRIPTION

Our demonstration will illustrate how FEVER is used to model and evaluate entity resolution approaches for different datasets.

First, we will show how to set up typical match workflows with FEVER. For providing the necessary input sources of a match task we will demonstrate the data import from file and from a database. We then show how to interactively construct an entity resolution approach by selecting and adding operators to the operator tree. We will start with a very simple operator tree, e.g., consisting of just one blocking and one matching operator. The tree can then be executed and the resulting mapping inspected. To analyze the sensitivity of the operator tree towards the chosen configuration we will provide the perfect mapping and apply the user defined configuration strategy to test some manually selected configurations. To showcase a fair comparison evaluation we will define an alternative operator tree for the same input sources and demonstrate the application of the random, grid or gradient descent configuration strategy. We will save both simple trees as building boxes and then set up a third more complex tree reusing the simple trees as sub-trees of a merge operator. This allows for a comprehensive analysis and a study of whether the sub-tree combination may improve the overall result quality and - if yes - how the sub trees should be combined. Another demonstration scenario focuses on the parameterization and effort-based comparison of learning-based operator trees. In particular, we can demonstrate the influence of training data by varying the size and selection method. Furthermore, we can compare the effort and quality between learning-based and non-learning-based entity resolution methods.

All evaluation results can be visually displayed by graphs such as the one illustrated in Figure 2 indicating the F-measure results for three operator trees under different configuration effort values. In the shown example, a simple operator tree with few parameters such as operator tree 1 needed a very small configuration effort to reach a stable level. For low configuration efforts the medium-sized and complex operator trees were inferior to this simple operator tree, but with more configuration effort the medium tree eventually outperformed the simple match workflow.

We finally demonstrate how FEVER can be used to fine-tune a selected match workflow. We do this by setting a relatively high parameterization effort and inspect the parameter settings for the best-performing configurations. The found settings can then be applied as a manually specified configuration for a larger or similar match task.

REFERENCES

[1] Baxter, R., Christen, P., Churches, T.: A comparison of fast blocking methods for record linkage. *Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003.

[2] Arasu, A., Ganti, V., Kaushik, R.: Efficient Exact Set-Similarity Joins. *VLDB*, 2006.

[3] Chaudhuri, S., Chen, B.-C., Ganti,V., Kaushik, R.: Example-driven design of efficient record matching queries. *VLDB*, 2007.

[4] Cohen, W.W., Ravikumar, P., Fienberg, S. E.: A Comparison of String Distance Metrics for Name-Matching Tasks. *IIWeb*, 2003.

[5] Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 19(1), 2007.

[6] Hooke, R., Jeeves, T.A.: Direct search solution of numerical and statistical problems. *Journal of ACM*, Volume 8, 1961.

[7] Köpcke, H., Rahm, E.: Training Selection for Tuning Entity Matching. *QDB/MUD*, 2008.

[8] Koudas, N., Sarawagi, S., Srivastava, D.: Record linkage: Similarity measures and algorithms. *SIGMOD*, 2006.

[9] Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., Euler, T.: Yale: Rapid Prototyping for Complex Data Mining Tasks. *SIGKDD*, 2006.

[10] Neiling, M., Jurk, S., Lenz, H.-J., Naumann, F.: Object identification quality. *DQCIS*, 2003.

[11] Rahm, E., Do, H.-H.: Data Cleaning: Problems and Current Approaches. *IEEE Data Engineering Bulletin, 23(4)*, 2000.

[12] Thor, A., Rahm, E.: MOMA - A Mapping-based Object Matching System. *CIDR*, 2007

[13] Weis, M., Naumann, F., Brosy, F.: A Duplicate Detection Benchmark for XML (and Relational) Data. *IQIS*, 2006.

[14] Xiao, C., Wang, W., Lin, X.: Ed-Join: An Efficient Algorithm for Similarity Join with Edit Distance Constraints. *VLDB*, 2008.