ABSTRACT

We propose a learning algorithm that utilizes an innovative set of features to re-rank a list of top-K matches and improves upon the ranking of the best match. We provide a bound on the size of an initial match list, tying the number of matches in a desired level of confidence for finding the best match. We also propose the use of schema matching predictors as features in the learning task, and tailored nine new predictors for this purpose. A large scale empirical evaluation with real-world benchmark schema sets show the effectiveness of the proposed algorithmic solution for re-ranking top-K schema matches.

1. INTRODUCTION

Schema matching is a data integration task that provides correspondences between concepts describing the meaning of data in various heterogeneous, distributed data sources (e.g., attributes in database schemata). Schema matching holds both theoretical and practical appeal and enjoys a continued interest by researchers and practitioners (see surveys [3] and books [2] for an overview).

Data integration in general, and schema matching in particular, were recognized in the literature to be uncertain processes [1,8,11,15]. A schema matcher may be required to consider many probable correspondences, and therefore, its choice may be possibly wrong. A primary reason for the uncertainty of the matching process is the enormous ambiguity and heterogeneity of data description concepts. Miller et al. [28], for example, attribute matching uncertainty to the syntactic representation of schemata, which can be semantically misleading. Therefore, in many cases an exact match will not be identified by a matcher as a best match.

For these reasons, the use of top-K schema matches, i.e., a ranked list of the “best” K schema matches a matcher can generate, was proposed [9,14,34]. Top-K schema matches can be used to create a search space in uncertain settings [4] and can serve in assigning probabilities in probabilistic schema matching [38].

Gal et al. provides a theoretical grounding to the use of top-K matches in the form of the monotonicity principle [16]. According to this principle, a monotonic schema matcher ranks matches according to a similarity measure whose behavior is closely related to a measure of choice (e.g., precision). Therefore, monotonicity ensures that the top-K matches contain a sufficiently close match to the exact match. Clearly, the size of K becomes a crucial element in our ability to identify the best match in a list.

In this work we tackle the challenge of identifying the best match within a top-K list and propose a novel re-ranking approach, applied as a post-processing step of top-K schema matching. We propose a learning algorithm that utilizes an innovative set of features to re-rank a list of top-K matches and improves upon the ranking of the best match. Our contribution is threefold:

- We provide a bound on the required size of K in a set of top-K matches, tying the number of matches with a desired level of confidence in finding the best match.
- Using schema matching predictors [39], we propose a new set of features to assist a learn-to-rank algorithm [5] to re-rank top-K matches so that the best matrix is ranked at the top.
- We performed large scale experiments with real-world benchmark schema sets, as well as synthetic data, to show the effectiveness of the proposed algorithmic solution for re-ranking top-K schema matches.

The rest of the paper is organized as follows. Section 2 presents the building blocks of our proposed model, namely similarity matrices, top-K matches, and predictors. A detailed discussion of the proposed learn-to-match algorithm (predictors as features, choice of algorithm, and the optimization criteria) is given in Section 3. Section 4 presents an extensive empirical analysis, showing that the proposed algorithm re-ranks better matches higher and out-performs other methods of re-ranking. We conclude with related work (Section 5) and concluding remarks (Section 6).

2. MODEL

In this section we provide the basic building blocks of the learn-to-match model. Section 2.1 introduces schema matching. Top-K matches are presented in Section 2.2, followed by an analysis of a bound on the value of K (Section 2.3). Finally, predictors are presented in Section 2.4.

2.1 Schema Matching

Let \((S, S')\) be a schema pair and let \({a_1, a_2, \ldots, a_n}\) and \({b_1, b_2, \ldots, b_m}\) be attributes of \(S\) and \(S'\), respectively. A
matching process employs one or more matching algorithms (or matchers for short), each making use of some schema characteristics, e.g., attribute labels, to deduce similarity. The output of such matchers can be conceptualized as a similarity matrix, formally defined for a matched pair \((S, S')\) as follows.

**Definition 1.** let \(S = S \times S'\) be the set of all possible correspondences between attributes of \(S\) and \(S'\). \(M(S, S')\) is an \(n \times m\) similarity matrix over \(S\).

\(M_{i,j}\) (typically a real number in \([0, 1]\)) represents a degree of similarity between \(a_i\) and \(b_j\). \(M(S, S')\) is a binary similarity matrix if for all \(1 \leq i \leq n\) and \(1 \leq j \leq m\), \(M_{i,j} \in \{0, 1\}\). A match between \(S\) and \(S'\) is a subset of \(M\)'s entries. For any matched pair \((S, S')\), \(\Sigma \subseteq 2^S\) is the set of all possible matches between attributes of the schema pair. It is worth noting that possible matches (those in \(\Sigma\)) are subject to application semantics using constrains such as 1:1 matching. We denote a match by \(\sigma \in \Sigma\).

**Example 1.** Table 1 provides an example of a similarity matrix over two purchase order schemata [25]. Schema \(S\) has four attributes, namely poDay and poTime, representing a timestamp of the order. poCode represents a purchase order number and city represents the city of shipment. The other schema, \(S'\), consists of three attributes, namely orderDate, orderNumber, and city for shipment date, as before.

<table>
<thead>
<tr>
<th>(S)</th>
<th>poDay</th>
<th>poTime</th>
<th>poCode</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>orderDate</td>
<td>0.22</td>
<td>0.11</td>
<td>0.11</td>
<td>0.11</td>
</tr>
<tr>
<td>orderNumber</td>
<td>0.00</td>
<td>0.09</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>city</td>
<td>0.20</td>
<td>0.17</td>
<td>0.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Let \(\mathcal{M}\) be the (infinite) set of all possible matrices over the schema pair \((S, S')\). \(f: \Sigma \times \mathcal{M} \rightarrow [0, 1]\) is a schema pair similarity function that assigns an overall value to a match \(\sigma\) given the individual similarity values of all matrix entries in \(\sigma\). Many of the similarity functions given in the literature of schema matching are monotone submodular functions, e.g.,

\[
\begin{align*}
\text{f}(\sigma, M) &= \sum_{(i,j) \in \sigma} M_{i,j} \\
\text{(1)}
\end{align*}
\]

Whenever the similarity matrix \(M\) is clear from the context, we shall refer to \(\text{f}(\sigma, M)\) simply as \(\text{f}(\sigma)\).

The similarity matrix of Example 1 is the outcome of Term [15], a string-based matcher. A Maximum Weight Bipartite Graph algorithm was applied to generate a match, \(\sigma = \{(\text{orderDate}, \text{poDay}), (\text{orderNumber}, \text{poTime}), (\text{city}, \text{city})\}\), marked by the boldface matrix entries, with \(\text{f}(\sigma) = 1.31\), using Eq. 1.

Let \(M^{\sigma}\) be a binary matrix, where \(M_{i,j} = 1\) whenever the correspondence \((a_i, b_j)\) is part of the exact match of the schema pair \((S, S')\) and \(M_{i,j} = 0\) otherwise. We define \(F_M^{\sigma}: \Sigma \rightarrow [0, 1]\) to be a relevance function, ranking matches according to their ability to identify correspondences in the exact match. Two common relevance functions in schema matching are precision and recall, defined as follows:

\[
\begin{align*}
P_{M^{\sigma}}(\sigma) &= \frac{|\sigma \cap M^{\sigma}|}{|\sigma|}, R_{M^{\sigma}}(\sigma) = \frac{|\sigma \cap M^{\sigma}|}{|M^{\sigma}|} \\
\text{(2)}
\end{align*}
\]

where \(M^{\sigma}\) represent the non-zero entries of \(M^{\sigma}\) and recalling that \(\sigma\) is a subset of \(M^{\sigma}\)'s entries. Using precision and recall, we further define the F1 measure \(F_{M^{\sigma}}(\sigma)\), calculated as the harmonic mean of \(P_{M^{\sigma}}(\sigma)\) and \(R_{M^{\sigma}}(\sigma)\). Whenever the exact match is clear from the context, we shall refer to \(F_{M^{\sigma}}(\sigma)\) simply as \(F(\sigma)\). Henceforth, we shall use \(F(\sigma)\) as our relevance function of choice.

Continuing with Example 1, let the exact match be \(\{(\text{orderDate}, \text{poDay}), (\text{orderDate}, \text{poTime}), (\text{orderNumber}, \text{poCode}), (\text{city}, \text{city})\}\). Then, we have that \(P(\sigma) = 0.67, R(\sigma) = 0.5\), and \(F(\sigma) = 0.59\).

**2.2 Top-K Schema Matches**

Top-K schema matches are the best \(K\) schema matches a matcher can generate, provided as a ranked list \(\sigma^K = (\sigma_1, \sigma_2, \ldots, \sigma_K)\). Top-K schema matches can be defined recursively as follows. For \(K = 1, \sigma_1\) satisfies

\[
\forall \sigma \in \Sigma, f(\sigma, M) \leq f(\sigma_1, M).
\]

Let \(\sigma_i\) denote the \(i\)-th best match, for any \(i > 1\). Then, given the best \(i - 1\) matches \(\sigma_1, \sigma_2, \ldots, \sigma_{i-1}\), the \(i\)-th best match \(\sigma_i\) is defined as a match that maximizes the schema pair similarity over matches that differ from each match in \(\sigma^{i-1} = (\sigma_1, \sigma_2, \ldots, \sigma_{i-1})\). Therefore, given top-K matches, any match \(\sigma \in \Sigma\) such that \(\sigma \notin \{\sigma_1, \sigma_2, \ldots, \sigma_K\}\) satisfies

\[
f(\sigma, M) \leq \min_{1 \leq j \leq K} f(\sigma_i, M) = f(\sigma, K) = f(\sigma, M).
\]

Continuing with Example 1, \(\sigma_2 = \{(\text{orderDate}, \text{poDay}), (\text{orderNumber}, \text{poCode}), (\text{city}, \text{city})\}\), with \(f(\sigma_2) = 1.31\) (the same as for best match computed above), \(P(\sigma_2) = 1.00, R(\sigma_2) = 0.75\), and \(F(\sigma_2) = 0.88\). It is worth noting that in this case, the second-best match is in fact better, in terms of \(P, R\), and \(F\), then the top match. A description of several algorithms for computing top-K matches can be found in [15](Ch. 5).

Each match \(\sigma_i\) \((1 \leq i \leq K)\) in the top-K schema match list \(\sigma^K = (\sigma_1, \sigma_2, \ldots, \sigma_K)\) is assigned with a relevance value, \(F(\sigma_i)\). We can transform relevance values into a ranked list so that the match with the highest \(F(\sigma_i)\) is assigned a value of 1 and so forth, aided by some tie-breaking mechanism. We term this ranking an ideal ranking of \(\sigma^K\) and denote it by \(\sigma^{(K)} = (\sigma^{(1)}, \sigma^{(2)}, \ldots, \sigma^{(K)})\). Then, for each \(\sigma_i\) in the list, we denote its rank in the ideal ranking by \(\text{Rank}(\sigma_i)\).

The two ranked lists, \(\sigma^K\) and \(\sigma^{(K)}\), can be jointly used to provide an overall quality measure of the ranking of the top-K match list. We introduce two such measures, namely Normalized Discounted Cumulative Gain (NDCG) [22] and Expected Reciprocal Rank (ERR [7]), both are shown to perform well in training machine learning models [5].

The NDCG (Normalized Discounted Cumulative Gain) is a non-binary evaluation measure that penalizes relevant items that appear low in a ranking. NDCG(\(\sigma^K\)) is given as the division of a discounted cumulative gain by the ideal discounted cumulative gain, as follows:

\[
\text{NDCG}(\sigma^K, \sigma^{(K)}) = \left( \frac{\sum_{i=1}^{k} 2^{F(\sigma^{(i)})} - 1}{\log_2(i + 1)} \right) \div \left( \frac{\sum_{i=1}^{k} 2^{F(\sigma^{(i)})} - 1}{\log_2(i + 1)} \right)
\]

(5)

The ERR (Expected Reciprocal Rank) assumes that the score of a match depends on the score of higher ranked matches (recall that \(\text{Rank}(\sigma_i)\) is computed with respect to \(\sigma^{(K)}\), the ideal ranking of \(\sigma^K\)).

\[
\text{ERR}(\sigma^K, \sigma^{(K)}) = \sum_{i=1}^{k} \frac{1}{\text{Rank}(\sigma_i)} \prod_{j=1}^{i-1} (1 - \text{Rank}(\sigma_j)).
\]

(6)
where

\[ \text{Rank}_i = \frac{2^{\text{Rank}(\sigma)} - 1}{2^k - 1} \]  \quad \text{(7)}

### 2.3 On Choosing an Optimal \( K \)

In this section we offer a theoretical bound on the size of the top-\( K \) list of matches that ensures, with high probability, an informative list.

There are two conflicting requirements when determining the size of a match list. On the one hand, a larger \( K \) provides a higher chance for the list to contain a good match, even for monotonic matchers, which are expected to assign high scores to good matches. On the other hand, a larger list may be challenging to a human judge when sifting through it; whereas for a machine learning algorithm it increases the risk of erroneous decision making and requires larger resources, in terms of both time and computation [24].

We shall denote by \( F^* = \max_{\sigma \in \Sigma} F(\sigma) \) the maximum value of the F1 over all possible matches. In addition, we let \( U(\sigma^K) = \max_{\sigma \in \Sigma} F(\sigma) \) denote the highest F1 in a given top-\( K \) match list.

Given the inherent uncertainty of the matching problem, the best top-\( K \) list for our purposes is a list that contains the best match possible, the one whose F1 is maximal. It is noteworthy that while we expect to find the exact match, the one whose F1 is maximal, it may not be present in \( \Sigma \), simply because it does not satisfy the constraints as set by the matching environment.

Given \( 0 < p < 1 \), a user-defined confidence level, we would like to find a \( K \) such that \( \Pr\{U(\sigma^K) = F^*\} \geq p \). That is, a top-\( K \) match list that is large enough to contain, with a probability \( p \), the match with the highest F1.

The monotonicity principle dictates a diminishing probability to find a match \( \sigma \), such that \( F(\sigma) = F^* \), low in a top-\( K \) match list. Therefore, for monotonic matchers, ranking such a match lower is still possible but with low probability.

In what follows, we define \( K \) to be a random variable that represents the size of \( \sigma^K \) that contains a match \( \sigma \) such that \( F(\sigma) = F^* \) while \( \sigma \not\in \sigma^{K-1} \) and assume \( K \) is geometrically distributed with some parameter \( \theta \). The \( \theta \) parameter is the probability of having the best match as the one with the highest F1, i.e., \( F(\sigma^*) = F^* \).

The use of a geometric distribution assumes there are infinite number of possible matches. While this assumption does not hold, the residual probability mass for high number of matches is sufficiently small for matchers to approximate a geometric behaviour quite closely, as shown empirically in Section 4.3.

**Theorem 1.** Let \( \sigma^K \) be a top-\( K \) match list such that \( K \sim \text{Geo}(\theta) \) is a random variable and exists \( \sigma \in \sigma^K \) for which \( F(\sigma) = F^* \). Then, for \( K \sim \left[ \frac{\ln(1-p)}{\ln(1-\theta)} \right] \), such that \( 0 < p < 1 \) is a user-defined parameter, \( \Pr\{U(\sigma^K) = F^*\} \geq p \).

**Proof.** Let \( k_p \geq 1 \) be the minimal value of \( K \) at which the probability that the top-\( K \) match list contains \( F^* \) is \( p \). \( k_p \) probabilistically guarantees (with a probability of \( p \)) that the match list contains a match \( \sigma \) such that \( F(\sigma) = F^* \).

Recall that \( \Pr\{K = k_p\} \) is the probability that the top-\( K \) match list contains \( F^* \) while \( F^* \) was not present in \( \sigma^K \), \( \forall k : 1 \leq k < K \). We are, in fact, interested in the probability that \( K \) is less than (or exactly) \( k_p \), i.e., we have seen \( \sigma \) such that \( F(\sigma) = F^* \) in top-\( K \) match list size smaller than (or equals) \( k_p \):

\[ \Pr\{K \leq k_p\} = p \]  \quad \text{(8)}

From the definition of \( K \) and Eq. 8,

\[ \Pr\{U(\sigma^{k_p}) = F^*\} = \Pr\{K \leq k_p\} = p \]  \quad \text{(9)}

\( K \) is geometrically distributed, and therefore

\[ p = \Pr\{K \leq k_p\} = 1 - \Pr\{K > k_p\} = 1 - \Pr\{K \geq k_p + 1\} = 1 - \sum_{k=k_p+1}^{\infty} \theta(1-\theta)^{k-1} \]  \quad \text{(10)}

By sum of a geometric series one has that

\[ 1 - \theta(1-\theta)^{k_p} \frac{1}{1-(1-\theta)} = p \]  \quad \text{(11)}

The left-hand side of Eq. 11 can be rewritten as follows:

\[ 1 - \frac{\theta(1-\theta)^{k_p}}{1-(1-\theta)} = 1 - \frac{\theta(1-\theta)^{k_p}}{\theta} = 1 - (1-\theta)^{k_p} \]  \quad \text{(12)}

and the following can be derived:

\[ \ln((1-\theta)^{k_p}) = \ln(1-p) \Rightarrow k_p\ln(1-\theta) = \ln(1-p) \Rightarrow \]

\[ k_p = \frac{\ln(1-p)}{\ln(1-\theta)} \]

The size of the top-match list is an integer, and therefore we define \( K \) to be:

\[ K = \lceil k_p \rceil = \left[ \frac{\ln(1-p)}{\ln(1-\theta)} \right] \]  \quad \text{(14)}

Combining Eqs. 9 and 14 one has that

\[ \Pr\{U(\sigma^K) = F^*\} = \Pr\{K \leq K\} = \Pr\left\{ K \leq \left[ \frac{\ln(1-p)}{\ln(1-\theta)} \right] \right\} \]

\[ \geq \Pr\left\{ K \leq \frac{\ln(1-p)}{\ln(1-\theta)} \right\} = \Pr\{K \leq k_p\} = p \]

which concludes the proof. \( \square \)

Theorem 1 provides a theoretical bound on the size of a top-\( K \) match list. For example, given \( K \sim \text{Geo}(0.38) \), to ensure a top-\( K \) match list that consists, with probability of at least 0.99, the best match one needs to obtain a match list of size:

\[ K = \left[ \frac{\ln(1-0.99)}{\ln(1-0.38)} \right] \approx [9.63] = 10 \]

### 2.4 Schema Matching Predictors

Schema matching predictors assess the quality of a match in the absence of an exact match [39]. Formally, a predictor is a function that maps a similarity matrix into a real prediction value.

Predictors may predict different qualities, such as precision or recall. In general, there are two types of predictors, *idealizers* and *idealizers* [39]. Idealizers are predictors that assume there is some principle that dictates high quality results and the prediction is based on the distance between the supplied result and an ideal result adhering to this
principle. For example, the Binary Matrix Converter (BMC) predictor [39] is based on the assumption that non-binary similarity matrices are binary matrices masked by random noise. With that said, this predictor prefers matrices with entries close to 0 or 1.

Internalizers are based on internal properties of the similarity matrix. For example, Max, STDEV, and Avg calculate the respective measure they are named after for each row in the matrix and average the values over the number of the rows in the matrix [39].

3. LEARNING TO MATCH

Given a top-$K$ match list that contains, with high probability, a close-to-exact match, we are now left with the problem of identifying which of the matches in the list is the one with the highest F1 value. In this work we advocate an algorithmic solution for re-ranking a top-$K$ match list, aiming at positioning at the top the match with the highest F1. We propose a learn-to-match algorithm, which utilizes novel predictors as its feature set for learning.

As a starting point, we use a body of research in the Information Retrieval community, called Learn-to-Rank (LTR) [23]. LTR algorithms learn to optimally combine features, extracted from query-document pairs through discriminative training, to re-rank a retrieved set of documents.

In this work we present an innovative use of feature vectors to represent successful matches in an LTR algorithm. Therefore, the right selection of features is the most important element in our proposed learning-to-match framework. Section 3.1 presents a new set of features using predictors (see Section 2.4 for predictor definition).

Any LTR algorithm is a framework with three basic elements, the choice of which is critical, namely features, a method, and an optimization function. In Section 3.2 we detail the last two elements in the context of schema matching and present a complete learn-to-match algorithm.

3.1 Predictors as Features

To construct feature vectors for the learn-to-match task we propose to use predictors over similarity matrices. Matrix-based predictors predict the usefulness of an entire similarity matrix for a matching task. In this work we propose three types of predictors, namely PCA (Section 3.1.1), entropy (Section 3.1.2) and matrix norm predictors (Section 3.1.3).

Most of the proposed predictors manipulate similarity matrices. Recall that a match is defined as a subset of a similarity matrix entries. We offer a simple transformation of a match into a similarity matrix, as follows. Let $M(S, S')$ be a similarity matrix over a schema pair $(S, S')$ and let $\sigma \in \Sigma$ be a match between $S$ and $S'$. A non-binary transformation of $\sigma$ is a matrix $M'$ such that

$$M'_{i,j} = \begin{cases} M_{i,j} & \text{if } M_{i,j} \in \sigma \\ 0 & \text{otherwise.} \end{cases}$$ (15)

For example, $M'$ is a non-binary transformation of the best match (Example 1):

$$M' = \begin{pmatrix} 0.22 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.09 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{pmatrix}$$

3.1.1 PCA Predictors

Principal Component Analysis (PCA) is a dimension reduction method, aiming at preserving as much variance as possible [32]. PCA is well-suited for schema matching, where matchers tend to focus on a specific semantic angle of the matching problem (e.g., attribute labels) while ignoring other aspects. Constructing a PCA model over such data provides a compact representation that ignores missing semantic angles while still maintaining the main variability of the match. Such representation is considered a good property of a match, allowing the differentiation of possibly correct correspondences from background noise [17].

PCA typically uses eigen-decomposition of a matrix, which is limited to square matrices. When dealing with non-square similarity matrices we propose to perform the PCA via singular value decomposition (SVD) [44].

We propose three new predictors, namely First Principal Component predictor (1PC), Second Principal Component predictor (2PC), and Principal Component Analysis predictor (PCA). The first two simply use the first and the second principal components as predictors. The third is a sum of informative components, to be described next.

Informative (“meaningful”) components are chosen as the leading principal components that balance the dimension reduction with keeping as much variance as possible. We use Horn’s Parallel Analysis [20], a Monte Carlo simulation process, to generate a PCA predictor. Formally, the only singular values that are considered significant and are associated with informative components are the ones larger than the mean value of the expected singular values. Such expected singular values are obtained by simulating normal random samples that capture the observed data in terms of sample size and number of variables. Therefore, the PCA predictor is given as follows:

$$\sum_{i=1}^{l} \mathbb{I}_{(\mu_i > \overline{\mu}_e)} \mu_i,$$ (16)

where $l$ is the number of singular values, $\mathbb{I}_{(1)}$ is an indicator function, $\mu_i$ is the $i$-th singular value, and $\overline{\mu}_e$ are the mean values of expected singular values of the random data.

As an example of computing the three predictors consider $M'$, the non-binary transformation of the best match (Eq. 15) of Example 1. Using SVD, the following singular values are obtained: $\mu_1 = 1.00, \mu_2 = 0.22, \mu_3 = 0.09$, where the first two values are the first and second principal components. The principal components are calculated using the orthogonal matrices, which is a result of the SVD.

For the third predictor PCA, let $\overline{\mu}_e = 0.28, \overline{\mu}_e = 0.21, \overline{\mu}_e = 0.13$ be a simulated normal random samples given $M'$. The predictor value would be $PCA = 1 \cdot 1.00 + 1 \cdot 0.22 + 0 \cdot 0.09 = 1.22$. For this example we have received a relatively low values of the predictors (1.00, 0.22, and 1.22) which indicates that $M'$ represents a good match. For comparison, recall the similarity matrix for Example 1, before the transformation. This matrix would produce the following predictors values (1.05, 0.28, and 1.31) indicating, by comparison, that $M'$ is better.

3.1.2 Entropy Predictors

Match diversity was shown to be useful in regulating a tradeoff between precision and recall [17]. We now propose two new diversity measures based on entropy, an information
theory measure that estimates data diversity according to the expected information contained in it [42].

The Mean Pure Attribute-wise Entropy predictor (MPE) averages entropy over all rows, each stands for a specific attribute:

$$\frac{1}{n} \sum_{i=1}^{n} \left( \sum_{j=1}^{m} -a_{ij} \ln a_{ij} \right)$$ (17)

The Von Neumann Entropy predictor (VNE) (Eq. 18) is borrowed from the field of quantum statistical mechanics [31], used to quantify the diversity of a system, described by a density matrix. A “match system” in a pure state has zero entropy, whereas large entropy indicates a more uncertain system, and hence, more chance for lower quality. VNE is computed to be the entropy of the singular values of the similarity matrix (recall that we use singular value decomposition instead of the common eigen-decomposition).

$$- \sum_{i=1}^{I} \mu_i \ln \mu_i$$ (18)

For $M'$ (Example 1), one has that $\text{VNE} \approx 0.55$, a relatively low value, indicating a good match.

### 3.1.3 Matrix Norm Predictors

A matrix norm $\|M\|$ of a matrix $M$, the equivalent of vector norm in vector space [21], satisfies the following conditions:

- $\|M\| \geq 0$.
- $\|M\| = 0$ iff $M = 0$.
- $\|M\| = \|\beta M\|$ for each $\beta \in \mathbb{R}$ and $M \in \mathbb{R}^{n \times n}$.
- $\|M + M'\| \leq \|M\| + \|M'\|$ for each $M, M' \in \mathbb{R}^{n \times n}$.

Matrix norms can be used to quantify errors in a matrix [6], assigning small norm value with a better matrix quality. Such errors can be attributed, in the case of similarity matrices, to the uncertainty involved in a matching process. To understand better this phenomenon, recall that the similarity matrix of Example 1 contains many non-zero entries. Each such entry that does not relate to a correspondence in the exact match, adds to the cumulative error of the matrix. Also, three of the four correspondences in the exact match are assigned with low similarity scores (less than 0.25), which again adds to the error introduced by the similarity matrix.

In this work we introduce four matrix norm predictors, as follows:

- **Norm1**, the maximum absolute column sum norm:
  $$\|M\|_1 = \max_j \sum_{i=1}^{n} |a_{ij}|$$ (19)

- **Norm2**, the square root of the maximum eigenvalue of the square matrix $M^HM$, where $M^H$ is the conjugate transpose. In our case, when there are no complex numbers, $M^H$ is equivalent to the original matrix transpose:
  $$\|M\|_2 = \sqrt{\mu_{\text{max}}(M^HM)}$$ (20)

- **NormInf**, a maximum absolute row sum norm:
  $$\|M\|_\infty = \max_i \sum_{j=1}^{m} |a_{ij}|$$ (21)

- **NormF**, the square root of the sum of the absolute squares of all the elements in the matrix:
  $$\|M\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} |a_{ij}|^2} = \sqrt{\text{trace}(M^HM)}$$ (22)

### 3.2 Learn-to-Match Algorithm

Having described the set of features, we are now ready to describe the proposed learn-to-match (LTM for short) algorithm (see Figure 1 for illustration). The algorithm has two phases, namely training and testing. The input to the training phase is a set of top-$K$ ideal lists $\sigma^{(K)}_i$ and its output is a learned model that can be used to re-rank a new top-$K$ list at the testing phase. Each input list $\sigma^{(K)}_i$ is first transformed into a set of feature vectors $FV(\sigma_i)$, one per each match $\sigma_i \in \sigma^{(K)}_i$ (upper left corner of Figure 1). Each feature vector contains the values of the various predictors that were calculated over a single match. The algorithm then learns a model that, given a top-$K$ match list obtained for a newly (unseen) schema pair similarity matrix, whose individual matches are represented by the same feature vectors (bottom left), is used for re-ranking that list (bottom right). We next describe in more details the algorithm's training (Section 3.2.1) and testing (Section 3.2.2) phases.

![Figure 1: Learn-to-Match Algorithm illustrated](image-url)
the non-binary transformation, and therefore, any non-zero value in it is considered an attribute correspondence, regardless of how low the similarity value may be. Such a matrix, with its expected high recall (and low precision) serves as a regulator by allowing the algorithm to explore matrices entries that the rest of the list avoids, allowing the algorithm to generalize better.

To learn the optimal \( F(\sigma) \), we use LambdaMart LTR algorithm, which combines LambdaRank and MART (Multiple Additive Regression Trees) and was proven to be successful for solving real-world ranking problems [5]. LambdaRank defines gradients with respect to each object (in our case \( \sigma \)) by using its feature vector representation \( (FV(\sigma)) \) and evaluation measures \( (\text{NDCG or ERR}) \) directly. Recall that since both \( \text{NDCG} \) and \( \text{ERR} \) are measures over the whole top-\( K \) list one has to define derivatives for these measures with respect to each object. We start by defining gradients for each pair of objects, \( \sigma_i \) and \( \sigma_j \) \((\lambda^{i,j})\) as a function of the difference in model scores between \( \sigma_i \) and \( \sigma_j \) and the difference in the selected evaluation measure by swapping \( \sigma_i \) and \( \sigma_j \) in the ranking. Consequently, we define the gradient for each \( \sigma_i \) \((\lambda^i)\) as a sum of gradient pairs \( \lambda^{i,j} \) \((1 \leq j \leq K)\) as follows:

\[
\lambda^i = \sum_{\sigma_j \in \sigma \setminus \{\sigma_i\}} \lambda^{i,j}.
\]

LambdaMart then uses \( T \geq 1 \) gradient boosted decision trees using a cost function derived from LambdaRank for solving the ranking task. Each tree calculates its own gradient, \( \lambda \), with respect to the gradients of the objects, which can be used for optimization (e.g., via gradient decent). The overall scoring function is derived as follows:

\[
\hat{F}(\sigma) = \sum_{t=1}^{T} \omega_t \psi_t(FV(\sigma)),
\]

where \( \psi_t(FV(\sigma)) \) and \( \omega_t \) denote a given single tree’s score and its relative weight at iteration \( t \), respectively. At each iteration, each tree creates the corresponding \( \lambda^i \)'s and uses a gradient decent step in order to improve its score until convergence.

### 3.2.2 Learn-to-Match Algorithm: Testing Phase

Using the learned model \( \hat{F}(\sigma) \), we can re-rank a top-\( K \) match list that was generated for some matched schema pair.

Algorithm 1 contains the pseudocode of the testing phase of the algorithm. Using a given matcher, the algorithm creates a similarity matrix (line 3) and generates a top-\( K \) list, where \( K \) is determined by Theorem 1 and given as input to the algorithm (line 4). \( f_{max}(\sigma) \) and \( f_{min}(\sigma) \) are computed (lines 5-6) to be used in normalizing similarity values with the predicted \( F1 \) (line 16). A feature vector is created for each of the matches in the list, using the non-binary transformation method, as was defined in Eq. 15 (lines 7-13). Each predictor \( \text{pred}_j \) is computed and added to the feature vector \( FV. \) \( \hat{F} \) is computed for each match using Eq. 23 (line 12). Re-ranking is performed in a way that minimizes the chance of model concept-drift [46] by interpolating the model score \( \hat{F}(\sigma) \) of each match \( \sigma \in \sigma^K \) with its similarity score \( f(\sigma, M) \) (line 16) using the normalizing values that were computed in lines 5, 6, 14, and 15. Finally, the re-ranked list \( \hat{\sigma}^{(K)} \) is returned (line 18).

### 4. EMPIRICAL EVALUATION

We now describe the evaluation of the proposed learn-to-match algorithm. We start with a description of datasets (Section 4.1) and experimental setup (Section 4.2). We next present empirical evidence for Theorem 1, which provides a bound on the value of \( K \) (Section 4.3). We then evaluate the impact of our newly proposed learning features (predictors) on the algorithm performance (Section 4.4). We conclude this section with a performance analysis of the learn-to-match algorithm, comparing it against state-of-the-art schema matching algorithms, and demonstrating favorable performance results (Section 4.5).

#### 4.1 Datasets

Table 1 details the datasets used in the experiments. For each dataset, the number of schemata it contains, its size range (in terms of attributes), and the number of schema pairs is reported. The exact match of the Web-forms and synthetic datasets is \( 1:n \) while for the others it is \( 1:1 \).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Schemata</th>
<th>#Attr</th>
<th>#Pairs</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web-forms</td>
<td>247</td>
<td>10-30</td>
<td>147</td>
<td>1:1</td>
</tr>
<tr>
<td>Purchase Order</td>
<td>10</td>
<td>50-400</td>
<td>44</td>
<td>1:n</td>
</tr>
<tr>
<td>University Applications</td>
<td>16</td>
<td>50-150</td>
<td>182</td>
<td>1:n</td>
</tr>
<tr>
<td>Beta</td>
<td>247</td>
<td>10-30</td>
<td>147</td>
<td>1:1</td>
</tr>
</tbody>
</table>

Table 1: Datasets used in the evaluation

The Web-forms [16] dataset contains schemata that were automatically extracted from Web forms using the OntoBuilder extractor. Exact matches were manually constructed by human judges. The Purchase Order dataset [25] contains XML documents describing purchase orders extracted from various systems. The University Applications dataset [39]...

---

1All datasets are available for download at [https://bitbucket.org/tosmers77/ontobuilder-research-environment/downloads/dataset.zip](https://bitbucket.org/tosmers77/ontobuilder-research-environment/downloads/dataset.zip)
contains university application forms from various US universities, collected as part of the NisB project and transferred into XML Schema Definition (XSD) format.

The three benchmark datasets were designed to present schema pairs with differing levels of difficulties, introducing alongside easy matches also complex relationships and attributes, which may result in fairly low levels of precision and recall, even using the strongest of matchers.

We also created a synthetic dataset, following a model of Marie et al. [27] in which we generated similarity matrices using two beta distributions, one for correspondences in the exact match and one for non-corresponding attributes. The two distributions differ in their parameters, where attribute correspondences are assigned values that are closer to 1 and the others are closer to 0. The synthetic dataset provides us with the opportunity to test the proposed features and the LTM algorithm independently of the quality of a matcher that creates similarity matrices.

We have implemented a method which takes 147 exact match matrices from the Web-Forms dataset and manipulated them as follows. Each of the attribute pair correspondences in the exact match was assigned a value that is beta distributed for corresponding attributes. Out of the remaining values, we chose randomly entries and assigned them with values that were randomly generated using a beta distribution for non-corresponding attributes. We chose the number of non-corresponding attribute pairs to be double the size of the exact match. All remaining entries are assigned with a value of 0.

Web-Forms, University Applications, and Beta datasets contain a sufficiently large number of schema pairs for the training phase of LTM. The Purchase Order dataset contains a small number of schema pairs, and therefore we use bootstrapping, a re-sampling method based on a small sample data.

4.2 Experimental Setup

Evaluations were performed using a Dell Inc. PowerEdge R720 server with a 20 true cores (40 virtual cores) Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz CPU, 128GB RAM (8x16GB), and a CentOS 6.4 operating system with x86_64 Kernel:2.6.32-358.6.2.el6.x86_64. The various matchers, predictors, and the LTM algorithm were implemented in Java(TM) SE Runtime Environment (build 1.8.0_45-b14).

Three schema matching tools were used in our experiments, namely ORE, COMA, and AMC. OntoBuilder Research Environment (ORE) is a research prototype, geared towards large scale matching experiments using various matchers on a variety of datasets, evaluating their performance using multiple quality measures. The Auto-Mapping Core (AMC) [33], developed by SAP Research, provides an infrastructure and a set of algorithms to establish correspondences between business schemata. We use AMC’s Token Path algorithm, which is embedded in ORE. We also use two matching algorithms from COMA 3.0, a state-of-the-art schema matching research tool, namely Threshold(ν) and Max-Delta(δ), whose details are given shortly.

<table>
<thead>
<tr>
<th>Matcher</th>
<th>System</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term (Syntactic)</td>
<td>OntoBuilder [30]</td>
<td>1LM</td>
</tr>
<tr>
<td>Token Path (Syntactic)</td>
<td>AMC [33]</td>
<td>1LM</td>
</tr>
<tr>
<td>WordNet (Semantic) [19,35,37]</td>
<td>ORE</td>
<td>1LM</td>
</tr>
<tr>
<td>Maximum weighted bipartite graph match (MWBG)</td>
<td>OntoBuilder</td>
<td>2LM</td>
</tr>
<tr>
<td>Stable marriage (SM)</td>
<td>OntoBuilder</td>
<td>2LM</td>
</tr>
<tr>
<td>Threshold</td>
<td>Coma</td>
<td>2LM</td>
</tr>
<tr>
<td>Max-Delta</td>
<td>Coma</td>
<td>2LM</td>
</tr>
</tbody>
</table>

Table 2: List of 1LMs and 2LMs used in the evaluation

In the experiments we use three matchers to generate similarity matrices (termed 1LM – First Line Matchers) and four matchers that create matches out of similarity matrices (termed 2LM – Second Line Matchers), see Table 2. The 1LMs are Term, WordNet, and Token Path. Term [15] compares attribute names to identify syntactically similar attributes (e.g., using edit distance and soundex). WordNet uses abbreviation expansion and tokenization methods to generate a set of related words for matching attribute names. Finally, Token Path [33] integrates node-wise similarity with structural information by comparing the syntactic similarity of full paths from root to a node.

Maximum weighted bipartite graph match (MWBG) [18] and stable marriage (SM) [27] use well-known algorithms for solving the problem of matching two sets of elements given an ordering of preferences for each element. Threshold(ν) and Max-Delta(δ) are selection rules, prevalent in many matching systems [10]. Threshold(ν) selects those entries (i,j) having \( M_{i,j} \geq \nu \). Max-Delta(δ) selects those entries that satisfy: \( M_{i,j} + \delta \geq \max_{j'} \), where \( max_{j'} \) denotes the maximum match value in either row i or column j.

For each (dataset, 1LM) pair, the K parameter was set according to Table 3 and a top-K list \( \sigma^K \) was created using the TopK algorithm [15] (implemented in ORE), employed with the similarity function of Eq. 1.

For learning the LambdaMart model we used the RankLib tool, which is commonly used in the Information Retrieval community for LTR tasks. We used the SVM \( ^{light} \) format to represent feature vectors. We used LambdaMart’s recommended parameter configuration [45] with either NDCG or ERR as its (optimization) target. Following previous recommendations [45], we selected among the two using NDCG as a selection criterion. The selected model was then evaluated over the test data. The smoothing parameter \( \alpha \) (see Algorithm 1) was chosen (during the training phase) to be in \( \alpha \in \{0.1, 0.2, \ldots, 1\} \).

4.3 Tuning parameter K

We start by providing an empirical validation to the Geometric distribution assumption of Theorem 1. To this end, for each schema pair in a given dataset we first obtain its similarity matrix. Using the similarity function of Eq. 1, we generate a sample of top-K matches using the TopK algorithm. Let N be the number of schema pairs in a given dataset (e.g., \( N = 147 \) for the Web-forms dataset). For each \( j \in [1, 2, \ldots, k] \) we have \( N \) top-j lists (so that for each schema pair, \( \sigma' \) is subsumed in \( \sigma^{(1)} \). \( O_j = \# \{ U(\sigma') = F^j \} \) captures the observed number of top-j matches \( \sigma' \) in the

2http://www.nisb-project.eu/
3https://bitbucket.org/romers77/ontobuilder-research-environment/wiki/Home
4http://sourceforge.net/p/coma-ce/mysvn/HEAD/tree/coma-project
5https://sourceforge.net/p/lemur/wiki/RankLib/
6https://sourceforge.net/p/lemur/wiki/RankLib/
pool of matched schema pairs, which obtained $F^*$. Using the moments parameter estimation method, for a given (dataset, 1LM) pair, we estimate the Geometric distribution parameter as follows: $\hat{\theta} = \frac{\bar{O}}{\bar{E}}$.

Next, we validate that $K \sim \text{Geo}(\hat{\theta})$ using the Chi-Square goodness of fit test [41]. For each $j \in \{1, 2, \ldots, k\}$, we calculate $E_j = \frac{N \cdot (1 - (1 - \hat{\theta})^j)}{N}$. The expected number of top-$j$ matches $\sigma^j$ in the pool of matched schema pairs which should have obtained $F^*$, under the assumption that $K \sim \text{Geo}(\hat{\theta})$ (Eq. 12). We next calculate the test statistics $\chi^2 = \sum_{j=1}^{k} \frac{(O_j - E_j)^2}{E_j}$ with $k - 1$ degrees of freedom [41].

<table>
<thead>
<tr>
<th>Setting</th>
<th>$\hat{\theta}$</th>
<th>p-value</th>
<th>$K(p)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Web Forms</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Term</td>
<td>0.43</td>
<td>0.001</td>
<td>9 (0.99)</td>
</tr>
<tr>
<td>WordNet</td>
<td>0.40</td>
<td>0.038</td>
<td>10 (0.99)</td>
</tr>
<tr>
<td>Token Path</td>
<td>0.39</td>
<td>0.017</td>
<td>10 (0.99)</td>
</tr>
<tr>
<td><strong>Purchase Order</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Term</td>
<td>0.22</td>
<td>0.041</td>
<td>12 (0.95)</td>
</tr>
<tr>
<td>WordNet</td>
<td>0.21</td>
<td>0.015</td>
<td>12 (0.95)</td>
</tr>
<tr>
<td>Token Path</td>
<td>0.31</td>
<td>0.004</td>
<td>10 (0.95)</td>
</tr>
<tr>
<td><strong>University</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applications</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Term</td>
<td>0.20</td>
<td>0.020</td>
<td>12 (0.95)</td>
</tr>
<tr>
<td>WordNet</td>
<td>0.17</td>
<td>0.001</td>
<td>13 (0.95)</td>
</tr>
<tr>
<td>Token Path</td>
<td>0.25</td>
<td>0.031</td>
<td>11 (0.95)</td>
</tr>
<tr>
<td>Beta</td>
<td>0.37</td>
<td>0.045</td>
<td>10 (0.99)</td>
</tr>
</tbody>
</table>

Table 3: Report of the estimated Geometric parameter $\hat{\theta}$, p-value and tuned $K$ for each (dataset, 1LM) pair.

Using $k = 30$, Table 3 reports for each (dataset, 1LM) pair and the Beta dataset, the values of the estimated parameter $\hat{\theta}$ and the p-value (significance) of the test. For all pairs, $K$ is Geometrically distributed and the results are all statistically significant (in at least 95% confidence).

With this empirical validation of Theorem 1, we can tune the optimal $K$ for each (dataset, 1LM) pair, as reported in Table 3. the confidence level parameter, $p$ is given in parentheses.

![Pair example of $U(\sigma^k)$ as a function of $K$.](image)

**Figure 2:** Left: an example of $U(\sigma^k)$ as a function of $K$. Right: curve illustrates expected values by the geometric distribution ($E_j$) and bars show the observed values ($O_j$). A vertical line is drawn in the calculated $K$ value (10).

We conclude this subsection with an illustration (see Figure 2) of the behaviour of $U(\sigma^k)$ as well as the Chi-Square test that we have conducted. Figure 2(left) presents an example pair from the Web-Forms dataset, matched by the Term matcher, demonstrating the diminishing growth of $U(\sigma^k)$. Figure 2(right) demonstrates, using the University Applications dataset matched with Term, the resemblance between the expected values by the geometric distribution (red curve) and the observed values (blue bars).

### 4.4 Predictor Evaluation

We next evaluate the prediction quality of the predictors proposed in Section 3.1. We assess their relative prediction quality by comparing them to previously suggested state-of-the-art predictors. We wish to validate that the proposed predictors provide competitive (or even better) representation of the similarity matrix. We therefore show that, using our proposed predictors as additional learning features for LTM could improve the overall re-ranking performance.

Following previous works [17,39], we measure prediction quality using Pearson’s $r$ correlation between predicted values and the actual match quality values in terms of $F_1$.

We assessed prediction quality over 3300 matrices generated by running all three 1LMs on 275 different schema pairs from the three datasets. Using the similarity matrix produced by each 1LM, we run the four 2LMs, namely MBWRG, SM, Max-Delta ($\delta$), and Threshold($\nu$). Following [17], the parameters of the last two 2LMs were set to $\nu = 0.5$ and $\delta = 0.1$, respectively. Prediction was then applied on the matrix outcome of each 2LM.

<table>
<thead>
<tr>
<th>Predictor</th>
<th>P</th>
<th>R</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMM</td>
<td>.1600*</td>
<td>.2464*</td>
<td>.1939*</td>
</tr>
<tr>
<td>BMC</td>
<td>.1681*</td>
<td>.1413*</td>
<td>.1611*</td>
</tr>
<tr>
<td>LMM</td>
<td>-.1065*</td>
<td>.2802*</td>
<td>-.0243</td>
</tr>
<tr>
<td>AvgMax</td>
<td>.0821</td>
<td>.3784*</td>
<td>.1272*</td>
</tr>
<tr>
<td>AvgSTDEV</td>
<td>.1309*</td>
<td>.5848*</td>
<td>.2070*</td>
</tr>
<tr>
<td>AvgDominants</td>
<td>.3694*</td>
<td>.3343*</td>
<td>.3897*</td>
</tr>
<tr>
<td>MCD</td>
<td>.3000*</td>
<td>.1120*</td>
<td>.3178*</td>
</tr>
</tbody>
</table>

Table 4: Pearson’s $r$ correlation to $F_1$ (P), Recall (R) and Precision (P) of the state-of-the-art matrix predictors

<table>
<thead>
<tr>
<th>Predictor</th>
<th>P</th>
<th>R</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPC</td>
<td>-2.979*</td>
<td>.2324*</td>
<td>-2.816*</td>
</tr>
<tr>
<td>2FC</td>
<td>-2.856*</td>
<td>.2136*</td>
<td>-2.696*</td>
</tr>
<tr>
<td>PCA</td>
<td>-.3972*</td>
<td>.1195*</td>
<td>-0.4156*</td>
</tr>
<tr>
<td>VNE</td>
<td>-.1805*</td>
<td>.3011*</td>
<td>-1.773*</td>
</tr>
<tr>
<td>MPE</td>
<td>-.2234*</td>
<td>.1767*</td>
<td>-2.426*</td>
</tr>
<tr>
<td>Norm1</td>
<td>-.2837*</td>
<td>.1837*</td>
<td>-2.836*</td>
</tr>
<tr>
<td>Norm2</td>
<td>-.3216*</td>
<td>.3747*</td>
<td>-3.324*</td>
</tr>
<tr>
<td>NormInf</td>
<td>-.2709*</td>
<td>.1370*</td>
<td>-2.822*</td>
</tr>
<tr>
<td>NormF</td>
<td>-.3757*</td>
<td>.1727*</td>
<td>-3.799*</td>
</tr>
</tbody>
</table>

Table 5: Pearson’s $r$ correlation to $F_1$ (P), Recall (R) and Precision (P) of the new matrix predictors

We compared the various predictors that were proposed in Section 3.1 with state-of-the-art baseline predictors as follows. As a first set of baselines, we used the BMC, Max, STDEV, Avg predictors, whose details were shortly described in Section 2.4. Additionally, we compared against the BMM [39] and LMM [39], Dominants [26], and MCD [17] predictors. Both BMM and LMM predictors are obtained by first “flattening” the similarity matrix $M$ into a vector with $n \cdot m$ entries; each vector entry uniquely corresponds to one entry value of matrix $M$. Then, BMM and LMM both measure the cosine similarity between that vector and an “ideal” (similarity) vector that is constructed from it. LMM constructs an ideal vector that has a single 1-valued entry per matrix row, while BMM define it by the “closest” binary vector [39]. The Dominants predictor counts the number of
matrix entries which are the largest in their respective row and column, divided by the number of matrix rows. Finally, the MCD (Match Competitor Deviation) predictor [17] measures the average difference between each matrix entry \( M_{ij} \)'s value and the mean entry value \( \bar{M}_{ij} \) calculated over all matrix row \( i \) and column \( j \) entries' values.

Table 4 reports on the Pearson's \( r \) correlation between the baseline predictors and the three match quality measures, F1 (F), Recall (R) and Precision (P). A two-tailed t-test of all matrix row \( i,j \) entries' values.

\[ \sigma^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})^2 \]

Table 5 reports the correlation values of the newly proposed predictors. We observe that all the correlations between the new predictors and quality measures are statistically significant. Furthermore, the results confirm our expectations; all these predictors are characterized by inverse expectations; all these predictors are characterized by inverse

\[ \text{Match Competitor Deviation} = \frac{\sum_{i,j} (M_{ij} - \bar{M}_{ij})^2}{\sum_{i,j} (\bar{M}_{ij})^2} \]

4.5 Learning to Match Model Evaluation

Finally, we evaluate the benefits of applying the LTM model for re-ranking top-\( K \) match lists. To this end, we utilize both previously suggested state-of-the-art predictors and the proposed predictors for representing match matrices as feature vectors. We show the effectiveness of LTM in re-ranking against the original top-\( K \) ranking, the ideal ranking and a learned model without the proposed new predictors.

The evaluation of LTM was performed using \( f \)-fold cross validation (with \( f \in \{2, 3, 5\} \)). For that, we first (randomly) split each dataset into \( f \) folds. We then repeat the experiment \( f \) times, using \( f-1 \) folds for training and the remaining fold for testing. We report on the average performance (i.e., Precision, Recall and F1) over all folds.

4.5.1 LTM effectiveness

We first evaluate LTM's effectiveness by training three LTM models. The first two models, \( LTM_{Base} \) and \( LTM_{New} \), were trained using only the previously suggested baseline predictors (Table 4) and only the newly proposed predictors (Table 5), respectively. The third, \( LTM_{Full} \), was trained using the full predictor (feature) set. We compared the three LTM variants against three different baselines: 1) The original best match (\( \sigma_1 \)); 2) Cross Entropy Matcher [17] (CEM), a state-of-the-art 2LM that utilizes match diversity as its additional target and was previously shown to give the overall best schema matching performance; 3) ideal ranking (defined in Section 2.2).

LTM comparative evaluation results are reported in Table 6 using Precision (P), Recall (R) and F1 (F) to compare the top of the relevant ranked list and the exact match (\( \sigma_{1} \)). Statistical significant differences in performance are marked with an asterisk (*) denote significant results (\( p\text{-value} < 0.01 \)).
Table 7: Comparing NDCG and ERR (See Eqs. 5 and 6, respectively) over $\sigma^K$ (Orig), $\hat{\sigma}^K$ by full LTM model ($LM_{Full}$), $\hat{\sigma}^K$ by basic LTM model using just the state-of-the-art predictors ($LM_{Base}$), $\hat{\sigma}^K$ by LTM model using just the new predictors ($LM_{New}$), and $\hat{\sigma}^K$ by CEM objective function.

75%-100%, and 88% of the ideal ranking, in terms of F1, on the four datasets separately. This shows that $LM_{Full}$ performs well when maximizing a given top-K list’s potential.

Table 7 analyzes the resemblance between the ordering of the top-K list obtained by the algorithm and the ideal ranking. Given the ideal top-K list we can use the ranking quality measures suggested in Section 2.2, namely NDCG and ERR, to quantify the success of re-ranking the top-K list. Again, statistical significant differences in performance are marked with an asterisk using a paired two-tailed t-test with a Bonferroni correction for $p$-value < .05.

In terms or ranking performance, $LM_{Full}$ significantly out-performs the competitors, improving NDCG (ERR) over the original ranking by 20%-36.7% (44.4%-68.7%) on average, using Term as a LTM, 17%-36.6% (18.3%-83.8%) using Token Path, and 3.3%-14.6% (8%-17.4%) using WordNet. A significant improvement is demonstrated using the Beta dataset, namely 81.8% in NDCG and 94.1% in ERR. Such an improvement is attributed to the learning abilities of LTM (and CEM to almost the same extent) and substantiate our claim that it is possible to learn to rank, given a good top-K list to start with.

Comparing $LM_{Full}$ to $LM_{Base}$ and $LM_{New}$, the former provides a significantly better performance over the last two: an improvement of +11.47(±5.65)%, +1.45(±0.48)%, and +46.51(±2.41)% in F1, for the real-world datasets and an improvement of 11.53% for the synthetic dataset was demonstrated over $LM_{Base}$, while improvement of +15.25(±5.84)%, 32.07(±4.98)%, 18.86(±1.83)%, and 4.81% respectively was shown over $LM_{New}$. Interestingly, in some cases the performance of the $LM_{Base}$ and $LM_{New}$ models actually degrades compared to the original top-K ranking. This demonstrates that the combination between the newly proposed set of features with the set of the state-of-the-art predictors are more informative than to use each one separately, allowing to correctly perform the re-ranking tasks. When comparing the performance of $LM_{Base}$ and $LM_{New}$, one has that in some use cases both perform similarly (see Web-Forms and Beta), while in others one out-performed the other (in University Application $LM_{New}$ was better, while in Purchase Order $LM_{Base}$).

We further evaluate LTM using a robustness measure, which quantifies the performance of a model by measuring its improvement over majority of datasets rather than an average improvement. Ideally, a robust model would never create a re-ranked list that is worse then the original ranking.

To this end, we use the robustness index (RI) measure [40] (also referred to as reliability of improvement index), computed in our context as follows. Let $\Sigma^K$ denote a dataset with $N$ top-K list instances to re-rank, and let $\hat{\Sigma^K}$ denote the resulting re-ranked dataset. For a given quality measure (i.e., P, R, or F1), the RI of a given re-ranking model is calculated to be:

$$RI(\Sigma^K, \hat{\Sigma^K}) = \frac{n_+ (\Sigma^K, \hat{\Sigma^K}) - n_- (\Sigma^K, \hat{\Sigma^K})}{N},$$

(24)

where $n_+ (\Sigma^K, \hat{\Sigma^K})$ and $n_- (\Sigma^K, \hat{\Sigma^K})$ denote the number of instances in which the re-ranking model performed better or worse than the original top-K ranking, respectively. It is worth noting that $n_+ (\Sigma^K, \hat{\Sigma^K}) + n_- (\Sigma^K, \hat{\Sigma^K}) \leq N$ and $RI(\Sigma^K, \hat{\Sigma^K}) \in [-1, 1]$.

Table 8 summarizes our robustness analysis. Overall, in the majority of use cases, LTM achieves high precision RI values. As for recall RI, in five out of the eight use cases, LTM’s re-ranking results in a lower recall. Yet, in all use cases we observe that, overall, LTM provides a relatively high F1 RI values, providing a good tradeoff between precision and recall. The Beta synthetic dataset provides an interesting observation, where re-ranking was able to significantly improve on precision while maintaining the original recall, which was high to begin with.
Table 8: RI, n+, and n− of the $LTM_{Full}$ model for the various (dataset, ILM) pairs

<table>
<thead>
<tr>
<th>1LM</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n+</td>
<td>n−</td>
<td>RI</td>
</tr>
<tr>
<td>Term</td>
<td>85</td>
<td>14</td>
<td>.49</td>
</tr>
<tr>
<td>Token</td>
<td>74</td>
<td>7</td>
<td>.46</td>
</tr>
<tr>
<td>Word</td>
<td>113</td>
<td>14</td>
<td>.68</td>
</tr>
<tr>
<td>Net</td>
<td>113</td>
<td>14</td>
<td>.68</td>
</tr>
<tr>
<td>Path</td>
<td>74</td>
<td>7</td>
<td>.46</td>
</tr>
<tr>
<td>(a) Web-forms</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1LM</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n+</td>
<td>n−</td>
<td>RI</td>
</tr>
<tr>
<td>Term</td>
<td>12</td>
<td>28</td>
<td>-.40</td>
</tr>
<tr>
<td>Token</td>
<td>20</td>
<td>8</td>
<td>.30</td>
</tr>
<tr>
<td>Word</td>
<td>23</td>
<td>7</td>
<td>.48</td>
</tr>
<tr>
<td>Net</td>
<td>23</td>
<td>7</td>
<td>.48</td>
</tr>
<tr>
<td>Path</td>
<td>20</td>
<td>8</td>
<td>.30</td>
</tr>
<tr>
<td>(b) Purchase Order</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1LM</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n+</td>
<td>n−</td>
<td>RI</td>
</tr>
<tr>
<td>Term</td>
<td>65</td>
<td>8</td>
<td>.77</td>
</tr>
<tr>
<td>Token</td>
<td>130</td>
<td>4</td>
<td>.82</td>
</tr>
<tr>
<td>Word</td>
<td>68</td>
<td>25</td>
<td>.46</td>
</tr>
<tr>
<td>Net</td>
<td>68</td>
<td>25</td>
<td>.46</td>
</tr>
<tr>
<td>Path</td>
<td>130</td>
<td>4</td>
<td>.82</td>
</tr>
<tr>
<td>(c) University Applications</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>n+</td>
<td>n−</td>
<td>RI</td>
</tr>
<tr>
<td>143</td>
<td>0</td>
<td>.98</td>
</tr>
<tr>
<td>(d) Beta</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: P, R, and F for various (increasing) $K$ values

Figure 3 provides the result of re-ranking top-$K$ lists with LTM using various $K$ values. We measure the performance of the best match after re-ranking. Precision (P) starts high, increases further until about $K = 20$ and remains stable at a high level from that point onwards. Both recall (R) and F1 (F), however, obtain their maximum at $K = 10$, followed by a sharp drop afterwards. This result serves as an empirical validation of F1’s theoretical bound (see again Table 3). Such performance sensitivity to $K$ makes it more evident that, choosing a proper $K$ parameter is highly important for guaranteeing the best LTM performance.

4.5.3 Empirical evaluation for parameter $K$

Our concluding analysis provides an empirical evidence to the impact of the choice of $K$ on LTM performance. For that, we report on a set of experiments for $K \in \{5, 10, 15, 20, 30, 40, 50, 75, 100\}$ using the Web-Forms dataset and Term as ILM.

4.5.2 LTM Feature Importance

As part of model generation, the LambdaMart method produces trees that are used to calculate the final ranking of each similarity matrix (Section 3.2.1). These trees perform feature selection and may choose to use only part of the feature set. Using Ranklib we can export and analyze these trees, allowing an interesting reflection on the way each of the learned trees uses features in terms of weights, frequency, etc. The importance of each feature can be determined by computing the number of times a tree used a specific feature combined with the weight of this tree in computing the overall $P(\sigma)$. Table 9 reports on the model’s top-5 most informative features used for each combination of a dataset and ILM. To start with, in the majority of cases, the most important feature was one of the newly proposed predictors (features), with at least two such features selected in the top-5 for each use case. Among those features, the most notable ones are the PCA features (i.e., 1PC, 2PC, and PCA). To recall, such features reduce the complexity of the problem (introducing less noise), which in turn helps LTM to make better decisions.

Further notable in their presence are the Entropy based features (i.e., MPE, VNE, and MCD, with higher presence of the first two new features). In the majority of use cases, at least one of those features appears in the top features list. This demonstrates that Entropy in general, which counts for either diversity (i.e., MPE or MCD) or less uncertainty (i.e., VNE), plays an important role in automatic schema matching decision making.

The newly proposed matrix norm features also have a respectful presence. Recall that such features encode the amount of uncertainty for schema matching decision making. Hence, such features bias the model towards matches that are less prone to errors. All in all, combining the effect of all these newly proposed features can explain why $LTM_{Full}$ is better suited for learning than $LTM_{Base}$ or $LTM_{New}$ separately (see again Table 6).

Finally, and interestingly enough, examining further the Beta dataset, we observe that the most important features are Dominants. This can be explained by the need of LTM to improve much on precision, since Dominants is geared towards precision over recall.

5. RELATED WORK

To the best of our knowledge, ours is the first algorithm for re-ranking a top-$K$ schema match list. We propose a new algorithm, which is based on principles of Learn-to-Rank (LTR) [23] and the LambdaMart method [5]. Being a feature based algorithm, our main contribution here is the definition of a new set of features, specifically tailored to schema matching, and the definition of an appropriate optimization goal for the learning algorithm.

Predictors were first proposed by Sagi & Gal [39] and used in process matching [43], Data analysis recommendations [20], and Web table matching [36]. In this work we
show the benefit of using internalizers, predictors that use matrix properties to the task of match list re-ranking.

Top-K schema matches were proposed in the literature as an effective tool for managing uncertainty in schema matching [4,9,14,34], mostly ranking correspondences of individual attributes. In this work we rank whole matches, taking into account constraints such as 1 : 1 matching, and proposed a method for re-ranking them to increase the chance of finding better matches at the top of the list.

### 6. CONCLUSION

In this work we proposed LTM, a learn-to-match algorithm, which re-ranks top-K match lists, aiming at putting the best match at the top. We provide a theoretical bound on the size of an initial top-K lists that probabilistically guarantees the presence of the best match possible in the list. Then, we offer a set of new features, similarity matrix predictors, and show empirically their positive impact on the re-ranking task.

This line of work fits well with the quest of combining machine learning into classical data management tasks, in this case, that of schema matching. In future work, we intend to investigate the way humans and machine learning algorithms can interact to boost performance in matching tasks.

**Acknowledgements**

We thank Arik Senderovich for useful discussions.

### 7. REFERENCES


