Web-Data Integration Project

Project Report Team 9

presented by
Thomas Schmidt (1598767)
Inigo Acha
Imanoal Gonzales Vallejo

submitted to the
Data and Web Science Group
Prof. Dr. Christian Bizer
University of Mannheim

December 2022

1 Phase I - Data Translation

1.1 Use Case & Data Profiling

Dataset	Source	Format	#E[1]	#A [2]	List Of Attributes
kaggle (kaggle_f)	link to dataset	csv	7.1M (491.830[3])	11	ID, name, domain, year founded (MV), industry, size range, locality (MV), country (MV), linkedin url, current employee estimate, total employee estimate
forbes	link to dataset	csv	2.000	9	Company, Country, Sales, Profits, Assets, Market Value, Sector, Industry
dataworld (dw)	link to dataset	csv	1.924	10	Global Rank, Company, Sales, Profits, Assets, Market Value, Country, Continent, Latitude, Longitude
dbpedia	Query provided in project outline	json	3.986	11	Name, industry_label, domain, founding_year, ceos, no_emp (MV), country (MV), location (MV), revenue (MV), income (MV), assets (MV)

Table 1: Dataset Overview. All dataset only refer to the class ”Company”, * For hyperlinks pls refer,[1]# of Entities, [2]# of Attributes, [3] Number of filtered companies from the original dataset might be smaller than this number, because the final XML was extracted from the previous XML by matching the filtered names (company with the same name might exist also in excluded category).

The goal of the project was to aggregate company information from several sources. To this end we used the suggestions from the project intro slides as a foundation for our use case. We also included an additional source and amended the *dbpedia* query to extract further relevant information. Thus, the relevent entity will be a company. We relied on four different data sources which are profiled in Table 1. In order to being able to process the *kaggle* dataset, we had to filter it down. To this end, we used the ”size range” attribute and only kept the categories ”10001+”, ”5001 - 10000”, ”1001 - 5000”, ”501 - 1000”, ”201 - 500”, ”51 - 200”. This is a valid approach since the *forbes* dataset contains data about the 2000 largest companies in the world, and the *dataworld (dw)* dataset is also called ”largest companies”, containing only companies of a larger size.

Class Name	Attribute Name	Attribute Type	Contained in DS...
Company	name	String	Kaggle, Forbes, dbpedia, dw
Company	domain	String	Kaggle, dbpedia
Company	Year founded	Integer	Kaggle, dbpedia
Company	Industry	String/List	Kaggle, Forbes, dbpedia
Company	Size_range	Category	Kaggle
Company	locality	String	Kaggle, dbpedia
Company	Country	String	Kaggle, Forbes, dw, dbpedia
Company	Linkedin url	String	Kaggle
Company	Current employee estimate	Integer	Kaggle, dbpedia
Company	Total employee estimate	String	Kaggle
Company	Sales	Integer	Forbes, dw, dbpedia
Company	Profits	Integer	Forbes, dw, dbpedia
Company	Assets	Integer	Forbes, dw, dbpedia
Company	Market Value	Integer	Forbes, dw, dbpedia
Company	sector	String	Forbes
Company	Global Rank	Integer	dw
Company	Latitude	Decimal	dw
Company	Longitude	Decimal	dw
Company	ceos	list	dbpedia

Table 2: Attribute Mapping

1.2 Consolidated Schema & Transformations

The consolidated schema was created by hand. Because we only considered one entity we were able to just add additional top level fields for each attribute. We created lists for the attribute `industry` and `CEOs`. The following transformations were applied to the input datasets:

1. Monetary values were normalized to the same base (1s).

2. For *forbes* and *dw* USD was added as the `currency`. For *dbpedia* the `currency` was parsed from the datatype annotation. Since the *dbpedia* dataset came with currency information we intended to use this information via a mapping table to convert all monetary values to USD. However, it turned out that there was a huge amount of currencies involved and it was not clear from which date the exchange rate should be retrieved. Therefore, the `currency` was kept as an additional attribute.

3. A unique `ID` was generated for each record that was mapped to the target schema.

4. Missing values in *dw* were partly encoded as #N/A which confused the parse_number function. An if-clause was implemented that checked for "#" and in these cases did not put a value, and otherwise the original value.

5. For *dbpedia* the pipe concatenated fields `ceos` and `industry` were tokenized and only distinct values transferred to the respective list node.

2 Phase II - Identity Resolution

2.1 Gold Standard

In order to create the gold standard we ran initial identity resolutions with two cheap and a more complex matching rule (MR). With a threshold of 0.2 we used three different MRs. (1) Jaccard-3-Grams on company names (with frequent tokens (FT) removed) (MR1), (2) Levensthein Similarity on company names (with FT removed) (MR2), (3) A combination of 1, 2, Longest Common Subsequence (LCS), and a token-based similarity, Rogue (MR27).

The results were combined in one file, and an average similarity of the three MRs was calculated. Afterwards, the correspondences were in turn sorted by each of the four similarities. Matches with a similarity > 0.9 were labeled as sure-matches, non-matches with such a similarity as corner cases. In the range of $0.9 > sim > 0.7$ Correspondences were labeled as corner case matches or non-matches. The correspondences with lowest similarity score were also reviewed. Non-matches were labeled as sure non-matches and the few matches that still were present in this area were labeled as corner-cases. By repeating this procedure for each of the four similarity scores we had a broad coverage of different correspondences. To achieve the distribution according to the rule of thumb outlined in the lecture, which states to include 20% matching record matching pairs, 30% corner-case matches and non-matches (fuzzy), and 50% non-matching record pairs a random sample out of the labeled correspondences was drawn.The data was then split into train and test set using a python script, with a test size of 0.25 and stratified on the gold standard category (sure match, corner case, sure non-match).

After running several identity resolutions the correspondences were analyzed. False positive matches were then added to the gold standard as further corner cases. Moreover, the group size distribution was analyzed. Large groups indicated false positive matches or duplicates. From several of these groups the true and some false matches were collected and added to the gold standard as corner-cases, trying to keep the number of non-matches below 70%. Afterwards, the gold standard was reviewed for duplicates which were then removed.

2.2 Matching Rules

2.2.1 General Setup

The `company name` was the sole variable we could rely on during the identity resolution across all datasets. Other possible attributes such as `Domain` or `Year founded` oftentimes had a good amount of missing values. Therefore, we implemented several string-based comparators which are outlined in table 3 and subsequently combined them to form different MRs. We implemented 31 different MRs[1], but limit the evaluation to the best and worst performing based on their F1-scores. Nearly all *MR x Final Matching Thresh*-Combinations were tested.

			Preprocessing	
ID	Similarity measure	Parameters	PWL[1]	Rm FT[2]
C1	Jaccard on ngrams	n: ngram length	✓	(✓)
C2	Jaccard on tokens		*	(✓)
C3	Levensthein		✓	(✓)
C4	Longest Common Subsequence	Normalization Flag	✓	(✓)
C5	RogueN on Tokens [Lin, 2004]		✓	-

Table 3: Comparator Overview [1]Lowercasing and removal of punctuation and whitespaces - latter not removed for token-based similarity metrics, [2]Removal of frequent tokens, * Preprocessing done by pre-implemented similarity measure, ✓ used in comparator, (✓) optional

Every comparator had options to include certain pre- and post-processing steps. Preprocessing steps generally included lowercasing, removal of punctuation, and removal of whitespaces (the latter was omitted for token-based similarity metrics). Optionally, FTs could be removed. Moreover, different optional post-processing capabilities were implemented. These included a threshold after which the similarity was set to zero, and an option to boost or penalize the similarity in relation to a certain threshold. For example, a similarity might be boosted up using a particular function[2] above a threshold of 0.8 and penalized below. The idea behind this is that if a similarity measure reaches a certain threshold a match becomes more likely although the score might be below the final matching threshold. To increase the probability for this match to be included in the correspondences, the similarity should be increased. Conversely, if the similarity is below a certain threshold a match becomes more unlikely and the similarity score should be further penalized to have a higher impact on the final score. For example consider C5 for ”Royal Dutch Shell”

[1]Please refer to the class Comparators.MATCHING_RULES.java for an overview of the concrete combination of comparators, weights, pre- and postprocessing.

[2]To review the ”boosting” functions please refer to Comparators.AbstractT9Comparator.java [double boost(double)]

and ”Shell” with a similarity of 0.67. One might say that this consitutes a high enough similarity for this comparator to be considered a match. Thus, one could set a boosting threshold of e.g. 0.6 meaning that $(sim - 0.6)^3/2 * boostFactor^3$ would be added on top of the similarity. The boosting functions are designed to have a small impact close to the threshold and a bigger impact further away. An IR was conducted for each dataset against the *kaggle_f* dataset, because it has the largest amount of entities and was thus likely to yield a sufficient amount of correspondences as required.

2.2.2 Major Challenge - Named Entity Matching

The matching of company names has been a challenge in our project due to their ”noisyness”. We found ourselves facing similar challenges as the Dutch Central Bank [Nijhuis, 2022]. As with named entities in general every data source has a different level of detail, different data quality, and use of abbreviations among others. Regarding this, our MRs had to cater to the following challenges:

Company name: In part the name of the legal entity was used, in other cases the name of the group, and in other cases some abbreviation (e.g., Anheuser-Busch InBev Germany Holding GmbH vs. Anheuser-Busch InBev vs. AB InBev) or tokens of the name were omitted (e.g., Royal Dutch Shell vs. Shell). To address the first we implemented a LCS Comparator (C4). We used this over a Longest Common Substring to capture the described cases. For the latter we implented C5 which is a token-based comparator. However, it uses a more favorable normalization that caters to the ”Shell” example. Instead of normalizing with the distinct set of tokens (aka Jaccard) it calculates the overlap for each name over the number of its tokens and then takes the average of both.

Data Quality: We had to cope with general data quality issues which are represented for example by typos. Levenstheing similarity (C3) caters to this.

Frequent tokens: There are several tokens that have a higher frequency in company names. These include for example legal entity descriptors (limited, incorporated, ...), industy descriptors (bank, motors, pharmaceuticals, ...) and stop words (the, and, of ...). These may let names seem more similar than they actually are (General Motors vs. Hyundai Motors). We analyzed FTs across our datasets and provided the MRs with the option to remove FTs. However, this option also introduces the problem that ”General Electric” and ”General Motors” now have a similarity of 1. Thus, this option has to be considered with caution and in combination with other metrics.

Token order: Company names of different companies might be composed

[3] This example considers the X3 boost function. Others include root-like or exponential functions.

of similar tokens in a different order. Token-based similarity metrics alone would classify such names as similar altough they are not (Commercial National Financial vs. National Financial Group). This means that token order matters. The LCS Comparator and Levensthein also take this into account.

In sum, we adressed these outlined challenges by combining comparators with different strengths in MRs which we systematically evaluated at different final matching thresholds. We also implemented some new similarity metrics to cater to our needs (Comparator 4 and 5).

2.2.3 Evaluation of Matching Rules

We evaluated the thresholds 0.7, 0.8, 0.85, 0.875, and 0.9 for local matching strategies, because our analysis indicated that *dbpedia* and *kaggle_f* included duplicates. While the precision of the MRs usually increased with higher thresholds, the recall usually decreased. While the sample from table 4 does not reflect this trend because at each threshold a different MR is considered, it was clearly visible in the over 400 experiments we conducted. This is intuitive since the easy matches have high similarity scores, while more ambiguous matches although having a highish similarity, fall behind the easy matches and are thus excluded at higher thresholds. An exception were the learned MRs, that changed their model at each threshold (e.g. MR7, MR26). The trade-off then depends on the use-case at hand. Considering F1-Scores, the matching threshold of 0.85 usually outperformed the other thresholds. However, considering the following step of data fusion, we favored higher precision over recall which. This usually led to an improved group size distribution but in turn was not always the MR with the highest F1-Score.

In general the F1-Scores of the *dw - kaggle_f* IRs fell behind the other IRs and *dbpedia -kaggle_f* yielded the best results. Simple MRs (i.e. MRs that might only consider Jaccard N-Grams or Levensthein similarity) were not able to outperform more complex MRs that made use of a combination of the different similarity measures outlined in 2.2.2. For example, MR2 (dbpedia+kaggle_f(0,85)) which consideres Jaccard 3-grams without removing FT achieves a high precision of 1 but lacks behind the other MR in terms of recall. At a threshold of 0.85 it is the worst MR for the particular dataset combination.

The top performing MRs MR21 (LC), 26 (pruned tree), 27 (LC), 29 (LC) all combine all but C2 in different ways.[4] They make use of different weights and pre-and post-processing options.

For example, MR26 is a pruned tree algorithm that removes FT for C1 and C3. All comparators have a post-processing threshold between 0.3 and 0.4 where

[4]LC: Linear Combination

DS Comb + Thresh	**MR**	**B****	**P**	**R**	**F1**	**#Corr**	**Time**	**RR**$^\bullet$
dbpedia+kaggle_f(0,9)	27	S	0,93	0,92	0,93	5643	16:56	0,9937
dbpedia+kaggle_f(0,875)	29	S	0,98	0,91	0,94	3966	17:57	0,9937
dbpedia+kaggle_f(0,85)	21	S	0,93	0,96	0,95	5677	14:53	0,9937
dbpedia+kaggle_f(0,85)	2	S	1,00	0,59	0,74	2549	05:10	0,9937
dbpedia+kaggle_f(0,875)	21	SNB(20)	1,00	0,17	0,30	755	00:48	0,9997
dbpedia+kaggle_f(0,875)	21	SNB(100)	0,98	0,45	0,61	2019	03:57	0,9984
dbpedia+kaggle_f(0,875)	21	S (4g)	0,98	0,89	0,93	4437	18:40	0,9915
dw+kaggle_f(0,9)	29	S	0,94	0,73	0,83	2086	05:54	0,9941
dw+kaggle_f(0,875)	29	S	0,92	0,80	0,86	2493	05:54	0,9941
dw+kaggle_f(0,85)	26	S	0,91	0,88	0,89	4197	05:22	0,9941
dw+kaggle_f(0,85)	15	S	1,00	0,36	0,52	1575	04:27	0,9941
forbes+kaggle_f(0,9)	26	S	0,94	0,83	0,88	2712	07:58	0,9936
forbes+kaggle_f(0,9)	31 $^\circledast$	S	0,93	0,73	0,82	3163	07:18	0,9936
forbes+kaggle_f(0,875)	21	S	0,91	0,84	0,87	3289	05:16	0,9936
forbes+kaggle_f(0,85)	24	S	0,94	0,86	0,90	3254	06:30	0,9936
forbes+kaggle_f(0,85)	7	S	1,00	0,01	0,03	1549	12:24	0,9936

Table 4: Matching Rule and Blocker Performance Overview (Excerpt) Yellow: Correspondences used for data fusion. Red: Worst MR at Threshold 0.85. Blue: Blocker comparison. *S (Standard Blocker) - 3 first letters of each token, S(4g) - 4grams, SNB(n) (Sorted Neighborhood Blocker). $^\bullet$Reduction Ratio. $^\circledast$Same as MR26 without post-processing.

they are reduced to zero and make use of different boosting schemes. C4 and C5 have lower boosting thresholds compared to C1 and C3 because their similarity score might be lower, even if they encounter a true positive. Table 4 shows that MR31 which is MR26 without post-processing performs worse, which speaks for the implemented post-processing options.

The other top performing MRs tested different post-processing options and different comparator weights. A possible improvement in this area could be a more systematic evaluation of the different post-processing options that were implemented to validate their effectiveness. Also learning the parameter settings might be an option.

When analyzing the group size distribution of the correspondence sets which are put forward to data fusion, several large groups can be identified (Table 5). While this is no ideal for data fusion, other group size distributions performed even worse, so that we have to use this imperfect correspondence set in the absence of the possibility for global matching.

{MR}_{B}_{Thresh}_{ds1}_{ds2}	2	3-5	6-10	11-20	21-30	30+	PC*
29_10_87_dbpedia_kaggle_f	1376	710	75	10	2	0	1
29_10_9_dw_kaggle_f	729	426	43	3	0	0	1
26_10_90_forbes_kaggle_f	787	508	81	9	1	0	1
	61%	35%	4%	0%	0%	0%	

Table 5: Group Size Distribution of Correspondences used for Data Fusion. *Pair Completeness after Blocking

2.3 Blockers

We intended to use three different types of blockers (no blocker, symmetric, sorted neighborhood) and different blocking key generators. The following key generators were implemented, all with certain preprocessing options. (1) First letter of the company name, (2) Qgrams([Gravano et al., 2001] suggest this blocking technique to ensure a low degree of missed pairs while staying computationally efficient), (3) N starting characters of each company name token (in practice we mainly used 3 because 3 letter names are possible e.g. STX Group).

While we reduced the dataset size of *kaggle* significantly by filtering, we were still not able to evaluate a ”No Blocker” and ”3-gram blocker” against an IR that included *kaggle_f* although we increased the JVM Heap Space to 10 GB. However, we were able to circumvent this problem by using a standard blocker in combination with the first 3 letter of each name token as keys, that provided a pair completeness of 100%. We also evaluated 4-grams. Table 4 includes different blockers for *dbpedia* + *kaggle_f* with MR21 as reference, the other dataset combinations obviously behaved similarly in terms of runtime and pair completeness. Using the Sorted Neighborhood Blocker (SNB) in combination with a 3-gram key generator yieled very bad results in terms of recall. This was the case for a window size of 20 as well as larger ones, e.g. 100. The SNB was able to reach the highest reduction ratios and thus the best runtimes, but performed bad in terms of pair completeness. SNB(20) had a pair completeness of 31%, SNB(100) 50%, which is significantly worse than S(4g) with 97%. This could be due to the fact that several blocks contained more than 100 blocked elements. The first 3 letters of each token generated fewer blocked pairs while still achieving a pair completeness of 100%. The 4-grams achieved the lowest reduction ratio (still >99%) and hence had the longest runtime.

2.4 Analysis of Errors

An example of a match with large group size for *dbpedia_kaggle_f* is ”Shift Inc.” that matched with 19 companies named ”shift” (similarity of 0.89). However, none of these actually represent a match, when taking `domain` and `Year founded` into account. We did not incorporate these into many MR because they often have missing values. For example, MR28 which is actually a specialization of a top performing rule and considers a Jaccrad/URL-root combination if present and otherwise falls back to simple Jaccard, was not able to outperform the other MRs. A false positive match for example is ”Volkswagen Group” and ”volkswagen group uk ltd” (0,90). While they represent the same group, the latter represents the uk branch and thus a different entity. This is a problem that we had with other groups as well. Due to the high token overlap oftentimes they are considered a match. A missed pair includes ”Amrutanjan Healthcare” and ”amrutanjan health care limited” (0.84). Because ”health” and ”care” are separate tokens in the latter one, the token-based comparator had difficulties to get to a high enough similarity.

Summary. While the removal of FT oftentimes was useful (e.g. ”Telus Corporation”,”telus”) it could be seen that it raised new problems by creating non-matching correspondences (e.g ”Ams AG”, ”ams technologies ag”). We tried to remediate this by including more comparators that still considered FTs but were less sensible to them. LCS is able to ”skip” them, and RogueToken does not penalize the company name with missing FT as much as JaccardToken. In general it could be seen that we were able to achieve reasonable results with this. The ”subsidiary” problem could be partly addressed by having a country table and identifying tokens that refer to countries and giving them a stronger weight. A possible improvement in general would have been the removal of duplicates beforehand, to allow for global matching.

3 Phase III - Data Fusion

For data fusion we used all of our datasets. Especially *dbpedia* and *kaggle_f* had a good overlap, but also the other dataset provide some overlap. We focused on the follwing attributes: `name`, `country`, `industries`, `sales amount`, `sales currency`, `current employees`, and `year founded`.

We added the latest revision date of the datasets as provenance date, and increased the score of the *forbes* dataset compared to the others, because we consider it a high fidelity source. This will be used with the conflict resolution function ”Favour Source” for `sales amount` and `sales currency`.

Unfortunately, due to an error we were not able to evaluate the density of the datasets.[5]The attribute consistency is outlined in Table 6, along with several thresh-

[5]java.lang.NullPointerException: Cannot invoke ”de.uni_mannheim.informatik.dws.winter.model.DataSet.get()” because the return value of ”de.uni_mannheim.informatik.dws.winter.model.FusibleHashedDataSet.getSchema()” is null

Attribute	Consistency	Threshold	Conflict Resolution Function	Accuracy
YEAR_FOUNDED	0,98	+/- 5%	Median	0,47
INDUSTRY	0,36	>1 common label	Union	0,67
COUNTRY	0,82	Levensthein >0.95	Shortest String	0,13
CURRENT_EMPLOYEES	0,42	+/- 5%	Average	0
NAME	1,00	LCS >0.75	Longest String	0,73
SALES_AMOUNT	-	+/- 5%	Favour Source (forbes)	0
SALES_CURRENCY	-	JaccardToken = 1	Favour Source (forbes)	0

Table 6: Data Fusion Evaluation and Conflict Resolution Overview

olds and consistency measures. The table also outlines the conflict resolution functions we used. For example, Union was used for the `industry` list attribute. This is appropriate, because industry labels are rather vague and without a supporting taxonomy of other lookup table a more concrete overlap comparison for example is difficult. We decided to favor the *forbes* dataset for the `sales amount`, because we consider it a credible source. We used the same conflict resolution function for the according currency symbol to keep the data consistent. For the `company name` longest string was applied, as it might contain additional information such as the company type (i.e. ltd, AG, GmbH, ...).

A gold standard with 15 samples was created based on entities from the *forbes* dataset. To this end the attributes used during fusion were manually reviewed against external sources (e.g. Statista, Wikipedia, and Bloomberg).

Table 6 also outlines the Accuracy we achieved with our data fusion. Unfortunatly we have to see, that the results are rather poor. While `name` and `industry` perform somewhat ok, the other attributes fail to meet a reasonable accuracy.

4 Summary

Compared to the largest dataset we actually reduced the number of entities. Since we were not able to get the density report, we cannot make any statement about how the density increased after the data fusion. The accuracy as outlined in Table 6 is not satisfactory.

References

Luis Gravano, Panagiotis G Ipeirotis, Hosagrahar Visvesvaraya Jagadish, Nick Koudas, Shanmugauelayut Muthukrishnan, Divesh Srivastava, and others. Approximate string joins in a database (almost) for free. In *VLDB*, volume 1, pages 491–500, 2001.

Chin-Yew Lin. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL `https://aclanthology.org/W04-1013`.

Michiel Nijhuis. Company Name Matching. *Medium.com*, March 2022. URL `https://medium.com/dnb-data-science-hub/company-name-matching-6a6330710334`.