The Extremes of Good and Evil

Master Thesis

presented by
Earl Hickey
Matriculation Number 9083894

submitted to the
Data and Web Science Group
Prof. Dr. Right Name Here
University of Mannheim

August 2014

List of Tables

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 Appendix	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 mathching the filtered names (company with the same name might exist also in excluded category).

The goal of the project is to aggregate company information from several sources. To this end we used the suggestions from the project into 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 4 different datasources 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 certain 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 transfered 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. With a threshold of 0.2 we used three different matching rules. (1) Jaccard-3-Grams on company names (with frequent tokens removed) (MR1), (2) Levensthein Similarity on company names (with frequent tokens 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 matching rules 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 labeld 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 labled 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 postitive matches or duplicates. From several of these groups the true and some false matches were collected and added to the gold standard as as corner-cases, trying to keep the number of non-matches below 70%. Afterwards, the goldstandard 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. Therefore, we implemented several string-based comparators which are outlined in table 3 and subsequently combined them to form different matching rules. We implemented 27 different matching rules[1], but limit the evaluation to the best and worst performing based on their F1-scores.

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, frequent tokens (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 based on 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 the similarity should be increased. Conversly, 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" 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* dataset, because it has the largest amount of entities and is 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

[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)]

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

ID	Similarity measure	Parameters	Preprocessing PWL[1]	Rm FT[2]	Focus
1	Jaccard on ngrams	n: ngram length	✓	(✓)	Overlap
2	Jaccard on tokens		*	(✓)	Overlap
3	Levensthein		✓	(✓)	Typos / Edit-distance
4	Longest Common Subsequence	Normalization Flag	✓	(✓)	Order
5	RogueN on Tokens [2]		✓	-	Overlap

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

Bank [4]. 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 matching rules 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 Longest Common Subsequence (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 tokens from it 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 frequent tokens across our datasets and provided the matching rules with the option to remove frequent tokens. However, with this option you also introduce the problem that "General

[4]https://medium.com/dnb-data-science-hub/company-name-matching-6a6330710334

Electric" and "General Motors" now have a similarity of 1. So this option has to be considered with caution and in combination with other metrics.

Token order: Company names of different companies might be composed 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 matching rules 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 local matching strategies at the thresholds 0.7, 0.8, 0.85, 0.875, and 0.9. While the precision of the matching rules usually increased with higher thresholds, the recall usually decreased[5].This is intuitive since the easy matches have high similarity scores, while more ambigous matches although having a highish similarity, fall behind the easy matches and are thus excluded at higher thresholds. 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.

In general the F1-Scores of the *dw - kaggle_f* IRs fell behind the other IRs. At the threshold of 0.9 the simple MR5 outperformed other more complex matching rules across all datasets. This is in line with the above argumentation. Table 4 outlines the results for the top matching rules at 3 thresholds, as well as the worst matching rule at the 0.85 threshold.

MR21(LC), MR26 (pruned-tree), MR27 (LC) all combine 4 similarity measures with different strenghts and pre/post-processing options. They rely on a combination of Jaccard 3-grams(C1) and Levensthein(C3) w/o FT, as well as a LCS (C4) and RogueN on Tokens (C5). The latter two are also penalized or boosted based on a threshold.

MR5 is a simple combination of Jaccard 3-grams(C1) and Levensthein(C3) w/o frequent tokens. As described earlier, this is the most successfull matching rule at a threshold of 0.9. MR15 (the worst at the threshold 0.85) is a combination of MR5 with additionally considering Jaccard-on-Tokens(C2) with FT included. While the intuition was to capture the excluded frequent tokens fromt the other

[5]While the sample from table 4 does not reflect this trend, it was clearly visible in the over 400 experiments we conducted.

DS Comb + Thresh	MR	B*	P	R	F1	#Corr	Time	RR•
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				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(0,875)	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). •Reduction Ratio.

metrics with this similarity metric the problem is that the Jaccard-on-Tokens penalizes missing/non-overlapping tokens too strongly considering the few tokens in company names. This also led to the implementation of C5 which is a little bit more relaxed in this regard.

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([1] 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 the kaggle file 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, the standard blocker in combination with the the first 3 letter of each name token as keys proved to be a good choice in terms of efficiency while missing few matches during blocking. We also evaluated 4-grams. Table 4 includes

	2	3	4	5	6-10	11-20	21-30	30+
21_10_85_dbpedia_kaggle_f	1323	547	241	135	186	26	2	0
27_10_85_dw_kaggle_f	625	331	194	115	190	60	6	2
5_10_90_forbes_kaggle_f	678	324	178	96	178	77	17	15
(Cumulative %)	86,3%				10,0%	2,9%	0,5%	0,3%

Table 5: Group Size Distribution

different blockers for *dbpedia + kaggle_f* with MR21 as reference, the other dataset combinations obviously behaved similarly in terms of runtime and catched pairs. 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 apparently missed pairs. This could be due to the fact that several blocks contained more than 100 blocked elements were present. The first 3 letters of each token generated fewer blocked pairs while still bringing potential pairs together. The 4-grams achieved the lowest reduction ratio (still >99%) and hence had the longest runtime.

The group size distribution shows several groups with many participants, which is not ideal for data fusion. An example from *forbes* is Forbes1081 (First Financial Holding). Containing two FTs, this company matched with many companies that had a first in their name, due to the simple matching rule.

2.4 Analysis of Errors

29_10_87_dbpedia_kaggle_f

dbpedia - kaggle_f (MR21—0.85). One error already presents one problem we had to face. ”Ams AG” and ”ams technologies ag” had a similarity of 0.90. ”Technologies” and ”ag” are considered as FTs, giving them a lower weight in this MR. However, in this case it would have been the distinctive difference. ”Community Health Systems” and ”community council health systems” (0.853) barely made it into the correspondence set, however the high token overlap explains why. ”Telecommunication Company of Iran”,”telecommunication company of iran - tci” (0.82) was missed because ”telecommunication, company, of” are considered FT and two comparators thus did not work well. The LCS comparator however boosted the similarity again. ”Gulf Bank of Kuwait” and ”gulfbank-kuwait” were missed because two tokens could not be split.

dw - kaggle_f (MR27—0.85). One more problem we faced was with subsidiaries or divisions of companies. Those usually have the same tokens in their

names. For example, ”Wells Fargo”, ”wells fargo insurance services” (0.96) was a match although they refer to different entities. The high similarity is due to the fact that ”insurance, services” are considered FT. Another good example is ”Thermo Fisher Scientific”, ”thermo fisher scientific singapore” (0.91). On the other hand, ”Devon Energy”,”devon bank”(0.85) again suffer from FT removal.

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.

3 Phase III - Data Fusion

4 Summary

	Baselines			Decision Tree			
Ontology	M(edian)	G(ood)	E(vil)	results	Δ-M	Δ-G	Δ-E
#301	0.825	0.877	0.877	0.855	+0.030	-0.022	-0.022
#302	0.709	0.753	0.753	0.753	+0.044	+0.000	+0.000
#303	0.804	0.860	0.891	0.816	+0.012	-0.044	-0.075
#304	0.940	0.961	0.961	0.967	+0.027	+0.006	+0.006
Average	**0.820**	**0.863**	**0.871**	**0.848**	**+0.028**	**-0.015**	**-0.023**

Table 6: Comparison between the Good and the Evil

If you cite something, do it in the following way.

- Conference Proceedings: This problem is typically addressed by approaches for selecting the optimal matcher based on the nature of the matching task and the known characteristics of the different matching systems. Such an approach is described in [**?**].
- Journal Article: S-Match, described in [**?**], employs sound and complete reasoning procedures. Nevertheless, the underlying semantic is restricted to

propositional logic due to the fact that ontologies are interpreted as tree-like structures.

- Book: According to Euzenat and Shvaiko [**?**], we define a correspondence as follows.

These are some randomly chosen examples from other works. Take a look at the end of this thesis so see how the bibliography is included.

References

[1] Luis Gravano, Panagiotis G Ipeirotis, H V Jagadish, Nick Koudas, and T Labs. Approximate String Joins in a Database (Almost) for Free. page 10.

[2] 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.

Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Master-/Bachelorarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Ich bin mir bewusst, dass eine falsche Er- klärung rechtliche Folgen haben wird.

Mannheim, den 31.08.2014 Unterschrift