A Robust Number Parser based on Conditional Random Fields

Heiko Paulheim

Data and Web Science Group, University of Mannheim, Germany

Abstract. When processing information from unstructured sources, numbers have to be parsed in many cases to do useful reasoning on that information. However, since numbers can be expressed in different ways, a robust number parser that can cope with number representations in different shapes is required in those cases. In this paper, we show how to train such a parser based on Conditional Random Fields. As training data, we use pairs of Wikipedia infobox entries and numbers from public knowledge graphs. We show that it is possible to parse numbers at an accuracy of more than 90%.

Keywords: Number Parsing, Number Interpretation, Conditional Random Fields

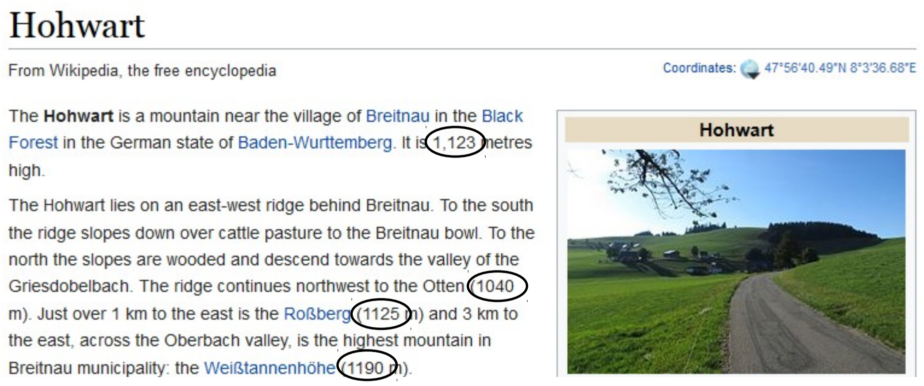
1 Introduction

Number Parsing denotes the conversion of a string representation of a number into a binary representation which is processed as an actual number, such as an integer or a double value. For many systems in which subsequent processing of numeric information (e.g., in terms of comparisons and/or arithmetic operations) is required, number parsing is a necessary preprocessing step.

Since numbers can be represented in different formats (e.g., using different thousands and decimal separators), using blanks in between or not, using different characters for negation, etc., number parsing can be challenging if no prior knowledge about the format at hand exists. For example, the representation `1,000` can be interpreted as one thousand or one, depending on the interpretation of the comma character as a thousands or a decimal separator.

When processing data from the Web, there are quite a few use cases where number parsing is essential. Examples include:

– Information extraction. The creation of knowledge bases from Web resources, e.g., DBpedia from Wikipedia [5], requires the processing of numeric information in Wikipedia infoboxes.
– Question answering on the Web [7]. Questions including comparisons or arithmetics (e.g., *What is the largest lake in the United States?*) require processing of numeric information, and, hence, parsing the corresponding numeric information.



Hohwart

From Wikipedia, the free encyclopedia

Coordinates: 47°56'40.49"N 8°3'36.68"E

The **Hohwart** is a mountain near the village of Breitnau in the Black Forest in the German state of Baden-Wurttemberg. It is 1,123 metres high.

The Hohwart lies on an east-west ridge behind Breitnau. To the south the ridge slopes down over cattle pasture to the Breitnau bowl. To the north the slopes are wooded and descend towards the valley of the Griesdobelbach. The ridge continues northwest to the Otten (1040 m). Just over 1 km to the east is the Roßberg (1125 m) and 3 km to the east, across the Oberbach valley, is the highest mountain in Breitnau municipality: the Weißtannenhöhe (1190 m).

Fig. 1. Example from a Web page with numerical information. Source: `https://en.wikipedia.org/wiki/Hohwart`, accessed May 9th, 2017

– Information integration. Approaches such as InfoGather [14], Google Fusion Tables [2] or the Mannheim Search Join Engine [6] foresee the integration of data from different sources, usually tables on the Web, into a joint database. In order to create meaningful resources here, which allow for useful processing, numbers need to be parsed into a common format.

A challenge which is common to all those use cases is that numeric data on the Web comes in a large variety of formats. In the example depicted in Fig. 1, if an intelligent agent was to answer the question whether one of the mountains was higher or lower than another, interpreting the text would imply interpreting the numbers, which, even in this small snippet from just one single source, come in two different formats (with and without and thousands separator).

Despite the presence of those use cases and challenges, robust number parsing is still far from being solved. Many named entity recognition tools and benchmarks foresee the detection of number expressions [8], but without any further parsing of the number (i.e., they merely identify a substring of a larger string which contains a numerical expression).

This paper introduces a robust number parser for numeric data from the Web. It is based on a Conditional Random Field (CRF) [4] annotator which separates a string encoding numeric information into relevant parts, such as digits, thousands or decimal separators. Based on this annotation, the string can be interpreted as a number. We create a large number of training examples using numerical values from two Semantic Web knowledge graphs generated from Wikipedia – i.e., DBpedia and YAGO – and the corresponding raw strings from Wikipedia. An evaluation on a sample of tables from the Web shows that the overall accuracy of the approach exceeds 90%.

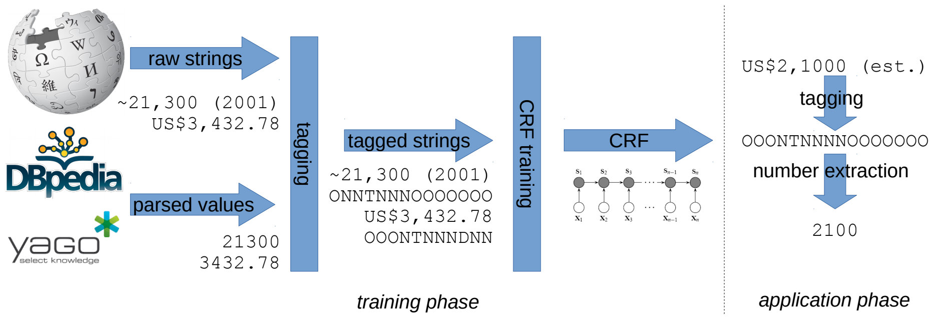


Fig. 2. Overall depiction of the approach

2 Approach

Our approach takes pairs of a number and a string representation of that number as input, as depicted in Fig. 2. While we used raw text values from Wikipedia infoboxes, along with the corresponding numerical values found in the public knowledge graphs DBpedia [5] and YAGO [12] for training, our approach can, in general, be trained with any set of such pairs.

In a preparation step, the raw strings are tagged. The example tagger processes both the *characters* of the raw string and the *digits* of the number from left to right to create the training example, using the tags N (pre-decimal digit), P (post-decimal digit), D (decimal separator), T (thousands separator), and O (other). Each character of the string is processed as follows:

```
if the current character is the current digit of the number
  if the current digit is a pre-decimal digit
    tag with N
  else
    tag with P
  current digit <- next digit
else if the current digit is a -
  tag with M
  current digit <- next digit
else if the remaining number of pre-decimal digits is 0
  tag with D
else if the remaining number of pre-decimal digits is a multitude of 3
  tag with T
else
  tag with O
```

If there are digits left after the string has been processed, the whole example is a mismatch and discarded as a training example.

After the tagging is done, the tagged strings are used train a CRF model. We use LingPipe 4.1.2[1] as an implementation of Conditional Random Fields, and the setup described in the tutorial[2]. This setup trains a linear chain CRF which uses the current token and the previous tag as features. As the only change to the example setup, we increased the number of maximum training cycles to make sure that the CRF training converged by itself.

For applying the CRF model, we pass it a raw string and inspect the tag sequence returned. A number is created by concatenating a `-` for an M tag, a `.` for a `D` tag, and the corresponding character for each `N` tag in the order in which they appear. Characters tagged as `T` and `O` are discarded. We create the numbers for all predicted sequences. If a number is not valid, it is discarded. In case two different sequences lead to the same number[3], the corresponding probabilities are added. The number with highest probability is returned.

3 Evaluation

We use two different datasets to create a large number of training examples, i.e., DBpedia [5] and YAGO [12], and we evaluate against a sample of HTML tables containing relational data, drawn from the *T2K* corpus [9, 10].

We compare our approach to two baselines, i.e., the built-in floating point parser in Java[4], and the number parsing engine of the Mannheim Search Join Engine [6], a framework for searching and on-the-fly-integration of a large number of relational tables.

3.1 Datasets

DBpedia and YAGO are large-scale knowledge graphs, which are created from Wikipedia infoboxes using a set of mapping rules which map those infoboxes to a backing ontology. For number parsing, they use hard-coded number parsing rules. Since the original strings from the Wikipedia infoboxes are also available, we are able to collect pairs of a parsed number together with the original raw text from which it was extracted (given a certain amount of noise, as explicated below).

To evaluate our approach, we use (a) a hold-out set of 1,000 pairs from both the DBpedia and the YAGO dataset which is not used for training, and (b) a sample from the T2K corpus. The latter is a collection of relational Web tables extracted from the CommonCrawl[5]. Those tables are annotated, among others, with relations defined in the DBpedia ontology. From those, we sampled 1,000 non-empty table cells which are annotated with a number-valued relation according to the ontology.

[1] `http://alias-i.com/lingpipe/`

[2] `http://alias-i.com/lingpipe/demos/tutorial/crf/read-me.html`

[3] This case may occur, e.g., if two sequences differ only in a `T` and an `O`.

[4] i.e., invoking `Double.parseDouble(s)` on a string `s`

[5] `https://commoncrawl.org`

For each parsing strategy, we compute the accuracy (where we count only deviations larger than 10^{-6} as errors, in order not to erroneously punishing rounding errors) and the root mean squared error (RMSE).

For the hold-out sets as well as the T2K set, we hand-annotated 1,000 string values each with the correct number value. Note that although we have a parsed value for the DBpedia and YAGO sets, we re-created that value manually, since we do not rely on the internal parsing of the DBpedia and YAGO frameworks. Furthermore, this gives as an estimate of how good those parsing techniques work. This comparison yields that the DBpedia built-in parser has an accuracy of 0.734, while the YAGO built-in parser has an accuracy of 0.500. This confirms shortcomings in the accuracy of the processing of numeric expressions from Wikipedia also reported in other works [1, 11, 13].

3.2 Results

From both corpora, we take different sample sizes to train the CRF (i.e., 50, 500, 5,000, 50,000, and 500,000 instances), and also mixed samples (i.e., half from DBpedia, half from YAGO). The results are depicted in table 1. We can observe that in all three cases, the mixed dataset with 500 training examples yields the best performing CRF. As expected, the CRFs trained on DBpedia and YAGO examples perform better than their counterpart on the respective holdout sets (although they are outperformed by the mixed variant, which is a bit surprising). The reason why the results constantly degrade when adding more than 500 examples is likely due to some overfitting effect.

It is also remarkable that the plain Java parser performs better in terms of RMSE in two out of three cases. The reason is that if it is capable of parsing a number, the result is usually always correct, otherwise it is not capable of parsing the number, in which case we treat the output as 0 for computing the RMSE, i.e., the absolute error is bound by the values in the dataset. In contrast, the CRF approach can also make predictions that are a few orders of magnitude away from the actual value (e.g., when erroneously concatenating a number and a year present in a string).

In terms of runtime, our approach is slower than the baselines: for parsing a single number, the Java API takes 0.008 milliseconds per string, MSJE takes 0.0215, while our approach takes 0.247.[6] On the other hand, this means that even with our approach, 4,000 strings can be parsed per second, which is fast enough for most applications.[7]

3.3 Error Analysis

In addition to the quantitative analysis, we inspected the mistakes made by our approach manually. There are several main sources of errors: strings containing

[6] Runtimes on a commodity Windows laptop

[7] The training of the CRF, however, can take up to several hours, but only needs to be performed once. An executable version with the best pre-trained CRF is available at `http://bit.ly/2qRbwDq`.

Table 1. Results on the T2K dataset and the holdout sets of DBpedia and YAGO

Dataset	T2K		DBpedia holdout		YAGO holdout	
Approach	Acc.	RMSE	Acc.	RMSE	Acc.	RMSE
Java	0.171	7.03E+09	0.223	**4.03E+09**	0.4055	**1.07E+07**
T2K/MSJE	0.2155	7.03E+09	0.3	4.24E+149	0.4495	1.69E+07
DBpedia50	0.82	8.74E+09	0.779	5.70E+09	0.936	1.52E+07
DBpedia500	0.923	2.12E+05	0.936	8.25E+09	0.963	1.52E+07
DBpedia5000	0.89	1.26E+06	0.932	8.25E+09	0.793	1.52E+07
DBpedia50000	0.853	3.77E+07	0.928	1.04E+10	0.73	1.52E+07
DBpedia500000	0.863	4.41E+06	0.928	6.29E+13	0.767	1.52E+07
YAGO50	0.823	9.94E+09	0.715	5.70E+09	0.939	1.52E+07
YAGO500	0.929	**1.84E+05**	0.927	5.70E+09	0.984	1.52E+07
YAGO5000	0.92	6.92E+05	0.901	8.25E+09	0.983	1.52E+07
YAGO50000	0.922	2.12E+05	0.911	1.42E+15	0.982	1.52E+07
YAGO500000	0.921	6.87E+05	0.906	1.04E+10	0.981	1.52E+07
Mix50	0.906	6.84E+05	0.891	5.70E+09	0.984	1.52E+07
Mix500	**0.933**	2.12E+05	**0.941**	8.25E+09	**0.986**	1.52E+07
Mix5000	0.922	2.12E+05	0.938	8.25E+09	0.984	1.52E+07
Mix50000	0.919	1.26E+06	0.932	8.25E+09	0.983	1.52E+07
Mix500000	0.918	8.65E+05	0.929	6.29E+13	0.983	1.52E+07

rare symbols adjacent to a number (such as rare currency symbols), negative numbers, and numbers that are expressed partly textually (e.g., `3.2 million`). Negative numbers are very underrepresented in our training sets, thus, the CRF always predicts a very low probability for a negative number. Partly textual numbers cannot be handled by our approach, since it only tags digits and separators in a string. Extending the approach to semi-textual numbers is subject to future work.

4 Conclusion and Outlook

In this paper, we have presented a first approach to train a robust number parser using Conditional Random Fields. The parser was tested on different datasets, where it constantly yields more than 90% accuracy.

Current limitations are the presence of rare symbols. To overcome those, better and more diverse training sets are needed, although this is not trivial: in an experiment, resampling the dataset for an equal coverage of symbols using a Kennard Stone sample [3] did not bring any improvement here. Further challenges to be addressed include: methods are needed to cope with partly textual, partly numerical representations, such as `3 million`, and the interpretation of units of measurement [11].

For use cases in which a *set* of numbers which are likely to be formatted equally (e.g., from one column in a table) are to be processed, it would also be interesting to adapt the approach in a way that it exploits those common patterns.

References

1. Fleischhacker, D., Paulheim, H., Bryl, V., Völker, J., Bizer, C.: Detecting errors in numerical linked data using cross-checked outlier detection. In: International Semantic Web Conference. pp. 357–372. Springer (2014)
2. Gonzalez, H., Halevy, A.Y., Jensen, C.S., Langen, A., Madhavan, J., Shapley, R., Shen, W., Goldberg-Kidon, J.: Google fusion tables: web-centered data management and collaboration. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. pp. 1061–1066. ACM (2010)
3. Kennard, R.W., Stone, L.A.: Computer aided design of experiments. Technometrics 11(1), 137–148 (1969), http://www.jstor.org/stable/1266770
4. Lafferty, J., McCallum, A., Pereira, F., et al.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the eighteenth international conference on machine learning, ICML. vol. 1, pp. 282–289 (2001)
5. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia – A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. Semantic Web Journal 6(2) (2013)
6. Lehmberg, O., Ritze, D., Ristoski, P., Meusel, R., Paulheim, H., Bizer, C.: The mannheim search join engine. Web semantics: science, services and agents on the World Wide Web 35, 159–166 (2015)
7. Lopez, V., Uren, V., Sabou, M., Motta, E.: Is question answering fit for the semantic web?: a survey. Semantic Web 2(2), 125–155 (2011)
8. Nadeau, D., Sekine, S.: A survey of named entity recognition and classification. Lingvisticae Investigationes 30(1), 3–26 (2007)
9. Ritze, D., Lehmberg, O., Bizer, C.: Matching html tables to dbpedia. In: Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics. p. 10. ACM (2015)
10. Ritze, D., Lehmberg, O., Oulabi, Y., Bizer, C.: Profiling the potential of web tables for augmenting cross-domain knowledge bases. In: Proceedings of the 25th International Conference on World Wide Web. pp. 251–261. International World Wide Web Conferences Steering Committee (2016)
11. Subercaze, J.: Chaudron: Extending DBpedia with measurement. In: 14th European Semantic Web Conference (2017), to appear
12. Suchanek, F.M., Kasneci, G., Weikum, G.: YAGO: A Core of Semantic Knowledge Unifying WordNet and Wikipedia. In: 16th international conference on World Wide Web. pp. 697–706 (2007)
13. Wienand, D., Paulheim, H.: Detecting incorrect numerical data in dbpedia. In: European Semantic Web Conference. pp. 504–518. Springer (2014)
14. Yakout, M., Ganjam, K., Chakrabarti, K., Chaudhuri, S.: Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. pp. 97–108. ACM (2012)