Automated methods for the comparison of natural languages

Gabriel Wittum^{1,2} · Michael Hoffer¹ · Babett Lemke¹ · Robert Jabs¹ · Arne Nägel¹

Received: 30 March 2019 / Accepted: 16 December 2019 / Published online: 18 May 2020
© The Author(s) 2020

Abstract

Starting from the general question, if there is a connection between the mathematical capabilities of a student and his native language, we aim at comparing natural languages with mathematical language quantitatively. In [20] we set up an approach to compare language structures using Natural Language Processors (NLP). However, difficulties arose with the quality of the structural analysis of the NLP used just comparing simple sentences in different but closely related natural languages. We now present a comparison of different available NLPs and discuss the results. The comparison confirms the results from [20], showing that current NLPs are not capable of analysing even simple sentences such that resulting structures between different natural languages can be compared.

Keywords Natural language processing · Tree edit distance · Comparing natural languages

1 Introduction

In Wittum et al. [20], we presented an approach to compare the structure of natural languages with the language of mathematics. The question arises from empirical studies, suggesting such a connection and its relevance for learning [1, 2, 23, 24].

On the one hand, mathematics is a language with the formulae as vocabulary and logics as grammar. On the other hand, mathematics has its own language, which has been developed over the last centuries. Mathematical formalism constitutes the core and is embedded into natural language, which is usually quite simply structured and uses only a minimum, but very specific words. To compare it with natural language, we suggested to deconstruct texts grammatically and to compare the corresponding structure graphs with corresponding graphs for mathematical texts, see Wittum et al. [20]. To that end, we suggested to use the constrained tree edit distance see [7] as distance between the trees.

The algorithm, we set up in Wittum et al. [20] uses the following steps

1. Deconstruct two or more sentences into a dependence tree
2. Compute the constrained tree edit distance of the trees
3. PCA or cluster analysis of the distance matrix

In what follows, we describe details of the components of this algorithm.

In Wittum et al. [20], we encountered problems comparing simple sentences from natural languages which are very close such as German and English. As pointed out in the aforementioned work, the reason for these problems was, that the natural language processor (NLP) used was trained with data using inconsistent annotation for the different languages. In the present paper, we repeat these tests with different NLPs using the critical sentences from Wittum et al. [20]. The result shows, that these problems are serious with all the tools tested.

2 Dependence trees and their distance

For step 1, we use Natural Language Processors (NLP) which have been trained on a large set of data and are available via an OpenSource License, as described in Wittum et al. [20]. These NLPs are all based on Probabilistic Context Free Grammars (PCFG), which have been trained on large data sets using machine learning. Pretrained PCFGs are e.g. SpaCy, [16], or CoreNLP, also known as Stanford parser [3].

Communicated by Michael Heisig.

✉ Gabriel Wittum
wittum@gcsc.uni-frankfurt.de

¹ G-CSC, University of Frankfurt, Frankfurt, Germany

² AMCS, King Abdullah University for Science and Technology, Thuwal, Saudi Arabia

```
doc = en_nlp(u'As well as considering probabilities defined over discrete sets of events, we \
also wish to consider probabilities with respect to continuous variables.')
```

Fig. 1 A dependency tree derived by SpaCy; IN = Preposition; JJ = Adjective; VBm = Verb, base form; VBG = Verb, gerund/present pple; VBN = Verb, past participle; VBP = Verb, non-3rd ps. sg. present; NN

= Noun, singular or mass; NNS = Noun, plural; . = Sentence-final punctuation; , = Comma; PRP = Personal pronoun; RB = Adverb; TO = infinitival to

Fig. 2 How to define the distance of trees T_1 and T_2 ?

Both APIs come with their learning methods, in case the user seeks to train PCFGs for own data. An alternative is the Natural Language Toolkit (NLTK) which works without pretrained methods, is slower than the pretrained tools, but offers a larger set of methods, [14]. It is good for processing smaller samples. Both SpaCy and NLTK are written in Python, while CoreNLP is written in Java. Figure 1 illustrates a dependency tree derived by SpaCy using the sentence:

As well as defined over discrete sets of events, we also wish to consider probabilities with respect to continuous variables.

After constructing dependence trees from texts, we can compare two trees by computing their tree edit distance.

Following Wittum et al. [20] and Heumann and Wittum [7], we describe the constrained tree edit distance as a measure to quantify similarity of trees. Wagner and Fischer [19] originally proposed a distance function between strings, which is the minimal cost of a sequence of edit-operations, which modify a string slightly by deleting, inserting or sub-

stituting characters. This is a generalisation of the ideas of Levenshtein [9] and Hamming [5]. The algorithm for computing such a distance is the basis for many problems which can be modelled as strings, such as DNA-sequencing. Later on, the edit distance was generalised to trees [15, 17]. There are algorithms for computing the edit-distance between ordered labeled trees, however, Kilpeläinen and Mannila [8] and Zhang et al. [21] showed that the computation of this tree edit distance is NP-complete for unordered trees. This makes it infeasible to be used for computational purposes. As a consequence, Zhang [22] proposed an algorithm to compute a slightly modified distance, the constraint tree edit distance, which has quadratic complexity. We introduce these distances below.

Let a rooted tree $T = (V, E)$ with a set of labeled vertices V and a set of edges E connecting the vertices as e.g. shown in Figs. 1 and 2. We now introduce the three *basic edit operations* shown in Fig. 3.

Definition 1 (*Basic edit operation*) The following basic operations on a labeled tree T are called basic edit operations:

- (1) Substitution: $\text{sub}(b, g)$: Replace label b in vertex labeled b by label g
- (2) Delete: $\text{del}(g)$: Delete vertex labelled g . Connects the predecessor of vertex g with the successors of g .
- (3) Insert: $\text{ins}(f; a, d)$: Insert new vertex f between vertices a and d .
or: $\text{ins}(f; d, \lambda)$ Adds new vertex f after vertex d . λ stands for an empty vertex.

Fig. 3 Transforming T_1 into T_2

The three basic edit operations illustrated. Substitution of a label (left), deletion of a vertex (center) and insertion of a vertex between two other vertices (right). The substitution step (left) is of course not necessary and can be omitted when just determining the minimum sequence of elementary edit operations to transform T_1 into T_2 .

Using these basic edit operations, we define the distance of two labeled trees, T_1 and T_2 . Let a sequence $S = (s_i)_{1 \leq i \leq n}$ of those atomic edit-operations transform tree T_1 into T_2 . By assigning a weight $\gamma(s_i) > 0$ to each basic edit operation s_i , the weight $\gamma(S)$ of each of these sequences S is just defined as the sum of its elements $\gamma(S) = \sum_{i=1}^n \gamma(s_i)$. We can now define the *tree edit distance* of the two labeled trees T_1 and T_2 .

Definition 2 (*Tree edit distance*) The tree edit distance of two labeled trees T_1 and T_2 is given by

$$d(T_1, T_2) := \min \left\{ \gamma(S) = \sum_{i=1}^n \gamma(s_i); S = (s_i)_{1 \leq i \leq n} : T_1 \mapsto T_2 \right\}. \quad (4)$$

It can be shown easily, that this distance is indeed a metric distance, that means it satisfies non-negativity, identity of indiscernibles, symmetry and the triangle inequality, if the weight γ of the edit-operations is a metric distance on the space of the labeled vertices joined with $\{\lambda\}$. Due to the finiteness of trees, it is always possible to find a sequence $S = (s_i)_{1 \leq i \leq n}$ of basic edit operations s_i , which transfer a tree T_1 into another tree T_2 . This means, that $0 \leq d(T_1, T_2) < \infty$. Computing the tree edit distance between two arbitrary unordered trees is NP complete, meaning the tree edit distance is not realistically usable for practical computations. The NP completeness, however, makes it crucial to find a distance, which inherits the nice properties of the tree edit distance, but is of moderate complexity. We use the constrained tree edit distance as described in Wittum et al. [20].

3 Cost functions for weighted edit-operations

In Wittum et al. [20], we introduced the tree edit distance for dependency trees with weighted edit-operations. The cost functions to compute those weights have to match the problem domain. For natural languages the cost function might consider the type or semantic meaning of the words that are attached to the specified vertices. As we will explain, this question is related to how words shall be represented to make them accessible to a meaningful mathematical analysis.

At first, it might seem attractive to associate words with arbitrary ids that are unique in a multilingual context, e.g.

Computer = id95106.

Zoo = id849484.

While this representation is sufficient to identifying words, it is not satisfying due to the fact that it does ignore any potential relational information that might exist between words or phrases.

Additionally, for a mathematical analysis, we need a metric that computes the distance between words which is not possible with the previously mentioned encoding. Finding good metrics between arbitrary words is an important topic in natural language processing because it is a necessity for the mathematical analysis of language semantics and structure. To accomplish that, words can be represented as vectors in a high-dimensional vector space. Such an encoding should preserve and map as much structural and semantic information to the vector representation as possible.

One of the most successful encoding techniques is *word2vec*, see Goldberg and Levy [4] and McCormick [11]. The main hypothesis of the *word2vec* model, as introduced in Mikolov et al. [12], is that the word is closer related to the words it frequently occurs with. That means, the word can be replaced by other words it is often found together with. The goal is to represent a word w_i by a dense feature vector v_i .

Fig. 4 Skip gram network architecture, from McCormick [11]

Fig. 5 Skip gram with window size 2, from McCormick [11]

The dimension of the vector space is defined by the number of unique words that shall be encoded. Each entry $v_{i,j}$ of the vector v_i represents the probability $p_{i,j}$ of word w_j to be found close to the encoded word w_i :

$$v_i = (p_{i,1}, p_{i,2}, \dots, p_{i,n})^T.$$

Let a starting encoding be given for the words $\{w_i, i = 1, \dots, n\}$. Let c be a context word, o be the current center word, u_o be the vector representation of the current center word in the current encoding, u_w the other center words in the same representation and v_c the context words under their

Fig. 6 System architecture of CoreNLP, from CoreNLP [3]

current encoding. Then the probabilities we want to obtain are defined as

$$p_{o,c} = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^n \exp(u_w^T v_c)} \quad (5)$$

where n is the number of encoded words. This is the *softmax* function applied to the scalar product of the vectors of a center and a context word in their respective encoding. The learning process is performed by the skip gram neural network model as introduced in Goldberg and Levy [4] and McCormick [11].

The fundamental idea is to obtain the probabilities by training the two-layer neural network depicted in Fig. 4 to perform the following task: For every given word w contained in the training data, select a random nearby word c . The output layer of the neural network contains the probabilities of each encoded word to co-occur with word w . The output layer is used to represent the dense encoding vector for word w mentioned above. In this context, a word w_j is defined as being a nearby word of word w_i if $|i - j| < m$ and m is referred to as size of the so called “sliding window”, see Fig. 5.

The previously discussed encoding allows us to define a metric between two words w_i and w_j , e.g., the cosine similarity between the vectors v_{w_i} and v_{w_j} , see Eq. (6).

$$\text{similarity}(v_{w_i}, v_{w_j}) = \cos(\theta) = \frac{v_{w_i} \cdot v_{w_j}}{|v_{w_i}| \cdot |v_{w_j}|} \quad (6)$$

The cosine similarity is often preferred over the euclidean distance since it is only a measure of orientation. That is, it works on non-normalised vectors as well. We can conclude that the weight (costs) for the substitution operation s_k of a

labeled vertex $vert_i$ with $vert_j$ in a given tree T can be defined as $\gamma(s_k) = 1 - |v_{w_i} - v_{w_j}|$.

4 Experiments: comparing natural languages

In Wittum et al. [20] we used SpaCy to construct dependency trees of several sentences in both, German and English. These trees are generated by language processors that are based on differently trained neural networks, one for English and another one for German. Unfortunately, SpaCy uses data sets with different annotation schemes for training each network. Since our previously obtained results heavily depend on the tree structure, we conducted experiments with different NLP tools, namely CoreNLP, SpaCy (with updated data sets) and UDPipe which claims to achieve very high tagging accuracy. These experiments are very important to the base data which we use to compute the constrained tree edit distance, see Wittum et al. [20].

4.1 NLP tools

CoreNLP is a Java-based NLP toolkit that supports POS tagging (part-of-speech tagging) as well as the construction of dependency trees. For German language support, it uses the NEGRA corpus, NEGRA [13]. The English language support is based on the Penn Treebank. The overall system architecture of CoreNLP is shown in Fig. 6.

First the system reads raw text and forwards it to a so-called annotation object. CoreNLP provides different

Fig. 7 SpaCy system architecture, from SpaCy [16]

annotators for producing additional information, such as POS tags. This information is added to the text. After all annotators processed the input text, CoreNLP returns the annotated text in either XML or plain text, see CORENLP.

SpaCy is one of the most popular NLP toolkits. It is implemented in Python. For performance-critical parts, it uses Cython (Python that compiles to native C instead of being interpreted). Just like CoreNLP, SpaCy uses different training data for English and German. Unlike CoreNLP, the primary focus of SpaCy is not to provide an NLP platform for teaching and research. This makes SpaCy one of the most efficient NLP libraries. However, it lacks some flexibility that might be required for advanced scientific applications. SpaCy's overall system architecture is shown in Fig. 7.

SpaCy works with two main data structures. The so-called *Doc* contains tokenised text as well as the annotations of those tokens. To prevent redundant copies of word vectors and

other information, SpaCy provides the *Vocab* object which is used as lookup table and can be accessed by multiple documents, see [16].

UDPipe has the advantage that the training data for all supported languages is based on the Universal Dependencies collection, see [18]. While the individual training data contains different texts, depending on the language, the UD data uses a much more homogenous tagging system among different languages. This is supposed to improve the comparability of dependency trees among different languages see [18].

4.2 The experiments

4.2.1 Experiment 1

For comparing the POS tagging and dependency tree parsing of SpaCy, CoreNLP and UDPipe we used the following sentence that showed critical in our experiments from Wittum et al. [20].

English: Steven likes long books and he also eats carrots, but he does not watch movies.

German: Steven mag lange Bücher und er isst auch Karotten, aber er guckt keine Filme.

This sentence consists of three main clauses glued together with the conjunctions “and” and “, but”. They correspond almost verbally in the two languages. From a reasonable NLP, we expect that it first discovers this basic structure and then constructs the dependency trees of the three clauses. Instead, the tools tested produce the following results.

SpaCy

See Fig. 8.

Fig. 8 Comparison of the dependence trees derived by SpaCy for the English and German test sentence in Exp. 1

Fig. 9 Comparison of the dependence trees derived by CoreNLP for the English and German test sentence in Experiment 1

Core NLP

As mentioned previously, CoreNLP uses different tagging systems for English and German. There's no automatic mapping to the Google POS tagging system. Additionally, the trees are structured differently although the English and German versions of the sentence are very similar (Fig. 9).

Even though CoreNLP did not produce the expected results, it has some advantages. The tagger can run completely independently and it is possible to develop, i.e., train custom UD based language models which use a consistent POS tagging system.

UD Pipe

UDPipe shows similar symptoms to CoreNLP where the German tree parser also used “Bücher”, the object of the first clause, instead of the verb “mag” as direct child of “<root>”. In contrast to CoreNLP, UDPipe uses consistent POS tags for both, German and English which makes it a good candidate for future research. Further advantages are the number of pretrained language models that are available for UDPipe (Fig. 10).

No one of these word processors achieves the correct structure. First, it should separate the three main clauses, just connected by the conjunctions. Then it should structure each sentence in itself.

Fig. 10 Comparison of the dependence trees derived by UD Pipe for the English and German test sentence in Experiment 1

Fig. 11 Sentence 1, English

Fig. 12 Sentence 1, German

Fig. 13 Sentence 2, English

Fig. 14 Sentence 2, German

Fig. 15 Sentence 1, English

Fig. 16 Sentence 1, German

4.2.2 Experiment 2

For the second experiment, we chose the following sentences:

Sentence 1, English: The lazy chicken crossed the large road.

Sentence 1, German: Das faule Huhn überquerte die große Straße.

Sentence 2, English: The lazy chicken crossed the big road and laid an egg.

Sentence 2, German: Das faule Huhn überquerte die große Straße und legte ein Ei.

Again, we are interested in whether the dependency trees or at least the POS tags of German and English versions of the same sentence are similar if they are structured similarly. Sentence 2 is just an extended version of sentence 1. We are particularly interested in whether the coordinating conjunction "and/und" is processed similarly in both languages.

Fig. 17 Sentence 2, English

Fig. 18 Sentence 2, German

Das faule Huhn überquerte die große Straße .

Fig. 20 Sentence 1, German

The lazy chicken crossed the big road .

Fig. 19 Sentence 1, English

Fig. 21 Sentence 2, English

Fig. 22 Sentence 2, German

SpaCy

For sentence 1, SpaCy's results are convincing. The trees are structured similarly for the German and English version of the sentence (Figs. 11, 12).

However, SpaCy returned different results for sentence 2. The conjunction “and/und” is handled differently, even though there is no apparent reason for it (Figs. 13, 14).

CoreNLP

Except for the differences in POS tagging, CoreNLP returned identical dependency trees for both sentences. Even the dependency labels are identical. Unlike in experiment 1, CoreNLP seems to recognise the conjunction correctly (Figs. 15, 16, 17, 18).

UDPipe

The results obtained for sentence 1 are as expected. The tree structure is identical. The results for sentence 2 are not identical though. UDPipe recognises the role of the verb “crossed/überquerte” correctly. For the second verb “legte/laid” the tagger recognises that it is a verb in both, English and German. But in the German version the root of the subtree is not “legte” as expected, but “Ei”. This behaviour is similar to the results obtained in experiment 1 (Figs. 19, 20, 21, 22).

5 Conclusion

As illustrated above, the current versions of multi-language Natural Language Processors are not capable to produce results, which can be compared across different languages. To successfully use the tree-edit distance in a multilingual context, it is necessary to use so-called parallel treebanks for at least two languages. Although there are some projects working on parallel treebanks for different languages, e.g. LinES [10] and Hansen-Schirra et al. [6] there is still a lot of

work left to be done for an accurate multilingual analysis of dependency trees.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Bisang, W.: hidden complexity: the neglected side of complexity and its implications. *Linguist. Vanguard* **1**(1), 177–187 (2015)
2. Brückner, S., Förster, M., Zlatkin-Troitschanskaia, O., Walstad, W.B.: Effects of prior economic education, native language, and gender on economic knowledge of first-year students in higher education. A comparative study between Germany and the USA. *Stud. Higher Educ.* **40**(3), 437–453 (2015). <https://doi.org/10.1080/03075079.2015.1004235>
3. CoreNLP: Stanford CoreNLP (2017). <https://stanfordnlp.github.io/CoreNLP/>. Accessed 10 June 2017
4. Goldberg, Y.; Levy, O.: word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv:1402.3722* (2014)
5. Hamming, R.W.: Error detecting and error correcting codes. *Bell Syst. Tech. J.* **26**, 147–160 (1950)
6. Hansen-Schirra, S., Neumann, S., Vela, M.: Multi-dimensional annotation and alignment in an English–German translation corpus, pp. 35–42 (2006). <https://doi.org/10.3115/1621034.1621040>
7. Heumann, H., Wittum, G.: The tree-edit-distance, a measure for quantifying neuronal morphology. *Neuroinformatics* **7**(3), 179–190 (2009)
8. Kilpeläinen, P., Mannila, H.: The tree inclusion problem. In: *Proceedings of International Joint Conference on the Theory and Practice of Software Development*, vol. 1, pp. 202–214 (1991)
9. Levenshtein, V.I.: Binary codes capable of correcting insertions and reversals. *Sov. Phys. Dokl.* **10**(8), 707–710 (1966)
10. LinES: An English-Swedish Parallel Treebank, Ahrenberg (2007)
11. McCormick, C.: Word2Vec tutorial - the skip-gram model (2016, April 19). <http://www.mccormickml.com>
12. Mikolov, T., et al.: Efficient estimation of word representations in vector space (2013). *arXiv:1301.3781*
13. NEGRA: <http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/>. Accessed 14 May 2018
14. NLTK: Natural language toolkit (2017). <http://www.nltk.org/>. Accessed 10 June 2017
15. Selkow, S.: The tree-to-tree editing problem. *Inf. Process. Lett.* **6**(6), 184–186 (1977)
16. SpaCy: (2017). <https://spacy.io/>. Accessed 14 May 2018
17. Tai, K.: The tree-to-tree correction problem. *J. Assoc. Comput. Mach.* **26**(3), 422–433 (1979)
18. UDPipe. <http://ufal.mff.cuni.cz/udpipe>
19. Wagner, R., Fischer, M.: The string-to-string correction problem. *J. Assoc. Comput. Mach.* **12**(1), 168–173 (1974)
20. Wittum, G., Hoffer, M., Jabs, R., Nägel, A., Bisang, W., Zlatkin-Troitschanskaia, O.: A concept for quantitative comparison of mathematical and natural language and the effect on learning. In: Dengel, A., Wittum, G., Zlatkin-Troitschanskaia, O. (eds.) *Positive Learning Technology*. Springer, Berlin (2017)
21. Zhang, K., Statman, R., Shasha, D.: On the editing distance between unordered labeled trees. *Inf. Process. Lett.* **42**, 133–139 (1992)
22. Zhang, K.: A constrained edit distance between unordered labeled trees. *Algorithmica* **15**, 205–222 (1996)
23. Zlatkin-Troitschanskaia, O., Brückner, S., Schmidt, S., Förster, M.: Messung ökonomischen Fachwissens bei Studierenden in Deutschland und den USA – Eine mehrbenenanalytische Betrachtung der hochschulinstitutionellen und individuellen Einflussfaktoren. *Unterrichtswissenschaft* **44**(1), 73–88 (2016). <https://doi.org/10.3262/UW1601073>
24. Zlatkin-Troitschanskaia, O., Förster, M., Brückner, S., Happ, R.: Insights from a German assessment of business and economics competence. In: Coates, H. (ed.) *Higher Education Learning Outcomes Assessment: International Perspectives*, pp. 175–197. Lang, Frankfurt am Main (2014). <https://doi.org/10.3726/978-3-653-04632-8>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.