Completing our understanding of diseases via text mining of mutations in natural language

Juan Miguel Cejuela[1], Aleksandar Bojchevski[1], Carsten Uhlig[1], Rustem Bekmukhametov[2], Sanjeev Kumar Karn[3], Shpend Mahmuti[1], Ashish Baghudana[5], Ankit Dubey[6], Venkata P. Satagopam[7], and Burkhard Rost[1]*

1 Technische Universität München, 2 Microsoft, 3 Ludwig Maximilians University of Munich (LMU) & Siemens Corporate Technology, Munich, 5 BITS-Pilani K. K. Birla Goa Campus, Zuarinagar, Goa, India - 403726, 6 Concur (Germany) GmbH, 7 Luxembourg Centre for Systems Biomedicine (LCSB), University of Luxembourg, 6, avenue du Swing, L-4367 Belvaux, Luxembourg

* To whom correspondence should be addressed: assistant@rostlab.org

Abstract

Extracting sequence variants from the literature can aid in the curation of databases towards our understanding of diseases and other changes in phenotype. Here, we present a new method for the extraction of mutation mentions from the biomedical literature. Our method is novel for addressing mentions written in natural language (e.g., “Glycine was substituted by Lysine at residue 18”), as compared to the previously-only-studied more standard forms (e.g., “p.18G>K” or “Gly 18 to Lys”). On different corpora, we observe that 32% to 45% of labeled documents contain at least one natural language mutation mention (NL) not written in the same text as standard form (ST), and that 5% to 12% of mentions are NL and do not have a corresponding equivalence in the same text as ST form. Our new method matches and improves the previous state of the art in ST mentions (95±1 compared to 90±2 in F-measure across corpora) while achieving a performance in NL mentions across corpora of 84±1 in F-measure. We use conditional random fields (CRFs) and traditional features combined with word embeddings unsupervisely learned from the whole MEDLINE/PubMed. Furthermore, we train on a new corpus iteratively built following an active learning approach. Our new method, API service, and corpus are freely available at https://www.tagtog.net/-corpora/IDP4+.

1 Introduction

Genetic mutations are the bread of life. They elicit biological evolution and phenotypic changes. Yet, a majority of mutations are harmful [1]. Our capacity to identify significant sequence variants, know their effect, and leverage them for drug design, turns critical in our fight against diseases. Mutagenesis observations are often deposited in the literature first, second or never in specialized databases. Catalogs like OMIM [2] continue to rely primarily on manual literature curation, an expensive and time consuming process. A simple keyword-based search on

PubMed[1] reveals over 1 million articles containing information on sequence variants; over 630 thousand for human only. In contrast, an equivalent search on UniProtKB/SwissProt [3] reveals circa 13 thousand publications[2] only and the Human Gene Mutation Database (HGMD) [4] lists around 179 thousand mutations in the professional version[3]. The information gap between the literature and specialized databases rapidly expands as genomic sequencing and PubMed continue to grow. The large disparity in numbers calls for an automatic literature extraction of mutation information.

The problem of recognizing mutation mentions has already been covered extensively in the past. It is an instance of named-entity recognition (NER). The task is defined as the extraction of text chunks that refer to a mutation, e.g., “Glycine was substituted by Lysine at residue 18” or “p.18G>K”. Both examples are in fact equivalent. The difference resides in their syntax form, the former written in natural language (NL), the latter written in a way that is more standard (ST). As expected, standard mentions are easier to recognize computationally. Further, simple mutation mentions were the primary target of previous work. How often NL mentions appear in the literature relative to ST mentions was an open question before starting this project. After our analysis with two different annotated corpora, Section 2.1, we conclude that there is a significant literature presence of NL mentions, often never paraphrased into ST form. Motivated by this, in this work we present the first method designed to specifically address NL mutation mentions. We start from the basis of the two most popular methods, MutationFinder (MF) [5] and tmVar [6]. MF uses a large set of regular expressions to primarily recognize single nucleotide polymorphisms (SNPs) written in simple ST form and slightly more complex semi-standard (SS) form (e.g., “Gly 18 to Lys” or “glycine for lysine 18”). tmVar expands on MF but uses probabilistic conditional random fields (CRFs) and recognizes various mutation types: SNPs, insertions, deletions, frameshifts, or duplications. Our work completes the picture by recognizing different mutation types for DNA or protein written in simple form or complex natural language.

2 Materials and Methods

2.1 Classification and significance of mutation mentions: ST, SS, and NL

As expected with natural language, there can be diverse opinions of what exactly a natural language mutation mention is or even what exactly constitutes a mutation mention. Thus developing a reliable classification of natural language (NL) or standard mentions (ST) is not straightforward. For example, some human annotators will consider a mention such as "alanine 27 substitution for valine" to be NL since it does not follow the standard HGVS nomenclature [7]. Others, however, may consider the same mention to be standard or semi standard (SS) since uncomplicated regular expressions potentially match such mentions. Previous mutation

[1] http://1.usa.gov/1rCrKwR
[2] http://www.uniprot.org/uniprot/?query=scope%3Avariation+OR+scope%3Amutagenesis&sort=score
[3] http://www.hgmd.cf.ac.uk/ac/index.php

extraction methods primarily followed regular expressions and did not try to extract long mutation mentions anyway. Therefore, in essence, we consider any mention that is not recognized by previous methods as NL, any mention that resembles the HGVS nomenclature as ST, and any mention in between, a twilight zone, as SS. We came up with various programmatic definers that could capture this idea. The final algorithm[4] follows a simple if-else chain to classify the three mutation subclasses: given a mutation mention, if it matches custom simple regular expressions or the tmVar method's, then is ST; else if it has 5 or more words or contains 2 or more English-dictionary words, then is NL; else if it contains 1 English-dictionary word, then is SS; else is ST. Exemplary subclass classifications of this *programmatic* definer are found in Table 1.

Subclass	Examples	MF	tmVar
ST	Q115P; Asp8Asn c.925delA; c.388+3insT; g.3912G>C delPhe1388; F33fsins; IVS3(+1)	yes no no	yes yes no
SS	3992-9g-->a mutation; codon 92, TAC-->TAT Gly 18 to Lys; leucine for arginine 90; arginine-127 into glutamine G643 to A	no yes no	yes no no
NL	glycine to arginine substitution at codon 20 Glycine was substituted by Lysine at residue 18 deletion of 10 and 8 residues from the N- and C-terminals	yes no no	no no no

Table 1. Subclassification of mutation mentions using our programmatic definer. The columns *MF* and *tmVar* indicate if the methods MutationFinder and tmVar, respectively, recognize the listed examples. The output of tmVar is verified using the demo [8].

We used the programmatic definer to guide the performance of our new method and assert whether we were effectively improving the recognition of natural language mentions. Furthermore, we used the definer to derive statistics for the considered corpora: OSIRIS [9], SETH [9,10], tmVar [6], Variome and Variome_120 [11], and the new corpus we present in this work, IDP4+ and its subsets IDP4 and nala. IDP4 and Variome are the only corpora containing full-text documents. We denote as *Variome_120* the version mentioned in [12] that contains the subset of mutations that can be mapped to a sequence position[5]. Whereas the original Variome corpus contains many vague and unmappable mutations such as “de novo mutation”, the Variome_120 is position-specific and closely resembles our corpus annotation guidelines. Figure 1 shows the proportion of NL, SS, and ST mentions for the considered corpora. As we

[4] https://github.com/Rostlab/nala/blob/HEAD/nala/preprocessing/definers.py#L65

[5] The original authors use 118 mentions but we found 2 extra NL annotations in the authors’ repository [13]. The authors communicated us that those 2 extra mutations could not be classified as either DNA’s or protein’s. In fact, they referenced a gene mutation and its translation to a protein mutation. Nonetheless, we considered these mentions relevant as NL mutation cases.

detail in Section 2.3, in our group we had annotated the IDP4 subcorpus previous to this work and did not focus on either ST or NL mentions. Likewise, the Variome corpus was not elaborated to construct a new method and therefore does not have a bias towards different mutation types [11]. In particular, it does not have a bias towards ST or NL mentions. The IDP4, Variome, and Variome_120 corpora have a high number of NL mentions compared to other corpora, 5.9%, 24.9%, and 7.5%, respectively.[6] The proportions are even higher for abstracts only, with 12.5%, 26.7%, and 11.8%, respectively. We argue that these numbers are representative of a random selection of sequence variants as expressed in the literature.

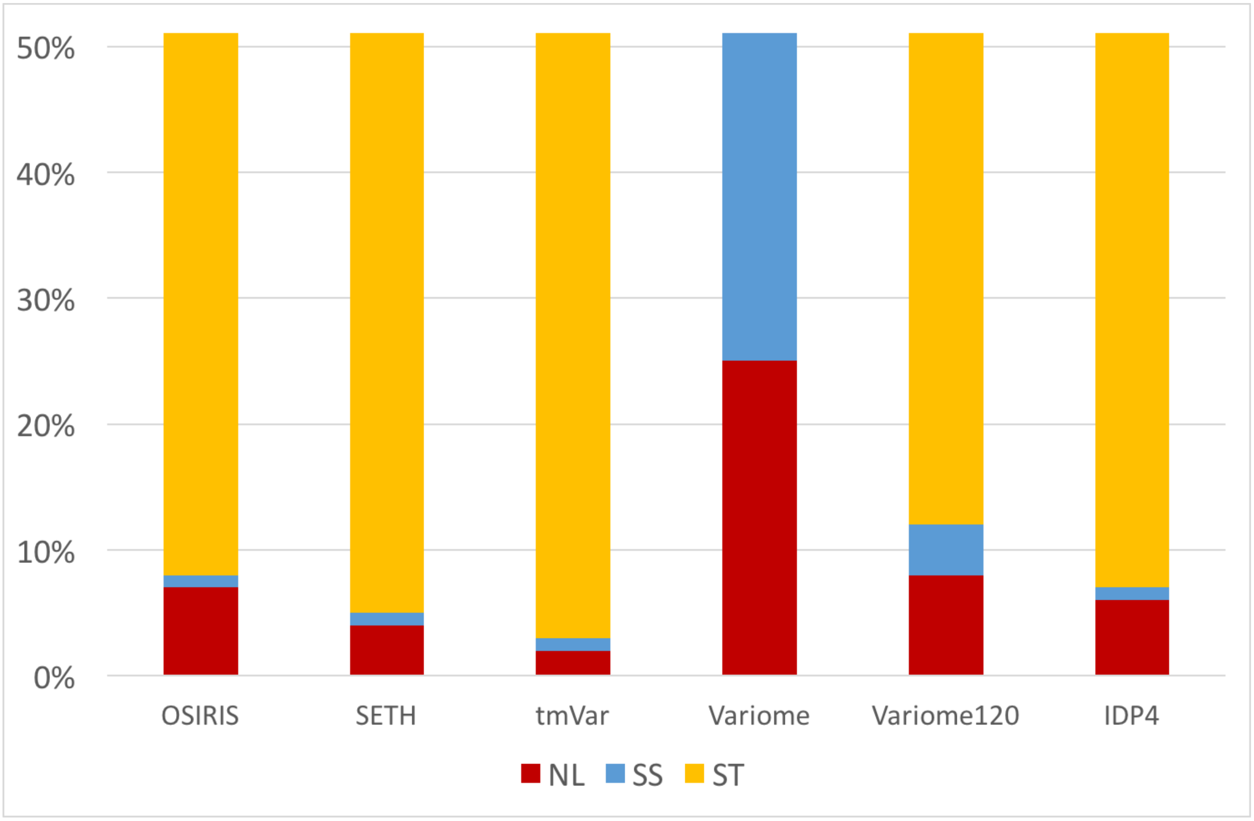

Figure 1. Accumulated proportion of NL, SS, and ST mentions for considered corpora. Due to the majority of ST mentions, the x axis is cut at 50% for clarity. Exceptionally, due to vague annotations, the Variome corpus contains a majority of SS mentions and NL+SS accounts for a 77%.

More important than the absolute proportion of NL mentions is the proportion of NL mentions that are never paraphrased as ST or SS mentions. In other words, as was the case until now, how many unique variants are left undiscovered in the literature if NL mentions are not extracted. The human annotator `abojchevski` (involved in both the IDP4 and nala corpora) manually classified as either NL or other (ST/SS) the annotations of two annotators of the IDP4 corpus and of the Variome corpus. The annotator also manually counted those NL mentions

[6] While having a comparable proportion of NL mentions at 7%, we do not consider the OSIRIS corpus for this analysis due to its exclusive focus on SNPs.

that are never paraphrased in a simpler form. The annotator followed the annotation guidelines described in Section 2.3 and his intuition that derives from Table 1 to classify NL mentions. Table 2 shows the analysis results of this classification[7]. From this, a 32% to 45% of labeled documents contain at least one NL mention not written in the same text in standard form (ST). Also, 5% to 12% of all labeled mentions are NL not written in the same text in ST form. The proportion is larger for abstracts only, from 14% to 19%.[8]

	IDP4 annotator_1		IDP4 annotator_2		Variome		Variome_120	
Stat.	A+F	A	A+F	A	A+F	A	A+F	A
Docs.	**32%**	30%	**45%**	42%	40%	22%	25%	33%
Mentions	10%	**14%**	**12%**	**19%**	**5%**	6%	8%	40%

Table 2. Significance of NL mentions. The table shows the proportion across corpora of documents that contain at least one NL mention not translated as ST (Docs.) and the proportion of total NL mentions not translated as ST (Mentions). *A*: abstracts only*;* *F*: full-text body only.

2.2 Evaluation measures

In this work we use the customary evaluation measures for named-entity recognition, namely, *precision* (P), *recall* (R), and *F-Measure* (F). We consider both *exact* matching (two entities match if their text offsets are the same) and *partial* matching (two entities match if their offsets overlap). Partial matching is more suitable to evaluate NL mentions as their boundaries are less-well defined. For instance, in the following mention extracting or not the prefix within square brackets is not differential, *"[changed a highly conserved] glutamine at residue 115 to a proline"*. We compute the total performance and the performance of each considered mutation subclass: ST, SS, NL. For this, first, test and predicted entities are classified with our programmatic definer, see Section 2.1. Second, a test entity of subclass X is correctly identified if any predicted entity matches, regardless of its subclass. Lastly, we calculate the standard error of predictions within a corpus and across corpora. Within a corpus, we calculate the standard error by randomly selecting 15% of test data without replacement in 1000 (n) bootstrap samples. If <x> is the overall performance for the entire test set and x_i is the performance for each subset, we calculate:

sigma = sqrt (SUM_i_to_n (x_i - <x>)^2 / n - 1)
SE = sigma / sqrt(n)

Across corpora, we calculate the standard error of the mean without subsampling.

[7] Note, the programmatic and manual (human annotator) definers are correlated but do not match perfectly.
[8] In Variome_120, the number grows from 8% to 40%. However, only 3 mentions are found in the abstracts and therefore the figure may not be informative.

2.3 New corpus: IDP4+

Prior to this work, we had annotated the IDP4 corpus with the focus and goal of mapping mutations to their biological sequences. For this work, we annotated the nala corpus with the focus of studying NL mutation mentions. We chose to annotate a new corpus after examining previous labeled corpora and concluding that they did not include enough NL mentions. Previously, the largest number of NL mentions was 198 for the IDP4 corpus itself and the sum of all NL mentions for all corpora was 313. By comparison, the sum of all ST mentions was 5996.

Both the IDP4 and nala corpora were directed by the same person[9] and followed similar annotation guidelines. Also both were annotated with the tool tagtog [Cejuela2014]. Given the similarities of both corpora, we combine both to form the IDP4+ corpus. Nonetheless, the different focus of the two subcorpora are reflected in some differences in the annotation process. We explain these now.

IDP4 corpus

As previously described, the focus of the corpus is the linkage of mutations to their original sequences. Consequently, we annotated on tagtog the entities *Mutation*, *Organism*, and *GGP* (gene or gene product) and annotated the relationships of *GGP* to both *Mutation* and *Organism*. We annotated documents with abstracts only and full-text documents too. We also annotated whether any figure or table in the full text contained a mutation mention, yes or no.

The document selection process was as follows:

1. Consider the organisms: `{human, arabidopsis, drosophila, Caenorhabditis elegans, Schizosaccharomyces pombe, Saccharomyces cerevisiae, mouse, rat, HIV}`
2. From all proteins of these organisms as listed on SwissProt, collect the linked PubMed identifiers that are cited for: `variation or mutagenesis`.
3. Filter in documents that their abstracts contain at least one of these keywords: `{mutation, variation, insertion, deletion, SNP}`
4. To obtain full-text articles, filter those documents that are `‘open access’` on PubMed Central.

We summarize now the annotation guidelines for `Mutation` entities.[10] For us, a mutation ideally should indicate the following three components: `W` (word), a clear word or pattern that informs about a variation and its type; `L` (letter) the changed nucleotides or residues; and `P` (position) the

[9] `jmcejuela`

[10] The reader can find the complete guidelines for the individual IDP4 corpus in [14].

changed or affected sequence location. For a candidate mutation mention, we say that `W` is present yes or no, that `L` is present yes or no, and that `P` is either exact, vague, or no. For example, `P=exact` as in "mutation of Tyr-838" or "Del 1473-IVS16(+2)" and `P=vague` as in "placed immediately downstream of I444" or "at the carboxyl end". Knowing the sequence position is particularly important since it is what enables an ulterior mapping to the original biological sequence and thus permits further biological studies. From the three components, we created the following rules to guide the annotation decisions:

- `W=yes, L=yes, P=yes/vague` → *annotate*. Examples: "p.Phe54Ser", "Arg-Thr insertion between 160 and 161 residues", "(499)leucine (TTA) to isoleucine (ATA)".
- `W=yes, L=no, P=yes` → *annotate*. Examples: "point mutation at amino acid 444", "SNPs affecting residues, 282, 319, and 333". The reason for annotation is that we can assign to the missing nucleotide/residue the unknown/any value *X*.
- Anything else → *do not annotate*

Following the annotation rules, we also annotated total gene knockouts ("Δ/Δ"), deletion of subparts ("deleted C1 domain"), or deletion of larger regions ("Deletions of chromosome 9p22.3"). We consider those position references to be exact. Furthermore, we also annotated rsids (reference SNP ID numbers).

We annotated the corpus over a 6 months period. Four annotators were involved, namely: `Ectelion`, `abojchevski`, `sanjeevkrn`, and `Shpendi`. All annotators were computer science graduates. It took circa 3 months to train the annotators and to agree on the final annotation guidelines. We threw away the first rounds of annotated documents. In the end, we annotated 157 total documents: 85 abstracts plus 72 full-text articles. To compute the inter-annotator agreement (IAA), the 4 annotators repeated the annotations for 15 documents, 10 abstracts and 5 full-text articles. The partial IAA resulted in F 91% for all mutation mentions; F 78% for NL mentions only.

nala corpus and active learning

As previously described, the focus of the corpus is the study of NL mutation mentions. In this case we annotated only abstracts for two reasons. First, as observed in Section 2.1, abstracts appear to have a larger proportion of NL mentions that are never paraphrased into simpler form. Second, annotating abstracts appears to be timewise-more efficient to obtain a larger number of NL mentions. Annotation time correlates to the length of the annotated text, also measurable by the number of tokens in the text. We can normalize the number of NL mentions of an abstract or full body by its number of tokens and divide both. The average ratios for IDP4, Variome, and Variome_120 are, respectively, 5.5, 1.6, and 3.8. Thus, abstracts contain more NL mentions per text length.

The most important characteristic of the corpus is the iterative process to build both the corpus and our new method. We build a first version of our method with standard features and train it on a *iteration_0* training set (the IDP4 corpus). We build a first version of our method (*nala_0*) closely resembling the implementation of the tmVar method. To generate the next iteration set of documents, in general *iteration_t*, we make use of the trained model on the previous iteration, *nala_t-1.* We yield candidate documents likely containing mutation mentions analogously as with the IDP4 corpus. These are further reduced by a combination of a *HighRecallFilter* and the *nala_t-1* method. The *HighRecallFillter* uses regular expressions and extracts as many NL mentions as possible, favoring recall over precision. For a candidate document, if either *HighRecallFilter* or *nala_t-1* predicts a non-ST mention (NL or SS), the document is accepted for the next selection step. We tried several parameters including the minimum number of non-ST mentions and the probability confidence of the nala predictions. However, these did not result in noticeable changes in prediction performance or annotation speed. In the last selection step, a candidate document is queried to a human annotator for acceptance or rejection. In a computer terminal, the document's abstract text is shown and the annotator accepts it if contains at least one NL mention. We repeat the process after collecting the required number of iteration documents, arbitrarily set to 10. The documents pre-annotated by *nala_t-1* are then posted to the tagtog annotation tool. The annotator manually reviews the annotations and applies on tagtog any entity offset correction, addition, or removal if necessary. Finally, the reviewed annotations are downloaded and saved as *iteration_t*. The document selection and annotation process is summarized as follows:

1. Consider the organisms: `{human, mouse}`
2. From all proteins of these organisms as listed on SwissProt, collect the linked PubMed articles that are cited for: `variation` or `mutagenesis`.
3. Filter in those documents that *HighRecallFilter* or *nala_t-1* find at least one non-ST predicted mention.
4. An annotator manually filters in those documents that contain at least one NL mention until collecting 10 total documents.
5. The documents pre-annotated by *nala_t-1* are manually reviewed on the tagtog interface and saved as *iteration_t*.

We found a greater challenge with the nala corpus in annotating NL mentions that strictly followed our previous rules of the IDP4 corpus. Not surprisingly, we found a greater variety of how mutations were expressed and we also found more often long mentions splitted into different parts or even different sentences. For these reasons, we relaxed the rules to also, at the annotator's discretion, accept mentions that did not contain exact or vague positions. Sometimes the position could be referenced indirectly somewhere else in the text of a same document. We specially relaxed the rules for insertions and deletions, e.g., "2-bp deletion in exon 6", "somatic 16-bp deletion", or "in-frame insertion of 45 nucleotides". Another particular difference of the new nala subcorpus is that genetic markers are annotated, e.g., "D17S250". Moreover, we also adopted the rule of thumb "if in doubt, include as much information as possible for the annotation". This was the opposite in the IDP4 guidelines (i.e., include as little information as possible). With this rule we did not want to create artificial NL mentions. Rather, we wanted to include information of the 3 required mutation components that were otherwise not present, e.g., "conserved residues" as substitute for position. The slight differences in annotation guidelines[11] and the bias of the nala subcorpus to annotate more NL annotations does change the distribution of mutation subclasses. We report this in the following subsection. Also in the nala corpus we did not enforce the annotation of organisms or GGP terms as with the IDP4 corpus. However, used the GNormPlus tagger [15] to automatically annotate gene/protein terms. Although not systematically, some annotators reviewed these automatic GGP annotations and manually added organism annotations. This extra information can be used in the future to further research the linkage of mutations to sequences.

We annotated the corpus over an 8 months period. Three annotators were involved, namely: `cuhlig` (bioinformatics undergraduate), `abojchevski` (computer science graduate), and `jmcejuela` (computer science postgraduate). The annotator `abojchevski` was involved in both corpora IDP4 and nala. The training for the other two annotators lasted around 1 month. The training concluded until both annotators reached a partial matching IAA of F > 90% and prior documents were discarded. In the end, we annotated 591 abstracts divided into 51 iterations for training (the *nala_training* set) and 9 extra iterations for test (the *nala_test* set). The test set with 90 documents represents a 15% of the total nala corpus. We calculated an optimal and reasonable amount of test documents by observing the standard error (SE) over a gradually-larger validation set. The SE stabilized around and after 6 iterations (60 documents). Finally, we calculated the IAA over 30 documents of the test set, which were repeated by all annotators. The partial matching IAA for the nala corpus resulted in F 92% for all mutation mentions combined; F 78% for NL mentions only.

Combined IDP4+ corpus and comparison of corpora

We release the corpora IDP4 and nala individually and also combined into the corpus IDP4+. The document contents and annotations can be downloaded in anndoc (JSON) format [16]. Overall, the IDP4+ corpus consists of 748 annotated documents, 676 abstracts and 72 full-text articles, a total of 609019 tokens, and 5445 total mutation annotations. The nala subcorpus alone contains 2108 mutation mentions. In comparison, the next largest corpus is tmVar corpus with 500 abstracts, 158975 tokens, and 1431 mutation annotations. The IDP4+ corpus has an averaged partial matching IAA of F 92% for all mutation mentions combined; F 78% for NL mentions only.

The test set totals 90 abstracts. We denote this as the nala_test set and is a synonymous of the IDP4+_test set. As we explained, the nala subcorpus and therefore the test set is biased towards NL mentions. However, the complete nala subcorpus still contains more ST mentions than NL, exactly 1097 ST (52.0%) over 841 NL (39.9%) and 170 SS (8.1%). Therefore, the test set is valid to benchmark both ST and NL mentions. The SS class is underrepresented and in general we do not focus on them in our studies. Figure 2 shows the proportion of ST, SS, and NL mentions for the new corpora and also in comparison with the Variome corpus.

[11] The reader can find the modification guidelines for the nala corpus in [14].

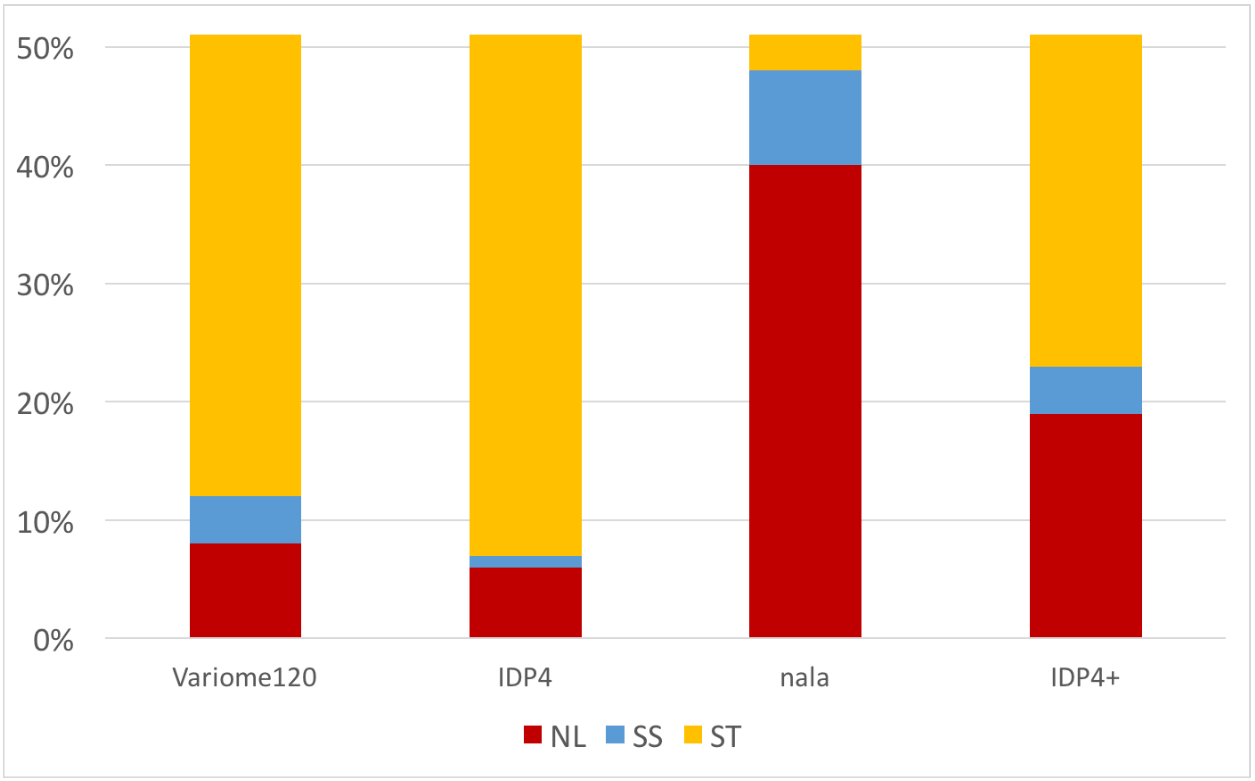

Figure 2. Accumulated proportion of NL, SS, and ST mentions for Variome_120 and the new IDP4, nala, and combined IDP4+ corpora. Due to the majority of ST mentions, the x axis is cut at 50% for clarity.

2.4 New method: nala

Following the example of some previous state-of-the-art named-entity recognition (NER) methods [6,15,17], we use conditional random fields (CRFs) as a basis for our implementation. CRFs are best suited for the labeling of sequential data, as in word sentences or pixel arrays for computer vision. We use the implementation python-crfsuite library [18], a python binding of the CRFSuite C++ library [19]. We use standard function features such as token stems, word patterns, prefix and suffix characters, presence of numbers, or the word belonging to various term dictionaries such as nucleotides, amino acids, or other biological common entities. Progressively as we added more training iterations to the nala corpus, Section 2.3, we added or changed features that performed better in cross-validation. We use our implementation of tmVar's tokenizer [6], with the difference that we do not split tokens upon case changes if it happens at the beginning of a sentence (e.g. "The" and not "T" + "he"). For token labelling we tried our implementation of the fine-grained model labeling of tmVar's with 11 labels. However, we finally chose the BIEO labelling model as in our experiments it performed better with NL mentions. We also add sanity post-processing rules such as fixing small boundary problems (for example "+1858C>T" and not "1858C>T"). Furthermore, we add special post-processing rules that can be switched on/off according to the user needs. Among others, these are `geneticmarkers=` (if false, delete annotations like “D17S250”) or `rsids=` (if false, delete annotations like “rs1801280”). As we shall see in Results, turning these simple rules on or off can have significant impacts in performance depending on the test corpus and its annotation guidelines.

As mentioned in Section 2.3, we used an active learning and iterative approach to build in parallel both the corpus and the method. In the end we did not use the confidence probability of the method predictions to guide the annotations. However, on occasion annotators accepted documents for review that specifically had annotation errors, either for missing entities, wrong offsets, or false positive predictions. The most important contribution of our method is the use of word embedding features (WE) added to the CRF model. Concretely, we use neural networks with the architecture CBOW (continuous bag of words) [20] to train on the whole MEDLINE/PubMed set of abstracts until mid 2015. We use a window size of 10 and a dimension D of 100. Tokens are converted to lowercase and different digits normalized to `0`. For each token, its generated vector of 100 real values is translated into 100 features where each real value acts as a form of weight, example: `word_embedding[0]=0.00492302`. We observed that WE features combined with hand-crafted features improved the overall performance. The recall of NL mentions improved specially. At run time the WE features can also be switched on and off, which has a significant impact in performance depending on the test corpus. Furthermore, WE features alone obtain a high performance on the nala corpus. We discuss these observations in Sections 3 and 4.1.

Finally, we note that in the end we trained on the nala subcorpus only. The performance of NL mentions was considerably better if trained on the nala_training set only. This is a direct consequence of the overrepresentation of NL mentions in the nala corpus. For ST mentions we could achieve a slightly better performance if we used the whole IDP4+ corpus and some different training parameters. However, the small performance increase did not justify the complexity of training and running two separate models (ST and NL).

3. Results

Jimeno and Verspoor [12] offer an excellent review of previous methods, corpora, and associated tasks to mutation mention recognition, such as the linkage to genes. The review concludes that SETH [10] and tmVar [6] are the best performing methods[12]. We report the performance results of the methods SETH, tmVar, and nala, evaluated on each individual considered corpus and across corpora. While we benchmarked the method MutationFinder [5], we do not report its performance as it was considerably lower than all other methods' in all categories. To run SETH we slightly modified the original scala code to print out the results in brat format [22] and wrote a python wrapper for it. To run tmVar we used its official API [23]. The API for free text did not work for us and so we ran the API version that processes only abstracts given PMIDs. This prevented us from benchmarking tmVar against the Variome_120 corpus,

[12] The EMU [21] method shows on par performance. However, it has worse exact matching performance than SETH, and worse partial matching performance than tmVar.

which contains full-text articles. As a replacement, we used the numbers provided in [12] for Variome_118 and manually updated the counts of the two extra mentions of Variome_120 using the tmVar web demo [8]. We could not use either the tmVar API to benchmark on the tmVar test set since the internal API method had been trained on its test data. As a replacement, we consider the exact matching performance provided in the tmVar paper [6] and project it to partial matching performance modeling tmVar's results on the SETH corpus.

As we detail in Section 4.2, arbitrary annotations of corpora and methods can have a significant impact in evaluation performance. For example, the tmVar and nala corpora and methods annotate rsids, whereas the SETH corpus and method does not[13]. Consequently, tmvar and nala rsid predictions are arbitrarily counted as false positive on the SETH corpus, whereas for the SETH method the absent rsid predictions on the other corpora are arbitrarily counted as false negatives. We remove such and other trivial annotation cases in the annotations and compute both the performance of a method with default evaluation and the best performance of a method on a particular corpus (e.g.: the best performance of tmVar, *tmVar_best*, on the SETH corpus disregards rsids). Figures 3-6 show the methods' performance on the respective corpora SETH, tmVar_test, Variome120, and nala_test. We report precision (P), recall, F-Measure (F), and the standard error of F-Measure (F_SE). We omit the best performance of a method if this is equal or within standard error of its default method performance. In particular, the best and default performance of every method are equal on its corresponding test corpus. Also note that the performance on NL mentions for SETH and tmVar never changes. Figure 7 shows the performance across all 4 corpora taking the average of each measure and computing the SE as explained in Section 2.2. As we could not directly run tmVar on tmVar_test and Variome_120, its subanalysis performance of ST and NL mentions only considers the results on SETH and nala_test. Finally, Figure 8 shows the subanalysis performance of NL mentions considering only the results on nala_test and Variome120 corpora (nala_test corpus for the tmVar method). Given that the nala and Variome corpora consistently annotated NL mentions, we regard these results as representative of NL performance.

SETH	Exact				Partial			
ST	P	R	F	F_SE	P	R	F	F_SE
SETH	0.9195	0.8875	0.9032	0.0012	0.9333	0.8981	0.9154	0.0011
tmVar	0.8994	0.8190	0.8573	0.0017	0.9158	0.8341	0.8731	0.0017
tmVar_best	0.9601	0.8171	0.8828	0.0016	0.9769	0.8324	0.8989	0.0015
nala	0.7769	0.9049	0.8360	0.0016	0.8398	0.9727	0.9014	0.0012
nala_best	0.9379	0.8759	0.9058	0.0010	0.9786	0.9206	0.9487	0.0007
NL	P	R	F	F_SE	P	R	F	F_SE
SETH	0.9091	0.2941	0.4444	0.0089	1.0000	0.3429	0.5106	0.0080
tmVar	0.8235	0.4242	0.5600	0.0085	1.0000	0.5556	0.7143	0.0066
nala	0.1071	0.4545	0.1734	0.0040	0.3291	1.0000	0.4952	0.0052

[13] The SETH corpus, however, and although not consistently, has some few rsids annotated.

nala_best	0.1897	0.3333	0.2418	0.0078	0.6184	0.9216	0.7402	0.0059
Combined	P	R	F	F_SE	P	R	F	F_SE
SETH	0.9196	0.8597	0.8886	0.0012	0.9345	0.8713	0.9018	0.0011
tmVar	0.8959	0.7998	0.8451	0.0018	0.9186	0.8208	0.8670	0.0018
tmVar_best	0.9545	0.7978	0.8691	0.0017	0.9777	0.8191	0.8914	0.0015
nala	0.6818	0.8816	0.7689	0.0017	0.7656	0.9736	0.8571	0.0013
nala_best	0.8779	0.8507	0.8640	0.0012	0.9421	0.9185	0.9302	0.0009

Table 3. Performance results on the SETH corpus. *F_SE:* standard error of F-measure.

tmVar_test	Exact				Partial			
ST	P	R	F	F_SE	P	R	F	F_SE
SETH	0.9671	0.7933	0.8716	0.0019	0.9947	0.8198	0.8988	0.0016
nala	0.8131	0.8112	0.8121	0.0022	0.9471	0.9641	0.9555	0.0010
nala_best	0.9014	0.8427	0.8711	0.0019	0.9823	0.9310	0.9560	0.0009
NL	P	R	F	F_SE	P	R	F	F_SE
SETH	0.3333	0.2353	0.2759	0.0198	0.6154	0.4000	0.4848	0.0150
nala	0.0700	0.4118	0.1197	0.0044	0.2273	1.0000	0.3704	0.0051
nala_best	0.1190	0.2941	0.1695	0.0086	0.4528	0.9600	0.6154	0.0058
Combined	P	R	F	F_SE	P	R	F	F_SE
SETH	0.9471	0.7716	0.8504	0.0019	0.9820	0.8008	0.8822	0.0016
tmVar*	0.9138	0.9140	0.9139		*0.9400*	*0.9400*	*0.9400*	
nala	0.6571	0.7931	0.7188	0.0023	0.8025	0.9660	0.8767	0.0015
nala_best	0.8102	0.8190	0.8146	0.0021	0.9114	0.9329	0.9220	0.0011

Table 4. Performance results on the tmVar_test corpus. * The partial matching performance of tmVar is coarsely estimated adding more than two percentage points to both precision and recall to the exact performance reported in the tmVar paper. This models the performance boost obtained on the similar SETH corpus.

Variome_120	Exact				Partial			
ST	P	R	F	F_SE	P	R	F	F_SE
SETH	0.9398	0.7647	0.8432	0.0106	0.9770	0.8095	0.8854	0.0066
nala	0.6870	0.8738	0.7692	0.0119	0.7746	0.9821	0.8661	0.0098
nala_best	0.8667	0.8922	0.8792	0.0093	0.9386	0.9817	0.9596	0.0053
NL	P	R	F	F_SE	P	R	F	F_SE
SETH	0.6667	0.3077	0.4211	0.0100	1.0000	0.6667	0.8000	0.0133
nala	0.1071	0.2500	0.1500	0.0181	0.5405	1.0000	0.7018	0.0092
nala_best	0.2667	0.3077	0.2857	0.0167	0.7826	0.9474	0.8571	0.0052
Combined	P	R	F	F_SE	P	R	F	F_SE
SETH	0.9213	0.6833	0.7847	0.0107	0.9794	0.7600	0.8559	0.0068
tmVar	*0.7500*	*0.4250*	*0.5423*		*0.9701*	*0.5417*	*0.6952*	
nala	0.5629	0.7833	0.6551	0.0105	0.7225	0.9857	0.8338	0.0062

nala_best	0.7638	0.8083	0.7854	0.0096	0.9048	0.9779	0.9399	0.0039

Table 5. Performance results on the Variome_120 corpus. The performance of tmVar is read from Variome_118 [12] plus using the tmVar web demo to predict the two extra annotations.

nala_test	Exact				Partial			
ST	P	R	F	F_SE	P	R	F	F_SE
SETH	0.9764	0.7209	0.8294	0.0040	0.9922	0.7356	0.8449	0.0040
SETH_best	0.9764	0.7294	0.8350	0.0041	0.9922	0.7442	0.8505	0.0039
tmVar	0.9424	0.7616	0.8424	0.0043	0.9931	0.8045	0.8889	0.0037
tmVar_best	0.9424	0.7706	0.8479	0.0040	0.9931	0.8136	0.8944	0.0035
nala	0.8389	0.7267	0.7788	0.0044	0.9884	0.8763	0.9290	0.0024
NL	P	R	F	F_SE	P	R	F	F_SE
SETH	0.4242	0.0979	0.1591	0.0029	1.0000	0.3210	0.4860	0.0041
tmVar	0.3571	0.1056	0.1630	0.0035	1.0000	0.4142	0.5858	0.0045
nala	0.4896	0.3310	0.3950	0.0036	0.9310	0.7459	0.8282	0.0023
Combined	P	R	F	F_SE	P	R	F	F_SE
SETH	0.8439	0.4294	0.5692	0.0038	0.9847	0.5302	0.6893	0.0031
SETH_best	0.8439	0.4320	0.5714	0.0038	0.9847	0.5331	0.6918	0.0032
tmVar	0.7650	0.4513	0.5677	0.0039	0.9959	0.6312	0.7727	0.0028
tmVar_best	0.7650	0.4540	0.5698	0.0040	0.9959	0.6345	0.7751	0.0030
nala	0.6755	0.5280	0.5927	0.0034	0.9658	0.8208	0.8874	0.0016

Table 6. Performance results on the nala_test corpus.

CrossCorpora	Exact				Partial			
ST	P	R	F	F_SE	P	R	F	F_SE
SETH	0.9507	0.7916	0.8619	0.0163	0.9743	0.8158	0.8861	0.0150
SETH_best	0.9507	0.7937	0.8633	0.0155	0.9743	0.8179	0.8875	0.0138
tmVar	0.9209	0.7903	0.8499	0.0074	0.9545	0.8193	0.8810	0.0079
tmVar_best	0.9513	0.7939	0.8654	0.0175	0.9850	0.8230	0.8967	0.0023
nala	0.7790	0.8292	0.7990	0.0154	0.8875	0.9488	0.9130	0.0191
nala_best	0.8862	0.8344	0.8587	0.0277	0.9720	0.9274	0.9483	0.0068
NL	P	R	F	F_SE	P	R	F	F_SE
SETH	0.5833	0.2338	0.3251	0.0667	0.9039	0.4327	0.5704	0.0768
tmVar	0.5903	0.2649	0.3615	0.1985	1.0000	0.4849	0.6501	0.0643
nala	0.1935	0.3618	0.2095	0.0628	0.5070	0.9365	0.5989	0.1025
nala_best	0.2663	0.3165	0.2730	0.0472	0.6962	0.8937	0.7602	0.0543
Combined	P	R	F	F_SE	P	R	F	F_SE
SETH	0.9080	0.6860	0.7732	0.0713	0.9702	0.7406	0.8323	0.0486
SETH_best	0.9080	0.6867	0.7738	0.0708	0.9702	0.7413	0.8329	0.0480
tmVar	0.8312	0.6475	0.7173	0.0949	0.9562	0.7334	0.8187	0.0536

tmVar_best	0.8458	0.6477	0.7238	0.0974	0.9709	0.7338	0.8254	0.0555
nala	0.6443	0.7465	0.6839	0.0383	0.8141	0.9365	0.8638	0.0118
nala_best	0.7819	0.7515	0.7642	0.0594	0.9310	0.9125	0.9199	0.0114

Table 7. Performance results across four considered corpora. The subanalysis of ST and NL performance for the tmVar method is computed using only the SETH and nala_test corpora.

NL computed between nala_test and Variome120 only								
	Exact				Partial			
	P	R	F	F_SE	P	R	F	F_SE
SETH	0.5455	0.2028	0.2901	0.1310	1.0000	0.4939	0.6430	0.1570
tmVar	0.3571	0.1056	0.1630	0.0035	1.0000	0.4142	0.5858	0.0045
nala	0.2984	0.2905	0.2725	0.1225	0.7358	0.8730	0.7650	0.0632
nala_best	0.3782	0.3194	0.3404	0.0547	0.8568	0.8467	0.8427	0.0145

Table 8. NL performance results across NL-relevant corpora: Variome120 and nala_test. The NL performance for the tmVar method is computed using only the nala_test corpus.

4. Discussion

4.1 Importance of word embeddings

The use of word embedding features (WE) significantly improves nala's performance on nala_test. Figure 9 compares the partial matching performance of the final nala method run without and with WE. F-Measure improves in more than 6 points with WE activated. F on NL mentions improves in 13 points (recall improves in 20 points) and F on ST improves in 2 points (recall improves in 4 points).

WE=off	P	R	F
ST	0.9938	0.8385	0.9096
NL	0.9787	0.5444	0.6996
Combined	0.9824	0.7081	0.823
WE=on	P	R	F
ST	0.9884	0.8763	0.929
NL	0.931	0.7459	0.8282
Combined	0.9658	0.8208	0.8874

Table 9. Partial matching performance of the final nala method on nala_test run without and with word embedding features.

Moreover, we found that WE alone could model well the language of mutation mentions. Figure 10 compares in 5-cross validation over nala_training the partial performance with combinations of different modules of the nala method. The tokens' lemmas[14] (TokenLemma) and WE features strictly alone achieve a high and similar performance, respectively, partial F 71.57 and F 70.21.

[14] We use the lemmatizer of the python spaCy library [13,24].

The sum of TokenLemma + WE achieves a performance in the same ballpark of the best and final nala method, F 83.55 vs F 89.84. Post-processing (PP) is the other major module of the nala method. The sum of the three achieves a partial matching F of 87.84, only 2 points behind the final nala method. It is important to highlight that WE features are entirely unsupervisedly learned and can adapt with minor adjustments to any task or corpus.[15] In contrast, the post-processing rules are crafted with great effort and are specific to mutation mentions (and even specific to the nala corpus due to differences in annotation guidelines). On the other hand, the post-processing rules once created do not suppose any noticeable overhead in running time and do achieve a major improvement in performance. All other features of the best nala method, including for example window features, increase the training running time 7-fold and the number of features 12-fold. Yet, they increase performance in only 2 points.

	Avg. Training Time (s)	Num. Features	P	R	F
TokenLemma	48	7866	0.9256	0.5834	0.7157
we	280	224	0.8686	0.5891	0.7021
TokenLemma + WE	323	8079	0.9229	0.7632	0.8355
TokenLemma + PP	50	7866	0.9369	0.7662	0.8430
TokenLemma + WE + PP	316	8079	0.9244	0.8368	0.8784
nala method (complete)	2303	96403	0.9305	0.8684	0.8984

Table 10. Performance comparison of the most important modules of the nala method. Partial matching performance is 5-cross validated over nala_training. The training time, in seconds, is the average of the running (user) time of the 5 folds. *WE*: word embeddings; *PP*: post-processing.

4.2 Comparison performance of arbitrary annotation guidelines

As we saw in Results, methods perform best in their original corpus. The performance of a method on another corpus is affected significantly by trivial and arbitrary differences in annotation guidelines. The tmVar method improves more than 2 percentage points on the SETH corpus when rsids are disregarded. The performance of nala on other corpora changes more severely due to its extra annotation of NL mentions, rsids, and genetic markers. As exemplary, Figure 11 breaks down the performance of the nala method on the SETH corpus tested with various evaluation filters. These are:

1. `WE=off`*:* word embedding features are not used at run time. This reduces the number of NL predictions.
2. `rsids=off`: reference and submitted SNP ID numbers (e.g. rs1801280 or ss221), are removed from the predictions with a post-processing regular expression.
3. `geneticmarkers=off`: analogously, genetic markers (e.g. D17S250), are removed with a post-processing regular expression.

[15] WE features implicitly use a tokenizer, which is often optimized for the task at hand, and a lemmatizer, which depends on the target language.

nala on SETH	tp	fp	fn	P	R	F
Default (3 on)	957	293	26	0.7656	0.9736	0.8571
only WE=off	894	200	63	0.8172	0.9342	0.8718
only rsids=off	945	244	27	0.7948	0.9722	0.8746
only geneticmarkers=off	939	149	33	0.8631	0.9660	0.9117
best (3 off)	879	54	78	0.9421	0.9185	0.9302

Table 11. Partial matching performance of nala on the SETH corpus using various evaluation filters.

For the nala method tested on the SETH corpus, both switching `rsid` and `geneticmarkers` off improves precision while leaving recall almost unaffected[16]. These two changes account for the largest increase in performance. Turning WE off, thus predicting less NL annotations, also improves precision due to unannotated NL mentions in SETH and other corpora. For instance, NL mutations that are otherwise counted as false positives are: "premature STOP codon at codon 667", "deletions of the exon 1 region", or "valine-to-methionine substitution in the carboxyl-terminal domain".[17] The effects of these evaluation filters are similar in all other corpora.

5. Conclusion

This paper offers the first study of the importance of mutation mentions written in natural language (NL), e.g., "84-bp deletion in exon 13 at position 1949”, relative to mentions written in standard form (ST), e.g., 1949del84. We found that the previously ignored NL mentions appear without translation into standard form in 32% to 45% of documents across annotated corpora (IDP4, Variome, and Variome_120). Moreover, 5% to 12% of all mentions across annotated corpora are written in NL and not translated in ST format. Considering only abstracts, more freely available than full-text articles, the proportion of untranslated NL mentions is 14% to 19%.

We created two new corpora, namely IDP4 and nala. Both corpora are combined into the IDP4+ corpus, the largest corpus of mutation annotations to date, with 748 annotated documents (676 abstracts and 72 full-text articles), and 5445 total mutation annotations.

The new nala method outperforms previous state of the art in ST mentions while achieving a high performance in NL mentions. With default settings and tested across four different corpora, for ST mentions the method achieves a partial matching F 91±2, for NL an F 77±6, and for all mutations combined and F 86±1.

[16] The recall decreases slightly since the SETH corpus has some few rsids and genetic markers annotated.
[17] In this case, recall decreases more significantly, revealing that SETH has a non-negligible amount of annotated NL mentions (3.7% of total annotations).

In this paper we also studied the large significance that arbitrary annotation guidelines can have in evaluation performance, for example the annotation of rsids or not. Disregarding trivial cases such as rsid or genetic marker annotations, the nala method achieves for ST mentions an F 95±1, for NL an F 84±1, and for all mutations combined an F 92±1. In contrast, the next best performance of another method is of F 91±1 for ST mentions (tmVar method), and of 64±16 for NL mentions (SETH method). The nala method uses standard conditional random fields and customary features. Furthermore, it also uses word embedding features (WE) unsupervisedly learnt from the whole PubMed/MEDLINE. WE features had already been used in the past for biomedical named-entity recognition [25–28]. We found WE to be a major driver of performance in the nala method. Features with token lemmas and WE alone could rival the best nala method. Adding post-processing rules the performance increases performance to almost match that of the best method.

Nevertheless, we observed that all evaluated methods perform best on their respective test sets. Ultimately, we argue that methods should be tested against specific and real use cases.[12,29] Methods that offer a complete pipeline of extracting mutation mentions from the literature, normalizing them, and mapping them to gene or protein sequences have recently appeared [30–32]. However, these methods primarily focus on SNPs and use outdated methods as MutationFinder. We plan next to research the performance of nala to effectively map HIV mutation mentions from the literature.

We release our method as open source on [33]. We also publish a REST API and the new corpora at [14].

Bibliography

1. Sawyer SA, Parsch J, Zhang Z, Hartl DL. Prevalence of positive selection among nearly neutral amino acid replacements in Drosophila. Proceedings of the National Academy of Sciences. 2007;104: 6504–6510. doi:10.1073/pnas.0701572104

2. OMIM - Online Mendelian Inheritance in Man [Internet]. [cited 13 May 2016]. Available: http://omim.org/

3. UniProt Consortium. UniProt: a hub for protein information. Nucleic Acids Res. 2015;43: D204–12. doi:10.1093/nar/gku989

4. Stenson PD, Ball EV, Matthew M, Phillips AD, Shiel JA, Thomas NST, et al. Human Gene Mutation Database (HGMD ®): 2003 update. Hum Mutat. 2003;21: 577–581. doi:10.1002/humu.10212

5. Caporaso JG, Baumgartner WA Jr, Randolph DA, Cohen KB, Hunter L. MutationFinder: a high-performance system for extracting point mutation mentions from text. Bioinformatics. 2007;23: 1862–1865. doi:10.1093/bioinformatics/btm235

6. Wei C-H, Harris BR, Kao H-Y, Lu Z. tmVar: a text mining approach for extracting sequence variants in biomedical literature. Bioinformatics. 2013;29: 1433–1439. doi:10.1093/bioinformatics/btt156

7. den Dunnen JT, Dalgleish R, Maglott DR, Hart RK, Greenblatt MS, McGowan-Jordan J, et al. HGVS Recommendations for the Description of Sequence Variants: 2016 Update. Hum Mutat. 2016;37: 564–569. doi:10.1002/humu.22981

8. tmVar (Mutation Recognition) demo [Internet]. [cited 13 May 2016]. Available: http://www.ncbi.nlm.nih.gov/CBBresearch/Lu/Demo/tmTools/demo/tmVar/

9. Furlong LI, Dach H, Hofmann-Apitius M, Sanz F. OSIRISv1.2: a named entity recognition system for sequence variants of genes in biomedical literature. BMC Bioinformatics. 2008;9: 84. doi:10.1186/1471-2105-9-84

10. SETH - SNP Extraction Tool for Human Variations [Internet]. [cited 13 May 2016]. Available: https://rockt.github.io/SETH/

11. Verspoor K, Jimeno Yepes A, Cavedon L, McIntosh T, Herten-Crabb A, Thomas Z, et al. Annotating the biomedical literature for the human variome. Database . 2013;2013: bat019. doi:10.1093/database/bat019

12. Jimeno Yepes A, Verspoor K. Mutation extraction tools can be combined for robust recognition of genetic variants in the literature. F1000Res. 2014;3: 18. doi:10.12688/f1000research.3-18.v2

13. readbiomed / MutationToolComparison / source / data / Variome_corpus / annotations_mutations_explicit — Bitbucket [Internet]. [cited 13 May 2016]. Available: https://bitbucket.org/readbiomed/mutationtoolcomparison/src/84bd508bea8f412fbc173bc89d83bf29caebf2ec/data/Variome_corpus/annotations_mutations_explicit/?at=default

14. Website [Internet]. [cited 13 May 2016]. Available: https://www.tagtog.net/-corpora/IDP4+

15. Wei C-H, Kao H-Y, Lu Z. GNormPlus: An Integrative Approach for Tagging Genes, Gene Families, and Protein Domains. Biomed Res Int. 2015;2015: 918710. doi:10.1155/2015/918710

16. tagtog. tagtog/tagtog-doc. In: GitHub [Internet]. [cited 13 May 2016]. Available: https://github.com/tagtog/tagtog-doc

17. Settles B, Burr S. Biomedical named entity recognition using conditional random fields and rich feature sets. Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications - JNLPBA ’04. 2004. doi:10.3115/1567594.1567618

18. tpeng. tpeng/python-crfsuite. In: GitHub [Internet]. [cited 13 May 2016]. Available: https://github.com/tpeng/python-crfsuite

19. CRFsuite - A fast implementation of Conditional Random Fields (CRFs) [Internet]. [cited 13 May 2016]. Available: http://www.chokkan.org/software/crfsuite/

20. Google Scholar Citations [Internet]. [cited 13 May 2016]. Available: https://scholar.google.com/citations?view_op=view_citation&hl=en&user=oBu8kMMAAAAJ&citation_for_view=oBu8kMMAAAAJ:UeHWp8X0CEIC

21. Doughty E, Kertesz-Farkas A, Bodenreider O, Thompson G, Adadey A, Peterson T, et al. Toward an automatic method for extracting cancer- and other disease-related point mutations from the biomedical literature. Bioinformatics. 2011;27: 408–415. doi:10.1093/bioinformatics/btq667

22. Standoff format - brat rapid annotation tool [Internet]. [cited 13 May 2016]. Available: http://brat.nlplab.org/standoff.html

23. NCBI Text Mining Tools [Internet]. [cited 13 May 2016]. Available: http://www.ncbi.nlm.nih.gov/CBBresearch/Lu/Demo/tmTools/#

24. spaCy.io. In: spaCy [Internet]. [cited 13 May 2016]. Available: https://spacy.io/

25. Tang B, Cao H, Wang X, Chen Q, Xu H. Evaluating word representation features in biomedical named entity recognition tasks. Biomed Res Int. 2014;2014: 240403. doi:10.1155/2014/240403

26. Passos A, Alexandre P, Vineet K, Andrew M. Lexicon Infused Phrase Embeddings for Named Entity Resolution. Proceedings of the Eighteenth Conference on Computational Natural Language Learning. 2014. doi:10.3115/v1/w14-1609

27. Guo J, Jiang G, Wanxiang C, Haifeng W, Ting L. Revisiting Embedding Features for Simple Semi-supervised Learning. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014. doi:10.3115/v1/d14-1012

28. Seok M, Miran S, Hye-Jeong S, Chan-Young P, Jong-Dae K, Yu-seop K. Named Entity Recognition using Word Embedding as a Feature. International Journal of Software Engineering and Its Applications. 2016;10: 93–104. doi:10.14257/ijseia.2016.10.2.08

29. Caporaso JG, Gregory Caporaso J, Nita D, Lynn Fink J, Bourne PE, Bretonnel Cohen K, et al. INTRINSIC EVALUATION OF TEXT MINING TOOLS MAY NOT PREDICT PERFORMANCE ON REALISTIC TASKS. Biocomputing 2008. 2007. doi:10.1142/9789812776136_0061

30. Mahmood ASMA, Wu T-J, Mazumder R, Vijay-Shanker K. DiMeX: A Text Mining System for Mutation-Disease Association Extraction. PLoS One. 2016;11: e0152725. doi:10.1371/journal.pone.0152725

31. Vohra S, Biggin PC. Mutationmapper: a tool to aid the mapping of protein mutation data. PLoS One. 2013;8: e71711. doi:10.1371/journal.pone.0071711

32. Ravikumar KE, Wagholikar KB, Dingcheng L, Jean-Pierre K, Hongfang L. Text mining facilitates database curation - extraction of mutation-disease associations from Bio-medical literature. BMC Bioinformatics. 2015;16. doi:10.1186/s12859-015-0609-x

33. Website [Internet]. [cited 13 May 2016]. Available: https://github.com/Rostlab/nala/