PhUSE US Connect 2020

Paper ML01

Metadata-based Auto-Programming Process – Part 3: Virtual AI assistant to automate TLG generation

Sumesh Kalappurakal, Srikanth Ramakrishnan, Shobha Rani, Janssen R&D, New Jersey, US
Harry Chen, Janssen R&D, Shanghai, China

ABSTRACT

Traditionally the programming process for statistical report generation relies heavily on manual work as we need to read and understand human language specification documents manually.

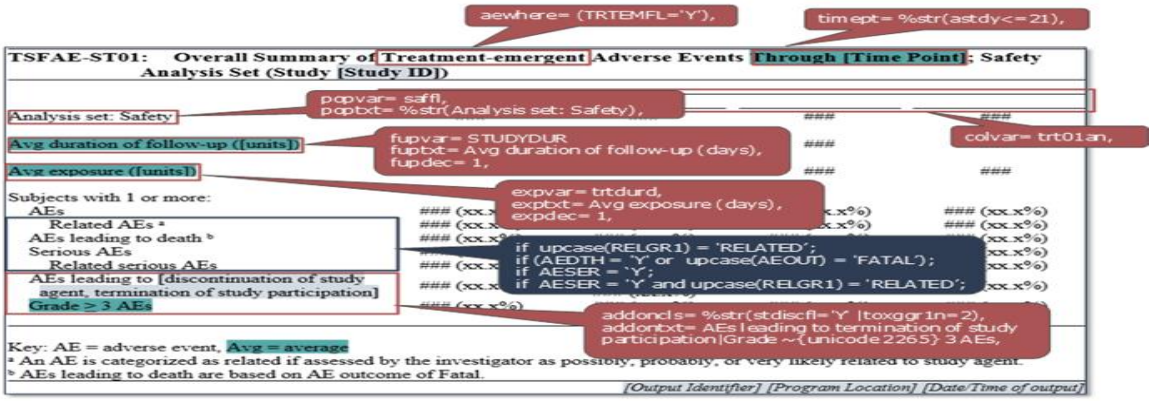
Adopting Machine Learning and leveraging the power of NLP, we developed virtual assistant system to optimize report generation process with limited human intervention. We build this by enhancing RASA open source machine learning framework with Named Entity Recognition neural models. The virtual assistant can analyze the TLG specification as user request and tackles these tasks with natural language understanding/dialogue management leading to identification of contextual named entities which can then feed the predictive models and decide the next best action for the right report generation.

To train the ML model, we scanned trial folders from 200+ compounds and collected 150K report titles, 2M macro call to learn the links between human language specification and the final programming codes. The model was then applied to do the job automatically with high quality and efficiency.

INTRODUCTION

TLG (Table/Listing/Graph) generation is common task for clinical studies and tens of thousands of TLGs need to be produced every year, and traditionally most of the task is done by manual coding process even when some standard macros or UI systems are available. We need to understand the TLG specification documents and investigate the source data and be proficient in programming to write the SAS/R codes, and usually it costs lots of time and efforts as writing and debugging the codes rely heavily on repetitive and recursive manual efforts.

Below is an example of mock-up AE table and we see the complex linkage between the text specifications and the codes.



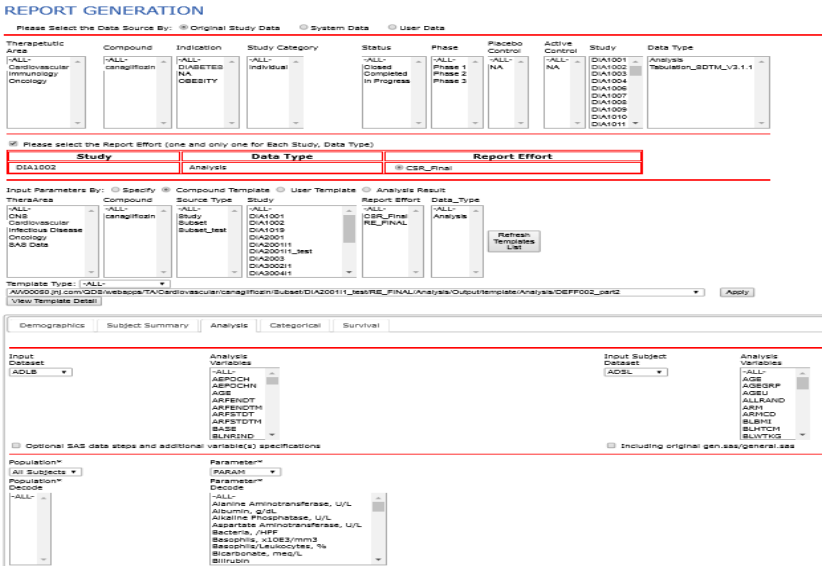
The standard macros or UI system really help a lot compared to non-standard open codes, however there are still many disadvantages. For macro call, we need to understand and remember all the standard macros and do lots of manual

typing and reviewing which is proven to be error prone. The UI system builds few linkages and can - populate some drop-down list, however user needs to understand each field and know which value to select. Tables mostly need few minor changes to fields from the standard template; however user needs to make decision which is the right field to change. For some relatively new tables, user needs to create it from scratch and manually populate all the fields. Additionally, in real life some studies can have hundreds of TLGs. Therefore, as demonstrated there is complex human cognitive processing is executed by the user to understand the text specifications and make critical decisions. There is no rule-based methodology to automate this process.

1) Macro based

```
%astrefst001(tbltype  = CORE,
             colvar   = trt01an,
             inbdsdsn = a_in.adae,
             insldsn  = a_in.adsl,
             popvar   = saffl,
             poptxt   = %str(Analysis set: Treated),
             ...
             cutoff   = pct1>=1 or pct2>=1,
             bdswhere = %str(TRTEMFL='Y' and aetoxgrn>=3 and aeser = 'Y'),
             ...
             subj1txt = AEs,
             lvlvars  = aebodsys*aedecod,
             lvltxt   = %str(System organ class|Preferred term),
             ...);
```

2) Simple UI



According to Gartner, "by 2020, natural-language generation and artificial intelligence will be a standard feature of 90% of modern business intelligence platforms"[3]. Following the trend, adopting Machine Learning and leveraging the power of NLP we can analyze human language, train the machine to convert defined algorithms to metadata and map them to the core pieces of code (e.g., SAS® or R) to automate the programing process.

We started an innovative project called Autocode in programming team. Autocode project's main objective is to automate generation of programming code (SAS® or R) for analysis datasets and analysis reports. Advantages of this project would be more automation, standardization and re-usability study after study. This project is still in the prototype stage and we will share our design, progress and experience in this paper.

This is Part 3 of the Autocode project, and we will focus on TLG generation, and our goal is to build a Virtual AI assistant to automate TLG generation.

GOAL

In the mock-up table, the title, sub-title, footnote and table layout provide all the specifications to guide programmer generate the codes to product the expected result. The title provides the most important information and other parts are linked to it.

For TLG generation we have two tasks in different stage.

1) Design stage

As a statistician/programmer, when designing a new study, we need to specify all the necessary TLGs to present the analysis result for the study. We want to search the standard library and historical study library to find the TLG title. We may have some idea (key words) in my mind, but don't want to review the hundreds of TLG titles to pick up what we want. We hope by typing some simple requirements we can find the target TLG titles.

2) Coding stage

As a programmer, after the specification are finalized, we need to read carefully the titles of all the required TLGs and understand them, then think about how to write code to generate them. We need to find the corresponding macros and fill in the right values for each parameter. The same process will repeat for another independent programmer in TLG validation process.

From technical side we can separate the task in three stages:

1) **Simple Q&A** - User gives the TLG title, the system can match it to the closest template (structure based) title and get the mockup table and template codes, by keyword search, semantic search, or sentence similarity

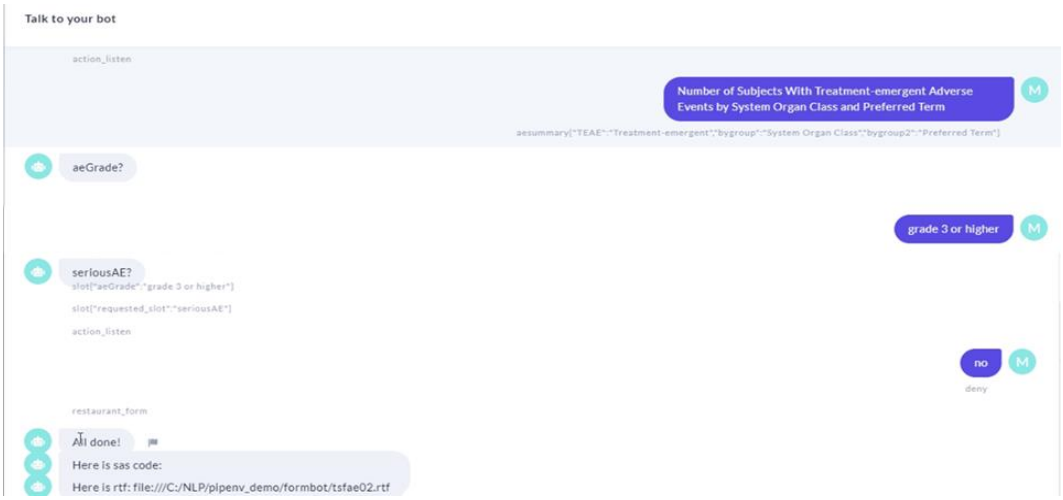
Output ID	Title
LSFAE06	Listing of Treatment-emergent Adverse Events Leading to Death
TSFDTH01	Summary of Deaths During [Study/Treatment]
TSFDTH02	Summary of Deaths During [Study/Treatment] by [Subgroup]
LSFDTH01	Listing of Deaths During [Study/Treatment]
TSFAE18	Overall Summary of [Treatment-emergent Adverse Events] by [Severity/Toxicity Grade] Through [Time Point]
TSFAE19	Number of Subjects With Treatment-emergent Adverse Events and Related Treatment-emergent Adverse Events by System Organ Class and Preferred Term
TSFAE20	Number of Subjects With Treatment-emergent Adverse Events Toxicity Grade 3 or Greater With Frequency of at Least [xx] % in [Any Treatment Group/the Combined Treatment Group] Through [Time Point] by System Organ Class and Preferred Term
TSFAE21	Number of Subjects With [Treatment-emergent Adverse Events] With Frequency of at Least [xx] % in [Any Treatment Group/the Combined Treatment Group] Through [Time Point] by System Organ Class and Preferred Term and First Onset Time
TSFAE22	Number of Subjects With [Treatment-emergent Adverse Events] With Frequency of at Least [xx] % in [Any Treatment Group/the Combined Treatment Group] Through [Time Point] by System Organ Class and Preferred Term and Onset Time
TSFLAB01	Summary of [Chemistry/Hematology] Laboratory Values and [Change/Percent Change] From Baseline Over Time Through [Time Point]
TSFLAB02	Summary of [Chemistry/Hematology] Laboratory Values and [Change/Percent Change] From Baseline at [Time Point]
TSFLAB03	Summary of [Chemistry/Hematology] Laboratory Values and [Change/Percent Change] From Baseline Over Time Through [Time Point] by [Subgroup]

2) **Query** - For any TLG title, system can understand the intent of the specification and extract necessary key words/phrases, convert the unstructured text into structured data, map it into (i.e. if pre-defined query exists) or auto-generate query scripts (get the right TLG macro, and the right parameter value for the right parameter)

TABLEID	TITLE
TSFAE-ST02	Number of Subjects With Treatment-emergent Adverse Events With Frequency of at Least 5% in Any Treatment Group by System Organ Class and Preferred Term
TSFAE-ST03	Number of Subjects With Treatment-emergent Adverse Events by System Organ Class, Preferred Term, and Sex
TSFAE-ST04	Number of Subjects With Selected Treatment-emergent Adverse Events of Interest by Special Interest Category and Preferred Term
TSFAE-ST05	Number of Subjects With Treatment-emergent Adverse Events With Frequency of at Least 5% in Any Treatment Group by System Organ Class and Preferred Term
TSFAE-ST06	Number of Subjects With Treatment-emergent Serious Adverse Events by System Organ Class and Preferred Term
TSFAE-ST07	Number of Subjects With Treatment-emergent Adverse Events Leading to Discontinuation of Study Agent by System Organ Class and Preferred Term

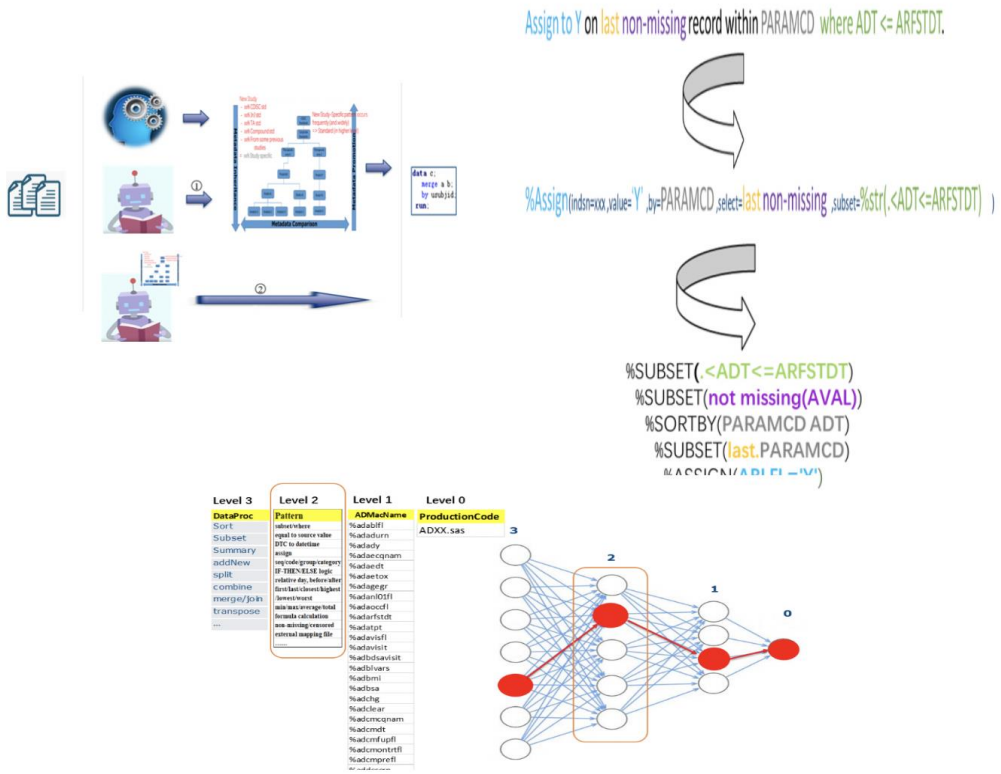
MACNAME	PARAM	EQU	PARAMVAL	PrgName tsfaest02	tsfaest03	tsfaest04	tsfaest05	tsfaest06	tsfaest07
%ASTREFST001	byvar	=		x		x	x	x	x
			sex		x				
	cutoff	=			x	x		x	x
			pct1>=5 or pct2>=5 or pct3>=5				x		
			pct1>5 or pct2>5 or pct3>5	x					
	lvltxt	=	%str(System organ class|Preferred term)	x					
			Special interest category|Preferred term			x			
			System organ class|Preferred term		x		x	x	x
	lvlvars	=	aebodsys"aedecod	x	x		x	x	x
			cq01nam"aedecod			x			
	occwhere	=	%str(TRTEMFL='Y' and AESER='Y')					x	
			%str(TRTEMFL='Y' and AEACN='DRUG WITHDRAWN')						x
			%str(TRTEMFL='Y' and CQ01NAM^=' ')	x	x	x	x		
	subj1txt	=	AEs	x	x		x		
			AEs leading to discontinuation						x
			SAEs					x	
			special interest AEs			x			

3) **Virtual Assistant Chat** – we envisioned the system to expose itself to the user as an AI assistant which enables required-rounds of interactions with user, when it needs further information from user for completing the task. While the virtual assistant is initially trained for the scope mentioned above, we have defined a roadmap of tasks which could be automated through a similar construct and hence leveraging the investment in this AI methodology.



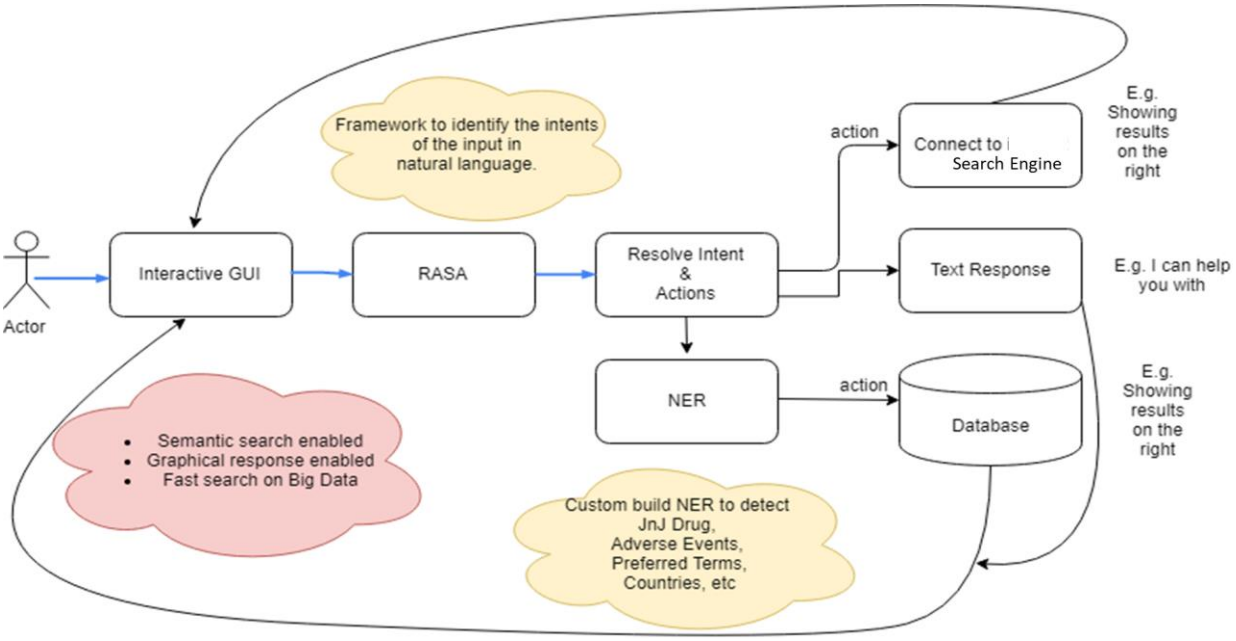
SOLUTION OVERVIEW

Our model analyzes document specifications and converts into executable SAS® code for TLG generation by adopting machine learning (ML) along with a combination of cognitive (neural) and rule-based programming. We have two-stage design to firstly map to standard macro call (level 1) and secondly further map to pattern macros (level 2) with more flexibility to cover wider scenarios including trial-specific scenarios.



To elaborate further on the above solution introduction, the high-level flow of the application can be summarized as follows:

- Interactive user interface which suggests and captures the user question and feeds the NLU engine (enhanced RASA)
- NLU Module is responsible for obtaining and interpreting the conversation text (convert unstructured data to structured data). This module also retains the state to preserve historic conversation
- The Neural Net NER engine invokes a knowledge graph to combine intent with known entities or prediction on entities. The next best action such as invoking a SOP, converting the natural question to a database or search engine query is automatically taken.
- Text Response- module gives a natural language response to the user along with the findings.



We will now explain the various components for the above architecture.

RASA OPEN SOURCE MACHINE LEARNING FRAMEWORK

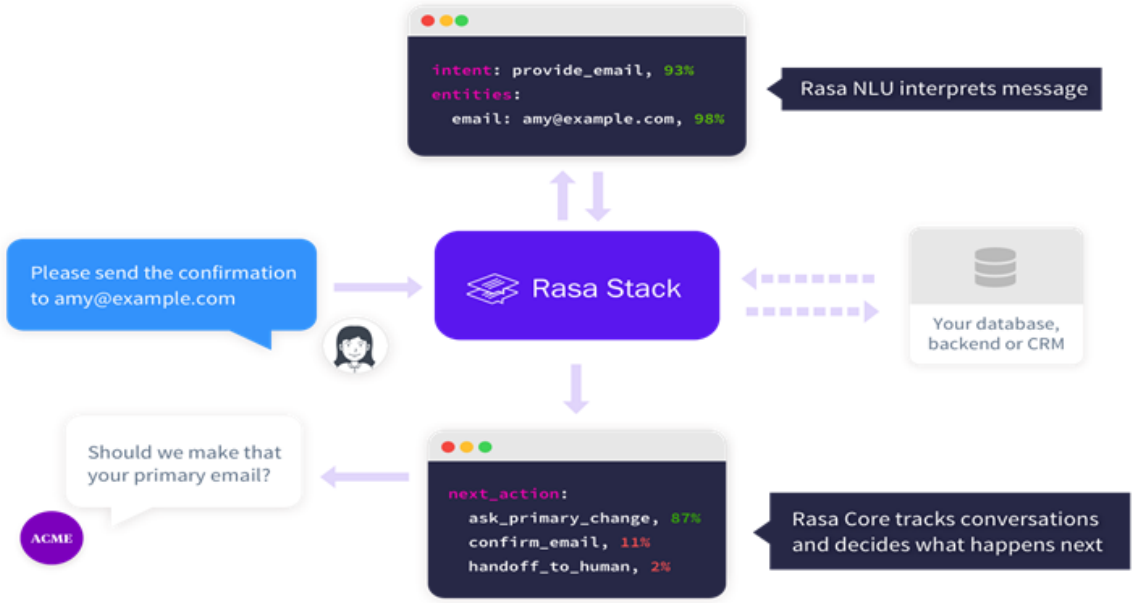
Rasa Open Source is a machine learning framework for automated text and voice-based conversations. Rasa has two main modules: Rasa NLU and Rasa Core.

Rasa NLU understands the user's message based on the previous training provided:

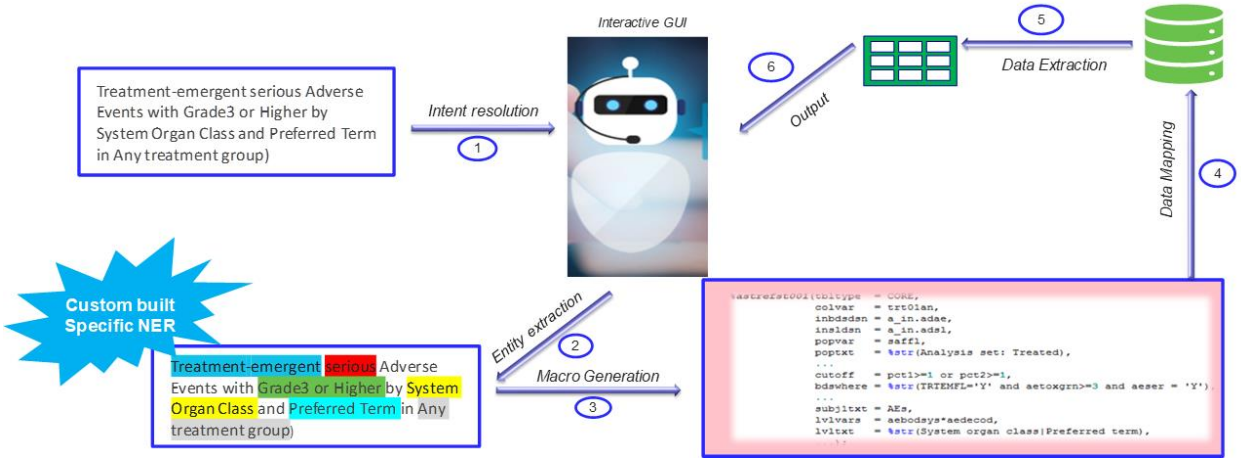
1. Intent classification: Interpreting meaning based on predefined intents (Example: "Please send the confirmation to amy@example.com" is a send confirmation intent with 93% confidence).
2. Entity extraction: Recognizing structured data (Example: amy@example.com is an email).

Rasa Core decides what happens next in this conversation. Its machine learning-based dialogue management predicts the next best action based on the input from NLU, the conversation history, and your training data

Rasa X is a tool that helps you build, improve, and deploy AI Assistants that are powered by the Rasa framework. Rasa X includes a user interface and a REST API.



We can utilize Rasa framework to train a virtual assistant for TLG generation in the three stages tasks. Here is the Data Flow for the solution.



For the TLG title, we need to understand the intent of the specification and also extract necessary key words/phrases and map to right macro, with the right parameter value for the right parameter.

Intent classification is to map to the right macro, and it is relatively easy as the standard text classification model can easily distinguish the AE table, DM table, LAB table etc. However, the Entity extraction, which is used to map the right parameter value for the right parameter, is the difficult part as we have too many different parameters, variables and values used in the macro call. Here we will focus on the Entity extraction part.

ENTITY EXTRACTION

Taking the AE tables below for example, it can be produced by one standard macro %ASTREFST001, however we have 3 parameters with great flexibility which usually involves lots of human efforts to fill in. We will introduce 3 different approaches to automatically extract the entities for each parameter.

Text	CUTOFF	BYVARS	WHERE
Incidence of Treatment-emergent Adverse Events Occurring in 10% or More Subjects by System Organ Class and	pct5>=10	aebodsys*aedecod	%str(TRTEMFL='Y')
Incidence of Treatment-emergent Adverse Events Occurring in 10% or More Subjects by Preferred Term	pct5>=10	aedecod	%str(TRTEMFL='Y')
Incidence of Treatment-emergent Adverse Events Occurring in 5% or More Subjects by System Organ Class and	pct6>=5	aebodsys*aedecod	%str(TRTEMFL='Y')
Incidence of Treatment-emergent Serious Adverse Events Occurring in 2% or More Subjects by Preferred Term	pct6>=2	aedecod	%str(TRTEMFL='Y' and AESER='Y')
Incidence of Grade 3 or 4 Treatment-emergent Adverse Events by Preferred Term Occurring in 2% or More Subjects	pct6>=2	aedecod	%str(TRTEMFL='Y' and (AETOXGR='3' or AETOXGR='4'))
Number of Subjects With Treatment-emergent Adverse Events With Frequency of at Least 5% in Any Treatment Group	pct1 ge 5 or pct2 ge 5	aebodsys*aedecod	%str(TRTEMFL='Y')
Number of Subjects With Adverse Events With Frequency of at Least 5% in Any Treatment Group by System Organ	pct1 ge 5 or pct2 ge 5	aebodsys*aedecod	%str(fupfl='Y')
Treatment-emergent Severe Adverse Events in at Least 1% of Subjects in Any Treatment Group	%str(pct1 ge 1 or pct2 ge 1)	aebodsys*aedecod	%str(saffl = "Y" and trtemfl = "Y" and aesev='SEVERE')
Treatment-emergent Adverse Events with Grade 3 or Higher by System Organ Class and Preferred Term (at Least 1	pct1>=1 or pct2>=1	aebodsys*aedecod	%str(TRTEMFL='Y' and aetoxgrn>=3)
Treatment-emergent Serious Adverse Events with Grade 3 or Higher by System Organ Class and Preferred Term (at	pct1>=1 or pct2>=1	aebodsys*aedecod	%str(TRTEMFL='Y' and aetoxgrn>=3 and aeser = 'Y')
Drug-related Treatment-emergent Adverse Events with Grade 3 or Higher by System Organ Class and Preferred Term	pct1>=1 or pct2>=1	aebodsys*aedecod	%str(TRTEMFL='Y' and aerel in ('POSSIBLE','PROBABLE','VERY LIKELY') and aetoxgrn>=3)

1) Text rule-based approach

E.g., for CUTOFF, the scenarios are limited, and text feature is obvious, and we can use rule-based regular expression to extract it.

```
re.findall(r'(\d+\s*(percent|pct|%))', title, flags=re.IGNORECASE)
```

2) Linguistic rule-based approach

For BYVARS, the text feature varies a lot, we can hardly write regular expressions to cover many different words, but the syntax features are relatively fixed, so we can use the linguistic rules to exact phrase matches.

```
import spacy
from spacy.pipeline import EntityRuler
nlp = spacy.load("en_core_sci_md", disable = ['ner'])
nlp.add_pipe(merge_noun_chunks)
nlp.add_pipe(merge_entities)

bygroup_pattern =[
                 {"LOWER":"by"}
                  ,{"POS": "{}", "OP": "*"}
                  ,{'POS':{'NOT_IN': ['ADP']}, "OP": "*"}
                  ,{'POS':{'NOT_IN': ['ADP','VERB','PART','CCONJ']}}
                 ]

patterns = [ {"label": "byGroup", "pattern":bygroup_pattern}]

ruler = EntityRuler(nlp, overwrite_ents=True)
ruler.add_patterns( patterns)


nlp.add_pipe(ruler)

_test = """Number of Patients with 1 or More Treatment-Emergent Adverse Events by Age group
and by MedDRA System Organ Class and Preferred Term"""
doc = nlp(_test)
displacy.render(doc, style="ent")

allbyEnt = []
for ent in doc.ents:
    allbyEnt.append(ent.text)
```

Number of Patients with 1 or More Treatment-Emergent Adverse Events by Age group BYGROUP and by MedDRA System Organ Class and Preferred Term BYGROUP

After we extract the short texts for by groups, we can further extract each by-group phrases similarly.

```
np_pattern = [{"TAG": {'IN': ['NN','NNP','NNS','NNPS']}, "OP": "+"}]

patterns = [ {"label": "NounPhrase", "pattern":np_pattern}]
```

```
allNpEnt = []

for i,test in enumerate(allbyEnt):
    doc = nlp(test)
    displacy.render(doc, style="ent")
    for ent in doc.ents:
        allNpEnt.append(ent.text)
```

by Age group NOUNPHRASE

by MedDRA System Organ Class NOUNPHRASE and Preferred Term NOUNPHRASE

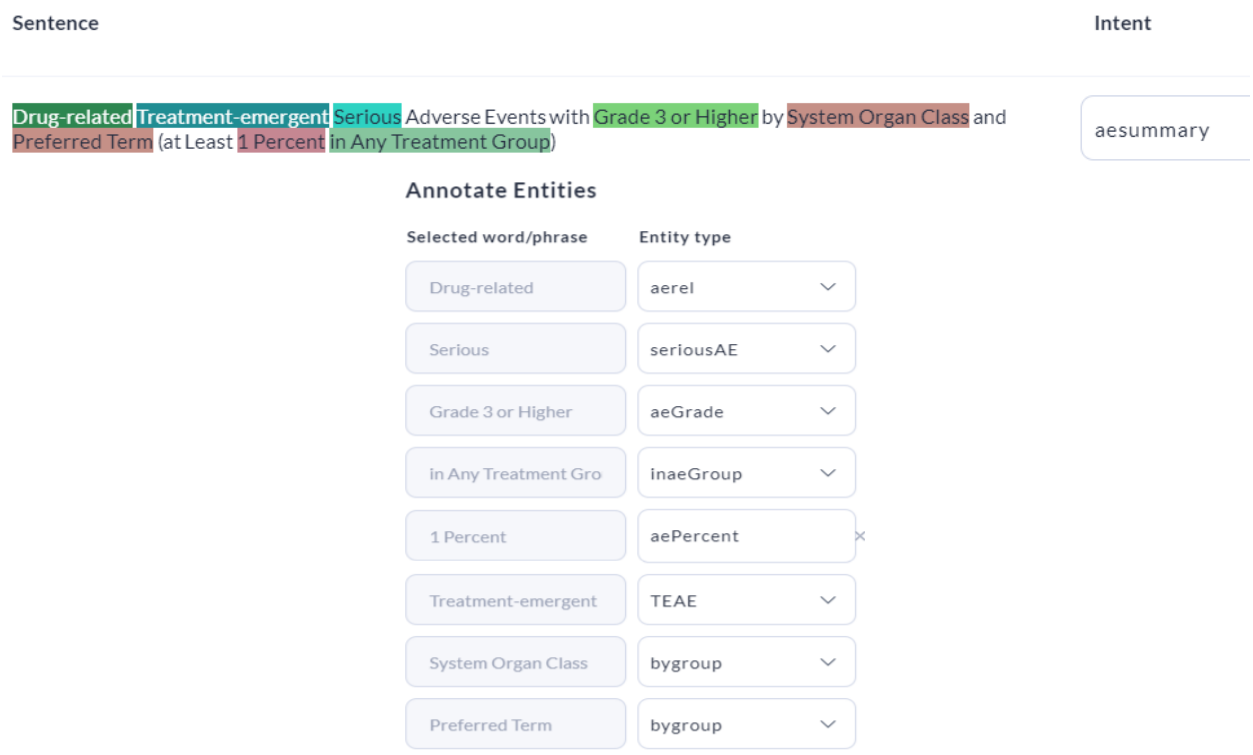
Similarly, we can write rules for other entities, like "AE Leading to xxxx" and "xxx Related AE", etc.
This approach can handle most scenarios if the TLG titles follows the standard template well, however there are always exceptions for some new studies across different TA and compounds. The manual-written rules eventually will become hard to maintain, and we need the next level of approach - data driven approach.

3) Machine learning approach

The subset where condition can be very complex as lots of variables and values can be involved in the real cases. Instead of manually writing complex rules to try to cover different scenarios, machine learning models, often referred to as data-driven models, can learn the rule from the historical data and predict the result for future scenarios.

We need to collect lots of data and teach machine to learn how to recognize these entities – first we need to do the data labelling, then use machine learning to learn the rules from the data.

RASA-x provides good UI for data labelling, and we can mark all these key pieces and map them into different pre-defined entities. This is labor-intensive as we need 'big' data - the more training data connected and labeled, the smarter the system can learn the patterns and recognize both historical scenarios and unseen scenarios.



We can utilize some rule-based algorithm to draft the data labeling, then update the training data with some manual review and minor modification, which will greatly reduce the manual data labeling work.

RASA provides CRFEntityExtractor - conditional random field model from machine learning library sklearn-crfsuite for training custom entities. We also have custom built specific Named Entity Recognition neural models.

Custom built specific NER neural models:

Named-entity recognition (NER) is a sub-task of information extraction that seeks to locate and classify named entities in text into pre-defined categories such as locations, expressions of times, quantities etc. However just having the entities identified may not always generate optimal results and keeping the context in mind is an important and integral part of various NLP tasks. In NER specifically, having knowledge of context is important. Therefore, a custom NER engine leveraging BERT based word embeddings was created with biomedical language specific to this domain to extract the relevant entities from the user query and identify the standard Table title based on those entities.

We will now presents a design of a machine-learning system that, if trained well, is able to process natural language verbatim information (NLP Neural Models).

Choice of the NLP Neural model was be governed by the following 4 capabilities:

1. Detect specific medical entities (listed below)
2. Detect causality phrases
3. Industry grade performance
4. Customization capability

Entities that need detection:

- Adverse events: e.g. rashes, swelling
- Serious events: e.g. hospitalization, disability/incapacity
- Important Medical Events (IME): e.g. pregnancy, breast-feeding period
- Organs: e.g. face, arms, skin
- Drug name
- Dose: e.g. 3 capsules, one finger-tip of cream
- Dose administration routes: e.g. oral
- Gender: e.g. male
- Age group: e.g. baby, child, grand-parent
- Date: Absolute e.g. 2nd Dec. or relative dates or periods e.g. 2 weeks ago
- Time: Times smaller than a day, e.g. ‘every 3 hours’
- Quantity and associated units: e.g. 2 spoons of syrup, 10 mg dose
- Ordinal: e.g. first dose, this is the third occurrence

Performance:

- High accuracy and in particular high Recall
- NER models need to load large language models and process large volume of records
- Must have efficient storage techniques and memory management
- Must be suitable for parallel processing
- Must be trainable and tunable based on custom needs

Core models: General-purpose pre-trained models to predict named entities, part-of-speech tags and syntactic dependencies. These can be used out-of-the-box and fine-tuned on more specific data.

Pre-trained starter models: Transfer learning based starting base models like **BERT**
In this section we understand the foundation of word-embeddings so as to understand how they assist in better prediction. Word embedding techniques captures the semantics of words (and their similarities) based on their surrounding words.

In 2003 Bengio et al. presented the 'Neural Probabilistic Language Model'.
A statistical model of language can be represented by the conditional probability of the next word given all the previous ones, since

$$\hat{P}(w_1^T) = \prod_{t=1}^{T} \hat{P}(w_t | w_1^{t-1})$$

where *w*t is the *t*th word
Language models take advantage of word order, and the fact that temporally closer words in the word sequence are statistically more dependent. This also helps incorporate context-sensitiveness to some extent. n-gram models construct tables of conditional probabilities for the next word, for each context i.e. combination of the last n−1 words.

1. The approach associates each word in the vocabulary a word feature real-value vector $\in \mathbb{R}^m$
2. Express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence.
3. Learn simultaneously the word feature vectors (embeddings) and the parameters of that probability function

The following two examples, that are essentially similar in context, show how the sequence of words can help establish a causality between product ('drug') and AE ('rash').

I experienced a rash after consuming the drug
On swallowing the drug, I experienced a rash

Note that the algorithm need not necessarily predict a word based on last *n-1* words, a last *p* words and next *q* words strategy can be equally effective.

A simple explanation of how word-embeddings are learnt is shown below. We assume the word 'applying' is being attempted to be predicted in this sentence.

I	experienced	a	rash	after	consuming	the	drug
4343	19665	1	4852	9665	[predict]	340	2123

The real numbers represent the vector representations. These are fixed values assigned to all words in the vocabulary. The first word of the vocabulary 'a' for example, could be represented by '1'.

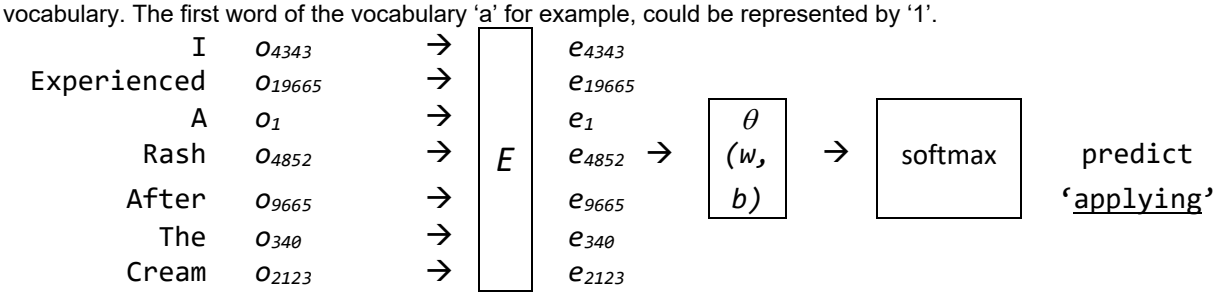


Figure 2: Word-embeddings

Here o_{4852}, represents a one-hot coded vector for that word i.e. 'rash'. A matrix of parameters *E* is used and the product *E* times o_{4852} provides the embedding vector e_{4852}.
The neural network is then trained with examples that contain such sentence constructions. This helps the network learn *simultaneously* the parameters *E*, *w* and *b* of the probability function.
Note that the same matrix *E* is used for *all* the words. Also the matrix is the same for different *positions* off the preceding *n* words. It is this that allows generalization for similar constructs ('I experienced a rash', 'been experiencing a rash')
Using probabilistic modelling allows to learn similar word embeddings for the AE 'rash' and say 'redness'. This allows prediction of the word 'rash' and 'acne' as the network sees similar examples

Training is achieved by finding θ that maximizes the penalized log-likelihood for the entire *training corpus*:

$$L = \frac{1}{T}\sum_{t} \log f(w_t, w_{t-1}, \cdots, w_{t-n+1}; \theta) + R(\theta)$$

ENTITY MAPPING

After extracting the key words/phrases for entities, we can build some rules to map to variables or statements. We can also do smart match to map to the variables and values.

We can run metadata for the source data, and create a dictionary for each variable, and based on the label, we can do fuzz match to connect the key words/phrases to variable names, even when there are some minor typos in the words/phrases.

name	label
AEBODSYS	System Organ Class
AEDECOD	Preferred Term
AGEGRP	Age Group

```
# visit https://github.com/seatgeek/fuzzywuzzy for more details
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
```

```
#standard variable name and label
self.vars_data=pd.read_csv('./data/variables.csv')
stdlabel = self.vars_data['label'].values.tolist()

for byvar, label in vardic.items():
    if label is not None:
      # use process.extractOne to get the closest matche
      matched_label, score = process.extractOne(label, stdlabel, scorer=fuzz.token_set_ratio)
      matched_row = self.vars_data.loc[self.vars_data['label'] == matched_label,]

      matched_varname = matched_row['name'].values[0]
      #update dic with standard varname and label in list
      vardic[byvar] = [matched_varname,matched_label]
```

There are hundreds of variables from ADaM standards, and it is hard to do data labeling for each variable, so one better approach is that we can label the data by variable group, as they share similar wordings, and then map to correct variable by further smart fuzzy match.

Count of Variable Name Variable Group	Data Structure Name Basic Data Structure	Subject Level Analysis Dataset	Grand Total
Analysis Descriptor Variables	1		1
Analysis Parameter Variables	23		23
Analysis Visit Windowing Variables	6		6
Data Point Traceability Variables	3		3
Flag Variables	30		30
Lab Related Analysis Variables	8		8
Population Indicator(s)		7	7
Study Identifiers		6	6
Subject Demographics		6	6
Subject Identifier Variable	4		4
Time to Event Variables	3		3
Timing Variables	56		56
Treatment Variables	8	13	21
Trial Dates		31	31
Grand Total	**142**	**63**	**205**

CONCLUSION

We have designed and explored virtual assistant system to optimize report generation process with limited human intervention leveraging an enhanced RASA open source machine learning framework with Named Entity Recognition Neural models. We combined both rule-based approach, linguistic features in NLP and machine learning models for the Named Entity Recognition task and map the extracted entities to macro codes using smart match approach.

The benefits of virtual assistant system for TLG generation are:

- Automatic Spec understanding and code generation to reduce manual work and potential human error which inevitably leads to wasted time in both coding and debugging
- Significant reduction in time to submission and increase in consistency and quality

- Accumulate and maintain internal knowledge in database and system
- Leave programmers more time for more complex tasks, in depth disease area understanding, more time for broadening skillsets

There are also some challenges to further improve the system for more automation:

- High-quality training data is the key factors for the success and more training data need to be collected and labeled
- More standard reporting macros instead of ad-hoc codes need to be applied
- Analysis data should meet the one-proc-away requirement for output generation

We are only just scratching the surface when it comes to the uses of AI and Machine Learning in the Pharmaceutical industry. However, even at this early stage, the technologies are proving to be tremendously promising when it comes to giving new mechanistic insights to disease and thereby helping to identify promising targets.

The technology will also help in terms of the industry's selection of patients for clinical trials and enable companies to identify any issues with compounds much earlier when it comes to efficacy and safety. So, the industry has much to gain by adopting AI and Machine Learning approaches. It can be used to good effect to build a strong, sustainable pipeline of new medicines[5].

REFERENCES

1. Metadata-based Auto-Programming Process https://www.lexjansen.com/phuse-us/2018/si/SI10.pdf
2. Metadata-based Auto-Programming Process – Part 2: Map human language to SAS® code by Natural Language Processing (NLP) https://www.lexjansen.com/phuse-us/2019/ml/ML05.pdf
3. Gartner, Augmented Analytics Is the Future of Data and Analytics, Rita L. Sallam, Cindi Howson, Carlie J. Idoine, 27 July 2017
4. https://www.sas.com/en_us/insights/analytics/what-is-natural-language-processing-nlp.html
5. https://www.drugtargetreview.com/article/15400/artificial-intelligence-drug-discovery
6. https://github.com/seatgeek/fuzzywuzzy
7. https://github.com/chartbeat-labs/textacy
8. https://spacy.io
9. https://www.nltk.org
10. https://rasa.com
11. https://towardsdatascience.com/create-chatbot-using-rasa-part-1-67f68e89ddad
12. http://www.cs.cornell.edu/~kilian/papers/wmd_metric.pdf
13. https://spacy.io/models/en#en_vectors_web_lg
14. https://en.wikipedia.org/wiki/Levenshtein_distance
15. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova Google AI Language {jacobdevlin,mingweichang,kentonl,kristout}@google.com

CONTACT INFORMATION

Special thanks to the Janssen Autocode team. Your comments and questions are valued and encouraged.

Contact the authors at:

Sumesh Kalappurakal
Janssen R&D
920 Route 202 S, Raritan
New Jersey - 08869 USA
Email: SKalappu@its.jnj.com

Harry Chen
Janssen China R&D
65 Gui Qing Lu
Shanghai 200231 China
Email: hchen104@its.jnj.com

Srikanth Ramakrishnan
Janssen R&D
920 Route 202 S, Raritan
New Jersey - 08869 USA

Shobha Rani
Janssen R&D
920 Route 202 S, Raritan
New Jersey - 08869 USA

Email: sramak22@its.jnj.com Email: srani1@its.jnj.com