The Impact of Data Challenges on Intent Detection and Slot Filling for the Home Assistant Scenario

Anda Stoica[1], Tibor Kadar[2], Camelia Lemnaru[3], Rodica Potolea[4] and Mihaela Dînșoreanu[5]
Department of Computer Science, Technical University of Cluj-Napoca, Romania
Email: [1]andadiana.stoica@gmail.com, [2]kadartibor24@gmail.com, [3]camelia.lemnaru@cs.utcluj.ro,
[4]rodica.potolea@cs.utcluj.ro, [5]mihaela.dinsoreanu@cs.utcluj.ro

***Abstract*—Natural Language Understanding (NLU) is currently a very high-interest domain to both academia and the commercial environment, due in the largest part to the recent increased popularity of conversational systems. In this paper we focus on the home assistant application context and identify a set of language and data-related challenges that can occur in such a scenario, such as: distribution shift, missing information and class imbalance. We systematically generate datasets in the Romanian language that model these data complexities and further investigate how well two of the most prominent tools – Wit.ai and Rasa NLU – solve the tasks of intent detection and slot filling, given the considered data complexities. We perform a thorough analysis of the errors produced by both tools, and provide the most probable justification for their occurrence. We found that both tools focus extensively on the verb for identifying intents, and that antonyms, class-imbalance and certain small variations in formulation greatly impact intent and slot identification. This opens new research directions to directly address these shortcomings.**

I. Introduction and Problem Statement

Natural Language Understanding (NLU) is currently an active research domain, equally driven perhaps by the traditional, academic focus on the topic, but also by a more recent increased commercial interest, with existing applications in conversational systems [1], machine translation [2], text categorization [3], or large-scale content analysis [4] (to name but a few). Such systems generally make a trade-off between the breadth and the depth of the understanding, according to the application domain, specific objectives, the challenges associated and how available resources (for that language and domain) can help tackle them.

Intent detection/classification and slot filling are key NLU tasks in developing (domain specific) conversational systems, such as personal AI assistants (e.g. Apple Siri, Google Now, Microsoft Cortana), smart home assistants (e.g. Google Home, Amazon Echo) or even humanoid conversational robots (e.g. SOTA, Pepper, Erica).

Projected onto the home assistant application scenario, the intent detection and slot filling task requires addressing a limited breadth of understanding and not necessarily too much depth, because in such an application scenario we normally have to recognize a (relatively) small set of clearly formulated commands, together with their parameters. However, even in such an apparently "easy" scenario, challenges which increase the difficulty of the problem might arise, such as the lack of natural language resources for the specific language, or particular data-related complexities, such as missing information, imbalance, or distribution shift (e.g. the model was trained to recognize intents formulated in a specific manner and using a certain vocabulary, but it has to be deployed in a new home, with a potentially different vocabulary and discourse style).

This paper aims to investigate specifically how challenges as the ones mentioned above affect the quality of the results produced by existing resources and tools for intent classification and slot filling, in the smart home assistant scenario. We motivate the likelihood of such phenomena to appear in this scenario and generate several datasets which model these data complexities. We focus our analysis on two of the most prominent existing tools for the task - Wit.ai [5] and Rasa NLU [6] and perform both quantitative and qualitative assessments on the performance of the models produced by the two.

A. Intent Detection and Slot Filling Problem

In the context of a home assistant scenario, extracting meaning from text comes down to two key tasks used in Natural Language Understanding systems: intent detection and slot filling. Intent detection can be defined as a classification problem, which takes the utterance as a whole and tries to label it as one of a predefined set of intents. These intents would coincide with the actions that the home assistant is capable of performing, like turning the lights on/off, adjusting the temperature, playing music, etc. Slot filling can be viewed as a sequence labelling problem, which takes each word in the utterance and assigns it to a slot type. Therefore, if intents are the actions, slots are the parameters of those actions, like the room in which to turn the lights on, or the song to play. A concrete example in Romanian can be seen in figure 1

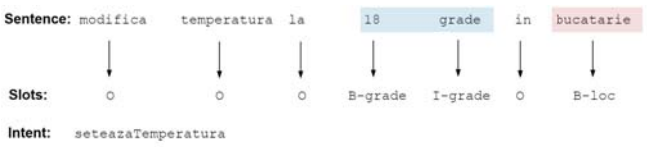


Fig. 1. Example utterance with the intent and slots labelled in IOB format [7]. The English translation of the sentence is "Change the temperature to 18 degrees in the kitchen".

978-1-7281-4914-1/19/$31.00 ©2019 IEEE

The rest of the paper is organized as follows: section II discusses several existing approaches for extracting meaning from text. Section III formulates the hypotheses we focus on and details the data generation process. In section IV we describe the evaluation methodology and analyze the results obtained, both quantitatively and qualitatively. Section V presents a discussion on how carefully engineering the training set might impact the performance and section VI concludes the paper.

II. Extracting Meaning from Text

Text understanding can be performed at different levels of depth, resulting in a variety of different tasks. At one end of the spectrum lie the semantic parsing techniques, which attempt to extract meaning from text, by transforming it into a structured representation (such as AMR [8], UCCA [9], or Universal Dependencies [10]), while at the other end we might find various text classification techniques, such as topic modelling, for example (which, though not as deep, is an extremely challenging task also). Intent detection (or classification) can be viewed as a lighter alternative for extracting meaning from text, since the output of the classification is not structured, and the focus is on correctly identifying the action/intent, not the entire elements in the sentence.

There are two main strategies in literature for solving intent detection and slot filling. The first one is to treat each task separately, while the second builds a model that can handle both tasks jointly. Recent solutions based on deep neural networks do not require feature engineering in order to exploit language properties, but they shift the focus on building architectures that capture relevant relationships in the data, such as the Deep Belief Network proposed in [11]. Capsule neural network architectures have been recently introduced as a solution to this task also, because of their inherent potential to support zero-shot learning capability [12]. On the other hand, slot filling can be approached independently with widely used sequence labelling solutions such as Conditional Random Fields (CRF) [13] and Recurrent Neural Networks (RNN) [14].

More recent solutions adopt a joint approach to intent detection and slot filling, with the added benefit that each task contributes to the other, ultimately producing better overall results. A joint model based on a Convolutional Neural Network (CNN) architecture is introduced in [15], where the features extracted by the CNN are used for both intent detection and slot filling. Recurrent neural network (RNN) models were also employed, for example in [16], where the authors propose an encoder-decoder architecture with an added attention mechanism. Additionally, capsule neural networks have also been explored in this joint learning setting in [17], where such an architecture tries to exploit the advantage of the hierarchical relationships between words, slots and intents.

III. Analysis of the Home Assistant Scenario

We set off in our analysis by building a set of intents for the home assistant problem, covering three different areas: lights, temperature and media center control commands. Thirteen intents were defined (see Table I for details - we provide the English intent name for reference). These intents and slots are not an exhaustive set. Next, we identified a set of potential challenges that are likely to appear in such an application context, which we enumerate below:

TABLE I
Intents defined for the home assistant scenario

Intent Name (Romanian)	Intent Name (English)	Slots	Category
AprindeLumina	TurnOnLight	Room	Light
StingeLumina	TurnOffLight	Room	Light
ScadeIntensitate Lumina	DecreaseIntensity Light	Room Level	Light
CresteIntensitate Lumina	IncreaseIntensity Light	Room Level	Light
CresteTemp	IncreaseTemp	Room Level	Temperature
ScadeTemp	DecreaseTemp	Room Level	Temperature
SeteazaTemp	SetTemp	Room Level	Temperature
PuneMuzica	PlayMusic	Artist	Media
OpresteMuzica	StopMusic	–	Media
SchimbaVolum	ChangeVolume	Level	Media
PornesteTV	StartTV	Channel	Media
OpresteTV	StopTV	–	Media
SchimbaCanal	ChangeChannel	Channel	Media

1) First, and foremost, starting from the assumption that the same meaning can be conveyed via various language constructs, we wanted to evaluate how well the tools we considered deal with synonymy. The hypothesis we wanted to test was that word embeddings as feature vectors successfully address this challenge.
2) Secondly, some slot values may be absent from the training data, or – for some intents – the slot might be missing altogether (E.g. ”Aprinde, te rog, lumina.” – en. ”Turn the lights on, please.” - no room specified). In such situations, the conversational model should compensate for the missing data by inferring the information from other sources (e.g. the user’s location). From the intent classification and slot filling point of view we wanted to investigate whether the lack of slot information, or the appearance of new slot values affect classification accuracy.
3) Thirdly, class imbalance is known to affect the learning ability of classifiers [18]–[20]. For the intent detection and slot filling task, the phenomenon is generated by emerging intents. In the home assistant scenario, this appears because more and more items in the homes become automated, and since manual labeling is generally costly and/or impractical most of the time (in an already deployed system), we have to prepare for detecting new intents with new slots and/or values [12].

A. Data Generation

Having defined the data complexities we want to investigate, we generated datasets corresponding to the three different challenges, and a baseline dataset - without any data complexity added to it. The datasets for the three complexities have been generated incrementally - i.e. the data for the third scenario contains all three challenges in it - synonymy, missing slot values and class imbalance. All datasets have been generated automatically by Chatito [21], using a set of handcrafted rules and associated vocabulary. The Chatito scripts for generating the data and the datasets are available on our GitHub repository[1]. A quantitative description of the datasets is presented in Table II.

1) Baseline scenario: The purpose of the baseline scenario is to evaluate how well existing solutions deal with classifying intents and identifying slot values from Romanian utterances (the language challenge). For this, we generated randomly, drawing from the same uniform distribution and using the same rules and vocabulary for both training and testing datasets, 90 training and 30 testing instances for each of the 13 intents.

2) Scenario 1 - synonyms: For this scenario, we focused on employing different formulations for specifying the intent (i.e. the verb identifying the intent in the sentence) in the training and testing data. The simplest form to obtain this result with Chatito is to provide different vocabularies in the train and the test splits for the entity which specifies the action part of a rule or the slot value (e.g. for temperature value, use "17" in the training set and "saptesprezece" in the test set), or to provide different rules for building an utterance conveying the same intent information. For this scenario, we generated a training set having 1018 training and 390 test instances (for some simple intents there was not enough variation in the rules and the vocabulary in the training set to allow for 90 instances to be generated).

3) Scenario 2 - missing slots: Only eleven out of the thirteen intents have slots, and we considered that the *ChangeVolume* intent must always have the level information associated. Therefore, only ten intents were considered for this scenario. For some slots, we partitioned the possible values between the train and the test sets, and also added new rules in the test set which generated utterances without slot information. Consequently, we obtained 813 instances for the train set and 300 instances for the test set.

4) Scenario 3 - class imbalance: For scenarios 1 and 2 we tried to keep a balanced class distribution in both training and test datasets. The third scenario makes changes specifically to this data characteristic: we under-represented each category of intents in turn, resulting in three different train/test datasets (three different sub-scenarios), which were considered independently. For each intent in a category, we normally reduced the number of intents in the training set by a factor of 10. The test sets maintained the same class distribution as before.

[1]https://github.com/eaaskt/nlu/tree/master/benchmark

IV. Empirical Evaluations

Having defined the problem and the various challenges that may arise, we moved on to implementing a pipeline which performs intent detection and slot filling. Given an input utterance $x = \{x_1, x_2, ..., x_n\}$ of n words, we want to obtain an intent i, where $i \in \{\textit{TurnOnLights}, \textit{TurnOffLights}, \ldots, \textit{ChangeChannel}\}$ and the sequence of slots $s = \{s_1, s_2, \ldots, s_n\}$, where each word x_k has an associated slot $s_k \in \{O, \textit{B-room}, \textit{I-room}, \ldots, \textit{B-channel}, \textit{I-channel}\}$.

From the pol of existing tools for solving these tasks, we decided to choose a closed source solution – Wit.ai [5] and an open source one – Rasa NLU [6].

Wit.ai is the NLU tool developed at Facebook. In addition to being closed source, the platform does not offer many configuration options (besides the choice of language), so the only way we can influence the quality of the output is by data manipulation. Wit.ai supports Romanian out of the box and is improving continuously as more and more developers create applications and gather training data.

On the opposite side of the spectrum, Rasa NLU is open source and allows the user to customize and configure the processing pipeline. The two main configurations offered by Rasa are: *pretrained_embeddings_spacy* and *supervised_embeddings*. The former uses SpaCy [2], whose language models are packaged with pre-trained word embeddings, while the latter trains the word vectors on the input dataset, making it language-independent. Since there was no language model for Romanian available in SpaCy, we had to extend it and build our own, in two separate versions: one without pre-trained word embeddings, but still containing context-sensitive token vectors inferred by SpaCy, and one with FastText[3] pre-trained word vectors. For our experiments, we trained 3 types of models for each scenario. The first two were trained using the *pretrained_embeddings_spacy* pipeline, one model for each of the two SpaCy language model versions: SpaCyBase and SpaCyFastText. The third model, SupervisedEmbeddings, was trained with the *supervised_embeddings* pipeline.

A. Quantitative Analysis

The overall quantitative results obtained by Wit.ai and Rasa for both intent detection and slot filling can be found in Table III and Table IV, respectively.

The SpaCyFastText model seems to be the one consistently outperforming the others in intent classification. For slot filling, SpaCyBase and SpaCyFastText have the same performance because the *pretrained_embeddings_spacy* pipeline uses CRF for slot labelling, without making use of the word vectors. The SupervisedEmbeddings model does a slightly better job with the slots in scenarios 1 and 3.2. Wit.ai is the worst performing for intent classification. However, for the slot filling task, Wit.ai performs better in all scenarios except the baseline and scenario 3.1 by a very small margin.

[2]https://spacy.io/
[3]https://fasttext.cc/

TABLE II
QUANTITATIVE DESCRIPTION OF THE DATASETS - NUMBER OF TRAIN/TEST INSTANCES PER INTENT, FOR ALL SCENARIOS

Intent Name	Baseline	Scenario 1	Scenario 2	Scenario 3.1	Scenario 3.2	Scenario 3.3
AprindeLumina	90/30	90/30	63/30	6/30	63/30	63/30
StingeLumina	90/30	60/30	30/30	3/30	30/30	30/30
ScadeIntensitate Lumina	90/30	90/30	90/30	9/30	90/30	90/30
CresteIntensitate Lumina	90/30	90/30	90/30	9/30	90/30	90/30
CresteTemp	90/30	90/30	90/30	90/30	9/30	90/30
ScadeTemp	90/30	90/30	90/30	90/30	9/30	90/30
SeteazaTemp	90/30	90/30	90/30	90/30	9/30	90/30
PuneMuzica	90/30	90/30	90/30	90/30	90/30	9/30
OpresteMuzica	90/30	23/30	–	23/30	23/30	3/30
SchimbaVolum	90/30	90/30	–	90/30	90/30	9/30
PornesteTV	90/30	90/30	90/30	90/30	90/30	9/30
OpresteTV	90/30	35/30	–	35/30	90/30	3/30
SchimbaCanal	90/30	90/30	90/30	90/30	90/30	9/30
TOTAL	1170/390	1018/390	813/300	715/390	715/390	585/390

TABLE III
WIT.AI RESULTS

Scenario	Intent F1 score	Slot F1 score
baseline	100	99.10
1	39.74	86.97
2	38.66	76.17
3.1	29.48	73.01
3.2	29.74	75.85
3.3	27.17	79.18

TABLE IV
RASA NLU RESULTS

Model	Scenario	Intent F1 Score	Slot F1 Score
SpaCyBase	baseline	99.23	100
	1	43.59	84.49
	2	33.67	74.14
	3.1	32.56	73.59
	3.2	36.15	74.44
	3.3	30.26	72.55
SpaCyFastText	baseline	100	100
	1	55.38	84.49
	2	45.67	74.14
	3.1	44.87	73.59
	3.2	45.38	74.44
	3.3	40.26	72.55
SupervisedEmbeddings	baseline	100	100
	1	43.08	85.55
	2	34.00	68.72
	3.1	41.03	71.04
	3.2	33.33	75.07
	3.3	33.08	72.55

B. Qualitative Analysis

By inspecting the errors made by the different models, we were able to identify several consistent patterns, which are detailed below. A note should be made on the fact that these errors are present in the models trained with both tools, but they were not the only types of errors made. However, considering their prevalence, solving them would most likely improve the behaviour of the models. Since the scenarios are built incrementally, the errors propagate from one scenario to the next; we present the identified error patterns in the first scenario in which they appear.

- Starting with Scenario 1, we observed that the models are struggling to tell the difference between intents which have opposite meanings (such as *TurnOnLight* and *TurnOffLight* or *TurnOnTV* and *TurnOffTV*). This behavior seems to be more prominent in the SPACYFASTTEXT model, where most of the *TurnOffLight* examples are classified as *TurnOnLight*, but not vice versa. Not only opposite intents are affected by this, *ChangeChannel* is also almost exclusively labelled as *StartTV* in the SPACYFASTTEXT model, while *StartTV* is classified as *ChangeChannel* about half of the time. The models seem to confuse words with different meanings but which are commonly used in the same context. An example for such an error is for the sentence *"As vrea sa opresti becurile in camera de zi"* (*"I would like you to turn off the lights in the living room"*), whose true intent is *StingeLumina* (*TurnOffLight*) but instead it is classified as *AprindeLumina* (*TurnOnLight*).
  Table V shows the cosine similarities between some pairs of words chosen from the dataset: antonyms/words with different meanings and synonyms. It can be seen that in many cases, there is a higher similarity between words which have different meanings, such as "aprinde" and "stinge" ("turn on" and "turn off") or "aprinde" and "schimba" ("turn on" and "change"), than between synonyms, such as "aprinde" and "porneste" ("turn on" and "start"). These values seem to explain this identified error pattern.
- Along the same lines, intents which have similar wording but in which only the object differs (e.g. "turn off the

light" and "turn off the song" – in Romanian "opreste lumina" and "opreste muzica") are sometimes mixed up. The similarities between object synonyms and objects belonging to different classes (such as "song", "melody" or "lights") are consistent with their meaning, so this error pattern has a different source than the previous one. An example would be the user command *"Stinge TV"* (*"Turn off the TV"*), labelled as *StingeLumina* (*TurnOffLight* instead of *OpresteTV* (*StopTV*). This indicates that the intent classification is relying heavily on the verb present in the sentence and uses it to indicate the overall intent. In this example, the intent is very likely misidentified because in the training set the verb *"stinge"* (*"turn off"*) was used in sentences having the intent *StingeLumina* (*TurnOffLight*).

- Starting with scenario 1, the model doesn't accurately label number slots when the number values are written in text instead of using digits. For example, even though the model has seen *"26"* in the training set, it doesn't label or only partially labels *"douazeci si sase"* (*"twenty-six"*). For the sentence *"Hey, modifica temperatura aproximativ la douazeci si cinci de grade in toata casa"* (*"Hey, change the temperature approximately to twenty-five degrees in the whole house"*), the words *"douazeci si cinci de grade"* (*"twenty-five degrees"*) should have been identified as belonging to the slot type *Level*, but instead only *"de grade"* (*"degrees"*) were attributed to that slot type, the rest being considered as not belonging to any slot.
- In scenario 2, where the missing slots problem is tested, locations aren't labelled if they appear in the middle of the sentence, yet the same value for location is detected if it appears at the very end of the utterance. For instance, in the sentence *"Buna Pepper, poti sa pornesti becul in camara"* (*"Hello Pepper, can you turn on the light in the pantry"*), the word *"camara"* (*"pantry"*) indicating the room is correctly labelled, while in the sentence *"Salut, redu intensitatea becurilor in camara la 3"* (*"Hello, reduce the intensity of the lights in the pantry to 3"*), *"camara"* is ignored as a slot.
- Also in scenario 2, artist names that are composed of more than one word are not labelled at all (e.g. in the sentence *"Pune melodia cantata de Leonard Cohen"* – *"Put on the song by Leonard Cohen"*) and occasionally the channel slot is also not identified (e.g. in the sentence *"Schimba pe programul FoodTV"* – *"Change the channel to FoodTV"*).
- For all of the sub-scenarios of scenario 3, the class which is the least represented is also the one which is the worst performing, which is in accordance with our expectations.

To classify these types of errors in a more formal manner and to obtain an estimate of how often they occur, we extracted a subsample from all of the errors produced in scenario 3 by Wit.ai and by the best performing Rasa model (SPACYFASTTEXT). In the case of Wit.ai there are a total of 833 intent errors and 324 slot errors, while for the Rasa SPACYFASTTEXT model, there are 661 intent errors and 421 slot errors. We randomly sampled 10% of the total intent errors and 10% of the total slot errors for each tool, manually classifying each error into one of 2 classes: pattern errors, which are those samples exhibiting one of the previously identified and described patterns, and unknown errors, the remaining ones for which we couldn't identify a pattern. For Wit.ai, around 80% of the intent errors and 85% of the slot errors were pattern errors, while in the case of the Rasa model, there were approximately 78% intent errors and 80% slot errors which we classified as pattern errors. This leads us to the conclusion that finding ways of addressing and fixing the erroneous patterns would significantly improve the performance of the models from both tools.

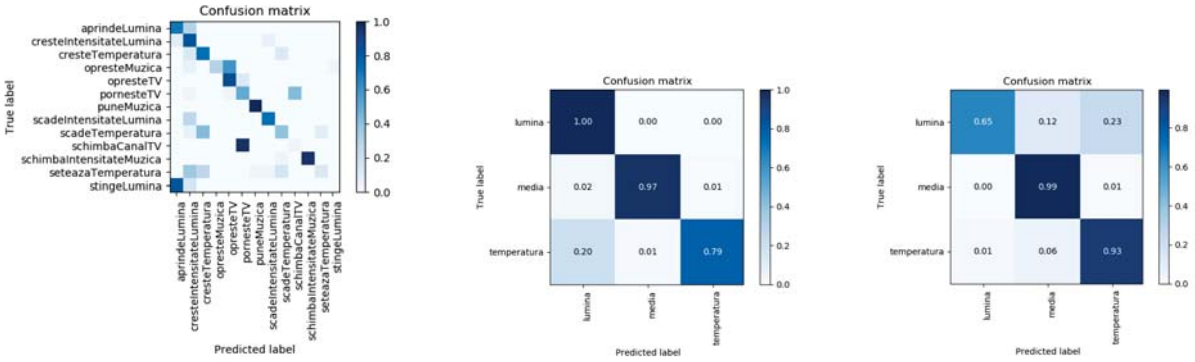
TABLE V
WORD VECTOR SIMILARITIES

Antonyms		**Synonyms**	
Word pairs	**Similarity**	**Word pairs**	**Similarity**
aprinde - stinge	61.74	aprinde - porneste	32.87
aprinde - scade	35.99		
aprinde - schimba	37.37		
aprinde - opreste	31.63		
aprinde - inchide	36.21		
creste - diminueaza	52.13	creste - ridica	45.32
creste - micsoreaza	50.63	creste - mareste	55.23
seteaza - mareste	50.67	seteaza - schimba	37.55
seteaza - micsoreaza	57.01	seteaza - ajusteaza	56.98
seteaza - porneste	48.60		
ridica - reduce	43.80	ridica - mareste	36.46

We also investigated the inter-class errors, between the three intent super-classes: light, temperature and media. Figures 2b and 2c contain the super-class confusion matrices for the SPACYFASTTEXT model trained on scenarios 1 and 3.1. It can be seen that, even though opposite intents are being confused due to the word embeddings, the model does a better job at keeping the classified intent within the same super-class. The confusion matrix of scenario 3.1 shows that the under-represented super-class (in this case, the light super-class) is also the one having the most inter-class errors.

Performing these experiments we have found that the default settings for the Wit.ai predictions where we did not explicitly request confidence levels for the intent left a significant amount of examples with no predicted intent. After we specifically requested confidence levels we observed that in those cases they were low, meaning that Wit.ai detects when the model is unsure and rather than producing a possible bad output leaves the example unclassified. This behaviour was detected mostly in the cases where an inter-class error was made.

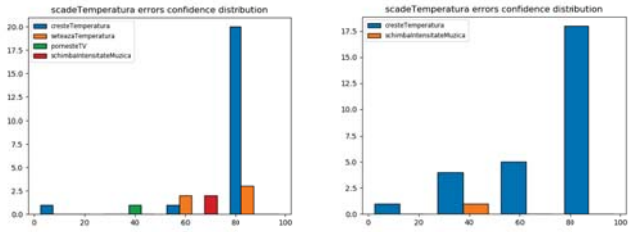
Looking at the reported confidence levels for the intent errors, we noticed that they range from very low to high values. Figures 3 and 4 show the distribution of the confidence level for the incorrect predictions of a certain intent. In figure 3 it can be seen that for both Wit.ai and the best performing Rasa NLU model (SPACYFASTTEXT), when the confidence value is high an error that is due to the word embeddings



(a) Intent confusion matrix for the SPACYFASTTEXT model, scenario 1 (b) Super-class confusion matrix for the SPACYFASTTEXT model, scenario 1 (c) Super-class confusion matrix for the SPACYFASTTEXT model, scenario 3.1

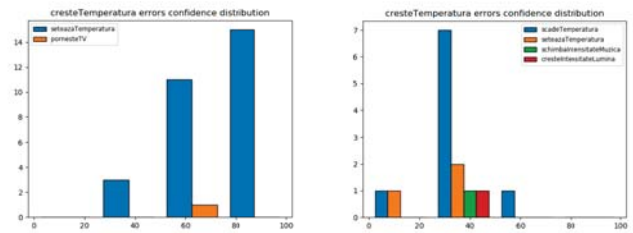
Fig. 2. A set of representative confusion matrices

similarity of opposite words is produced. This leads to intents with opposite meanings being mixed up. On the other hand, other errors that are random are made with lower confidence by the tools. Figure 4 shows a different situation, where in the case of Rasa, the confidence levels decrease slightly and the pattern errors (where *CresteTemperatura* is classified as *ScadeTemperatura* or *SeteazaTemperatura*) occur with low confidence. The opposite is true for Wit.ai, where the pattern errors are still made with high confidence. In figure 5 we plot the overall distribution of the confidence levels in erroneous predictions, for all intents. It can be seen that Wit.ai in general is more confident when making certain errors, while Rasa is trying to be more ”cautious”.



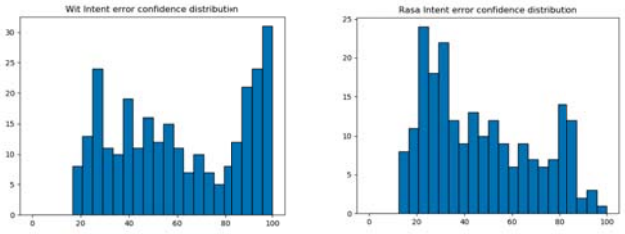
(a) Wit.ai model (b) SPACYFASTTEXT model

Fig. 3. Confidence distributions for the *scadeTemperatura* intent, scenario 3.1



(a) Wit.ai model (b) SPACYFASTTEXT model

Fig. 4. Confidence distributions for the *cresteTemperatura* intent, scenario 3.1



(a) Wit.ai model (b) SPACYFASTTEXT model

Fig. 5. Overall confidence distributions for scenario 3.1

V. DISCUSSION

To see how much the model performance would improve by anticipating the data from the test set distribution, we generated a second version of the datasets for all scenarios (except the baseline scenario), in which we introduced in the training set one example per intent from the test set distribution. After training and evaluating the models on the newly generated datasets, we saw that the performance increased by quite a lot, even after just one example, especially for the slot filling task. In the case of Wit.ai, which exhibited the most improvement across all scenarios of this new dataset, the slot filling F1 in scenario 3.1 increased from 73.01 to 96.25 and the intent detection F1 improved from 29.48 to 50.25. The SPACYFASTTEXT model also produced better results across all scenarios, the slot filling F1 increasing from 73.59 to 93.37 and the intent detection F1 increasing from 44.87 to 49.74. The highest impact was made for the number slots error pattern (”douazeci si sase”), all of the errors exhibiting this pattern being completely fixed after encountering 2-3 training examples containing this type of temperature value slot.

As for the other error patterns discussed, they seem to persist in these updated instances of the datasets, however the number of errors from the unknown class is reduced.

Since most of the error patterns identified are due to the word embeddings, further attention should be given to training word vectors in such a way that the meaning of opposite

words is better represented. Since this is the most frequent type of error produced, this would bring about the most improvement in the model. Because we don't have access to the underlying implementation of Wit.ai, we cannot control what word embeddings it uses. However, with Rasa we can plug in another set of word vectors.

Another source of problems in this field is that of emerging intents. The home assistant domain is expected to grow as more and more house utilities are integrated with it. For each of these emerging functionalities we would need to gather a labelled dataset to train the model, which is a tedious and time consuming task. Moreover, we would like to have a tool that is more robust to this kind of changes. For this, a solution would be to research and incorporate zero-shot learning techniques such as the one presented in the [12]. Unfortunately, the tools we investigated do not support this type of learning.

VI. Conclusion

In addition to the language challenge, the problem of intent detection and slot filling for building home assistant applications raises several data-related issues: class imbalance – unseen intents, synonyms and handling complex, unseen formulations.

In this paper we explore how these challenges affect the performance of two established tools (Wit.ai and Rasa NLU). We defined a set of thirteen home assistant intents belonging to three categories (lights, temperature and media) and generated several datasets tailored to the various data complexities considered, which we used to perform benchmark evaluations on the tools and test our hypotheses. In addition, we examined the types of errors produced by both tools and identified some recurring patterns, as well as their potential sources. This analysis indicated that both tools focus too much on the verb for identifying intents, and that antonyms, class-imbalance and certain small variations in formulation greatly impact intent and slot identification.

Acknowledgment

The work presented in this paper was supported by grant no. 72PCCDI/01.03.2018, *ROBIN - Robots and Society: Cognitive Systems for Personal Robots and Autonomous Vehicles*.

References

[1] J. Gao, M. Galley, and L. Li, "Neural approaches to conversational AI," *CoRR*, vol. abs/1809.08267, 2018. [Online]. Available: http://arxiv.org/abs/1809.08267

[2] A. Garg and M. Agarwal, "Machine translation: A literature review," *CoRR*, vol. abs/1901.01122, 2019. [Online]. Available: http://arxiv.org/abs/1901.01122

[3] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 1–47, Mar. 2002. [Online]. Available: http://doi.acm.org/10.1145/505282.505283

[4] G. Wiedemann and A. Niekler, "Document retrieval for large scale content analysis using contextualized dictionaries," *CoRR*, vol. abs/1707.03217, 2017. [Online]. Available: http://arxiv.org/abs/1707.03217

[5] "Wit.ai," https://wit.ai/, accessed: 2019-06.

[6] "Rasa," https://rasa.com/, accessed: 2019-06.

[7] L. A. Ramshaw and M. P. Marcus, "Text chunking using transformation-based learning," in *Natural language processing using very large corpora*. Springer, 1999, pp. 157–176.

[8] L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider, "Abstract meaning representation for sembanking," 2013.

[9] O. Abend and A. Rappoport, "UCCA: A semantics-based grammatical annotation scheme," in *Proceedings of the 10th International Conference on ComputationalSemantics, IWCS 2013, March 19-22, 2013, University of Potsdam, Potsdam, Germany*, 2013, pp. 1–12.

[10] R. McDonald, J. Nivre, Y. Quirmbach-Brundage, Y. Goldberg, D. Das, K. Ganchev, K. Hall, S. Petrov, H. Zhang, O. Täckström, C. Bedini, N. Bertomeu Castelló, and J. Lee, "Universal dependency annotation for multilingual parsing," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 92–97. [Online]. Available: https://www.aclweb.org/anthology/P13-2017

[11] R. Sarikaya, G. E. Hinton, and B. Ramabhadran, "Deep belief nets for natural language call-routing," in *2011 IEEE International conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2011, pp. 5680–5683.

[12] C. Xia, C. Zhang, X. Yan, Y. Chang, and P. S. Yu, "Zero-shot user intent detection via capsule neural networks," *arXiv preprint arXiv:1809.00385*, 2018.

[13] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[14] B. Liu and I. Lane, "Recurrent neural network structured output prediction for spoken language understanding," in *Proc. NIPS Workshop on Machine Learning for Spoken Language Understanding and Interactions*, 2015.

[15] P. Xu and R. Sarikaya, "Convolutional neural network based triangular crf for joint intent detection and slot filling," in *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*. IEEE, 2013, pp. 78–83.

[16] B. Liu and I. Lane, "Attention-based recurrent neural network models for joint intent detection and slot filling," *arXiv preprint arXiv:1609.01454*, 2016.

[17] C. Zhang, Y. Li, N. Du, W. Fan, and P. S. Yu, "Joint slot filling and intent detection via capsule neural networks," *arXiv preprint arXiv:1812.09471*, 2018.

[18] G. M. Weiss, "Mining with rarity: A unifying framework," *SIGKDD Explor. Newsl.*, vol. 6, no. 1, pp. 7–19, Jun. 2004. [Online]. Available: http://doi.acm.org/10.1145/1007730.1007734

[19] K. W. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *CoRR*, vol. abs/1106.1813, 2011. [Online]. Available: http://arxiv.org/abs/1106.1813

[20] C. Lemnaru and R. Potolea, "Imbalanced classification problems: Systematic study, issues and best practices," in *Enterprise Information Systems*, R. Zhang, J. Zhang, Z. Zhang, J. Filipe, and J. Cordeiro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 35–50.

[21] "Chatito," https://github.com/rodrigopivi/Chatito, accessed: 2019-04.