Cost effective, Multilingual, Privacy-driven voice-enabled Services
www.compriseh2020.eu

Call: H2020-ICT-2018-2020
Topic: ICT-29-2018
Type of action: RIA
Grant agreement Nº: 825081

WP Nº4:	Cost-effective multilingual voice interaction
Deliverable Nº4.2:	Initial weakly supervised learning library
Lead partner:	USAAR
Version Nº:	1.0
Date:	29/04/2020

Document information	
Deliverable Nº and title:	D4.2 – Initial weakly supervised learning library
Version Nº:	1.0
Lead beneficiary:	USAAR
Author(s):	Thomas Kleinbauer (USAAR), Ali Davody (USAAR), Imran Sheikh (INRIA)
Reviewers:	Marc Tommasi (INRIA), Askars Salimbajevs (TILDE)
Submission date:	29/04/2020
Due date:	30/04/2020
Type[1]:	OTHER
Dissemination level[2]:	PU

Document history			
Date	Version	Author(s)	Comments
25/03/2020	0.1	Thomas Kleinbauer & Imran Sheikh	Content draft
14/04/2020	0.2	Thomas Kleinbauer, Imran Sheikh, Ali Davody	Initial version
22/04/2020	0.3	Thomas Kleinbauer, Imran Sheikh, Ali Davody	Revised version integrating feedback and comments from the reviewers
29/04/2020	1.0	Zaineb Chelly & Emmanuel Vincent	Final version revised by the project manager and the coordinator

[1] **R**: Report, **DEC:** Websites, patent filling, videos; **DEM:** Demonstrator, pilot, prototype; **ORDP:** Open Research Data Pilot; **ETHICS:** Ethics requirement. **OTHER:** Software Tools

[2] **PU:** Public**; CO:** Confidential, only for members of the consortium (including the Commission Services)

Document summary

This deliverable introduces COMPRISE's weakly supervised learning library in its initial incarnation. Weakly supervised learning addresses the problem that state-of-the-art supervised learning methods for many relevant tasks require enormous amounts of manually labelled data. In this document, we explore alternatives to this costly endeavour for two specific areas of the project: Speech-to-Text and text processing.

Deliverable 4.2 consists of two parts: a software library and this document. An overview of the different system architectures is given in Section 2. The scientific approaches underlying the software tools are detailed in Section 3. For practical applications, Section 4 gives in-depth instructions on how to install and use the library. Finally, we discuss our achievements so far and present our plans for the next steps until month 27 (February 2021).

For speech-to-text, we discuss the role of supervised learning of acoustic and language models and the transition from semi-supervised to weakly supervised learning methods. Our work leverages error prediction in a semi-supervised setup and dialogue states for weak supervision. The merit of the proposed methods is demonstrated through experiments which evaluate the improvements in speech-to-text performance under limited data scenarios. We also present a discussion on the cost savings expected from the proposed methods.

For weakly supervised text processing, we focus on the Named Entity Recognition task as it is the basis for privacy-preserving text transformations in WP2. We present two approaches, namely a weakly supervised learning approach on dialogues and our first experiments with cross-domain Named Entity Recognition (currently on newswire data). We demonstrate the recognition gains achieved through our methods by presenting and analysing relevant experiments.

Further, this deliverable introduces all necessary software tools for users and developers to apply our techniques. The two parts of the library can be accessed here:

- **Speech-to-Text**: https://gitlab.inria.fr/comprise/spoken-language-understanding-weakly-supervised-learning
- **Text processing:** https://gitlab.inria.fr/comprise/speech-to-text-weakly-supervised-learning

Table of contents

1. Introduction 6
2. Design and implementation of learning components 7
  2.1. Learning components for Speech-to-Text (STT) 7
  2.2 Learning components for text processing 10
3. Scientific approach 12
  3.1. Weakly supervised learning for Speech-to-Text 12
    3.1.1 Semi-supervised to weakly supervised training 12
    3.1.1.1 Training with STT error prediction 13
    3.1.1.2 Weak supervision with dialogue state labels 14
    3.1.2 Experiments and evaluation 15
    3.1.2.1 Experimental setup 15
    3.1.2.2 Evaluation of error detection driven semi supervision 16
    3.1.2.3 Evaluation of dialogue state based weak supervision 19
    3.1.3 Discussion on cost effectiveness 19
  3.2. Weakly supervised learning for text processing 20
    3.2.1 Approach 22
    3.2.2 Data 23
    3.2.2.1 Clean labels 23
    3.2.2.2 Noisy labels 25
    3.2.3 Experiments and evaluation 28
    3.2.4 Cross-domain Named Entity Recognition 32
4. Software library 35
  4.1 Library for weakly supervised learning of Speech-to-Text 35
    4.1.1 Prerequisites 35
    4.1.2 Setup 36
    4.1.3 Typical usage steps 36
    4.1.3.1 Err2Unk based training 36
    4.1.3.2 Dialogue state based training 38
  4.2 Library for weakly supervised learning of text processing 40
    4.2.1 Prerequisites 40
    4.2.2 Configuration 40

4.2.3 Data format 41
4.2.4. Running the code 42
4.2.5 Auxiliary tools 42
5. Summary and outlook 44
References 46

1. Introduction

Voice and language technologies have seen a major boost in the last ten years thanks to the development of new techniques for statistical classification. Especially the advent of deep neural networks has pushed the performance boundaries of many speech and natural language processing tasks. However, despite the advantages of statistical learning, in practice very large amounts of data are often required in order to learn best-performing models. While collecting high quality data may already be a major effort, it is often only the first step: when *supervised* learning methods are employed, the collected data must also be annotated which adds a whole layer of complexity, effort, and cost. This is one of the reasons why the voice assistant market is currently dominated by big corporations: the cost for the entire pipeline of collecting and curating data, and then training good statistical models is often too high for small and medium enterprises.

In COMPRISE, we are exploring the possibility of using *weakly supervised* learning methods for speech recognition and spoken language understanding tasks that do not require the same amount of data annotation efforts. Weakly supervised learning methods exploit existing weak labels which are inexpensive and easier to obtain. The weak labels can be noisy predictions made by an existing model, error patterns of a model, high-level labels not directly related to the task at hand, etc. This contrasts with classical *supervised* learning methods which require detailed annotations for all data, and with *semi-supervised* learning which combines a small amount of labelled data with a large amount of completely unlabelled data.

The main goal of this line of our work is to enable smaller European companies to enter the market for voice-based technologies although they do not have the means to dedicate sizable budgets to elaborate annotation tasks. To this end, we examine multiple alternative approaches that, depending on the specific task, may bring the best possible results. While performance on par with supervised methods on large amounts of fully annotated data cannot be expected, we are looking for approaches that offer the best effort-to-outcome ratio, i.e., the best performance for the smallest annotation cost.

The two important parts of voice-based systems we focus on in COMPRISE are speech and text processing. Since these two fields are quite different in nature, they require individual methods tailored to the specificity of each task.

Speech-to-Text (STT) typically relies on two main component models, namely the Acoustic Model (AM) and the Language Model (LM). Most practical applications begin with an initial AM and an initial LM, which are typically trained on crowd- or expert-transcribed speech data. Speech data collected from the application users is then used to improve the AM and the LM on a regular basis. While improvements of the AM may saturate after some iterations, LM may continue to improve. In any case, there is a need for reliable transcription of the collected speech data, which incurs a heavy cost when employing human transcribers.

In favour of cost effectiveness, semi-supervised training of neural network AMs has been an active topic of research in the STT community. Most common approaches for semi-supervised training of AMs generate automatic transcripts for unsupervised speech data using a seed STT trained on the supervised speech data. However, domain mismatch with existing data collections and limited amount of realistic unsupervised speech data — a typical situation when building a new voice assistant app — limit the performance of state-of-the-art semi-supervised training methods. By contrast, weakly supervised learning methods are expected to achieve performance gains beyond those obtained from semi-supervised training, with zero or minimal increase in labelling costs. The speech processing part of the library provides two sets of methods for training STT models, namely training with STT error prediction and weak supervision with dialogue states. It focuses on obtaining reliable transcriptions which can then be used for training both STT AMs and LMs.

Besides speech processing, many voice-based applications require additional Spoken Language Understanding (SLU) steps to extract various types of information from the spoken user input or to classify a spoken utterance according to a given scheme. Examples include, e.g., Named Entity Recognition (NER) where mentions of specific persons, locations, organisations, etc., are to be automatically identified, and intent classification, where the task is to determine the purpose of each utterance in a conversation.

For this initial version, we focus on NER for the text processing part of the library because it plays a central role for the work on *Text Transformation* in Work Package 2. However, the approach described below is not limited to this task. In fact, one of our next steps will be to apply this weak supervision technique to other SLU and Dialogue Management tasks, and measure its effectiveness.

The situation addressed by this Work Package is that in which developers who intend to employ COMPRISE technology in their business application find themselves in a situation where they need to retrain some of the COMPRISE machine learning models to best match their application domain. Without such retraining, the performance would likely be poor because the specificities of the domain will not be captured sufficiently by general purpose models, or by models created for different domains.

In this deliverable, we present the current state of our research that addresses these problems for speech-to-text and text processing. We introduce both an initial version of the software components needed to reproduce, extend, and apply our results as well as a detailed description of the scientific underpinnings.

2. Design and implementation of learning components

2.1. Learning components for Speech-to-Text (STT)

The initial weakly supervised learning library of COMPRISE provides two main learning components for STT. These two components represent the two main approaches proposed in COMPRISE: (a) training driven by STT error predictions and (b) weakly supervised training based on utterance-level dialogue state labels. These two approaches can be used independently or combined together, if utterance-level weak labels are available. These approaches are presented and evaluated in Section 3.1.1. Details on the usage of the corresponding software components are provided in Section 4. In this section, we present a high-level overview of the design and implementation of the software components. As per the COMPRISE global architecture, weakly supervised learning of STT entirely takes place in the training branch and on the cloud platform.

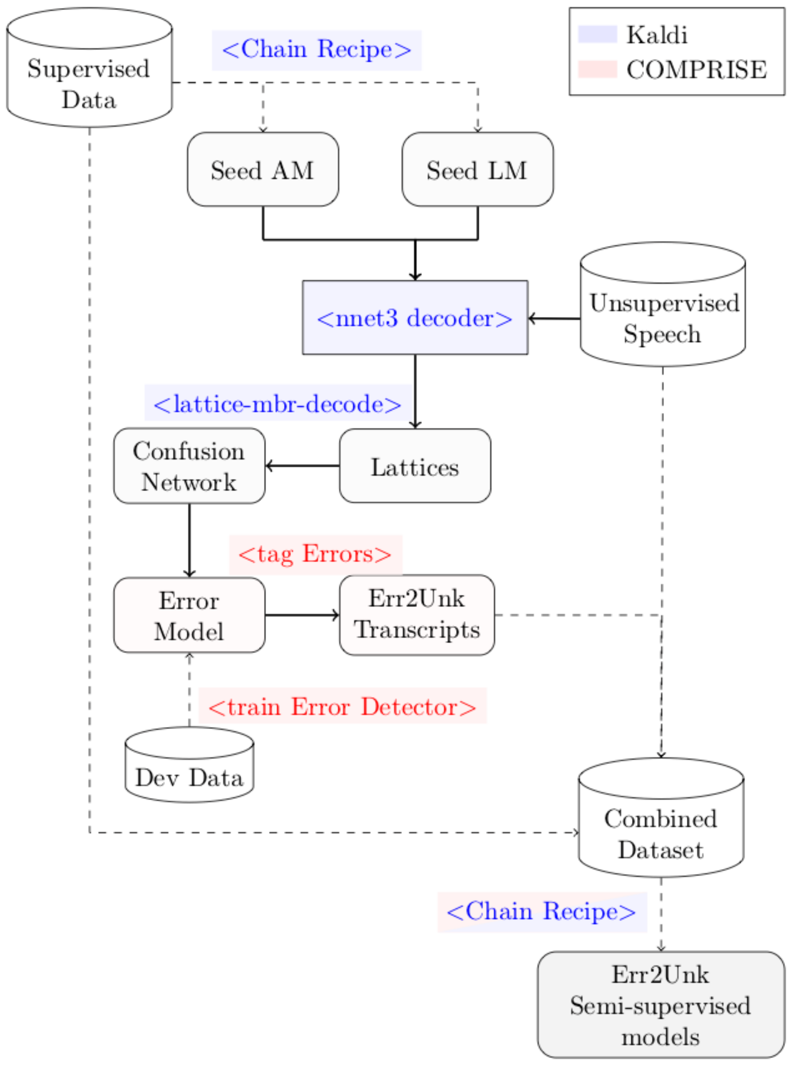

Figure 1: Overview of error detection driven training of STT models.

Figure 1 presents a block diagram to describe the design and implementation of error detection driven training of STT models. Table 1 presents a quick reference on its design and implementation. The implementation also relies on the Kaldi STT toolkit[3], as highlighted in blue. The main contributions through the COMPRISE library are highlighted in red.

Figure 2 presents a block diagram to describe the design and implementation of dialogue state driven weakly supervised training of STT models. Table 2 presents a quick reference on its design and implementation. Along with the Kaldi STT toolkit, the implementation relies on the SRILM toolkit[4].

[3] http://kaldi-asr.org/

[4] http://www.speech.sri.com/projects/srilm/

Table 1: Design and implementation of error detection driven training of STT.

Step	Description	Inputs	Outputs	Implementation notes
1	Train seed STT models	supervised data	seed AM, LM	Kaldi Chain recipe: Bash scripts with Kaldi binaries for processing speech, Perl scripts to manage inputs and train LM, Python scripts to train AM.
2	Prepare for Error Detection	unsupervised speech, dev data	confusion networks	Kaldi binaries to decode speech to lattices, convert lattice to confusion network
3	Train Error Detector	dev confusion networks	error detection model	COMPRISE libraries: Python scripts to extract features, train error model
4	Get unsupervised speech transcripts	unsupervised speech confusion networks	speech transcripts	COMPRISE libraries: Python scripts to tag errors, format transcripts
5	Retrain models	combined data	new AM, LM	Kaldi Chain recipe (as Step 1)

Table 2: Design and implementation of dialogue state driven weakly supervised training of STT.

Step	Description	Inputs	Outputs	Implementation notes
1	Train seed STT models	supervised data	seed AM, LM	Kaldi Chain recipe: Bash scripts with Kaldi binaries for processing speech, Perl scripts to manage inputs and train LM, Python scripts to train AM.
2	Prepare un-supervised data	unsupervised speech	decoded lattices	Kaldi binaries to decode speech to lattices
3	Train dialogue state LMs	supervised data	dialogue state LMs	COMPRISE libraries: Bash scripts relying on SRILM tool
4	Rescore lattices for training	unsupervised speech lattice, dialogue state LM	rescored lattices and transcripts	COMPRISE libraries: Bash scripts relying on Kaldi binaries
5	Retrain models	combined data	new AM, LM	Kaldi Chain recipe (as Step 1)

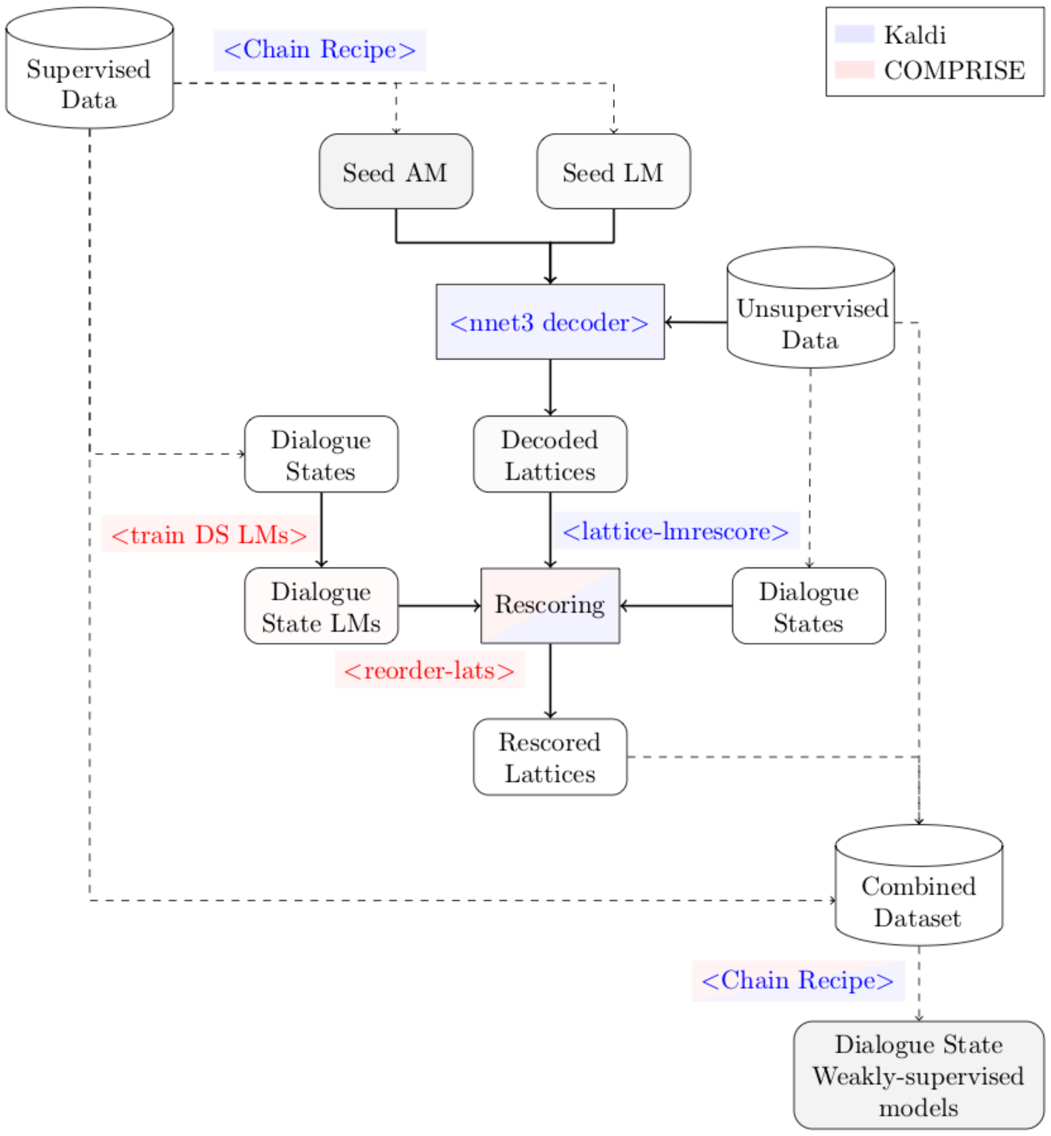

Figure 2: Overview of dialogue state driven weakly supervised training of STT models.

2.2 Learning components for text processing

The second part of this library concerns text processing. We present a high-level overview of the software components for our weakly supervised learning approach. In this initial version of the library, we focuse on one specific task, viz. Named Entity Recognition, although the implemented algorithm is general and will be transferred to a wider spectrum of tasks in future revisions.

Figure 3 displays the general architecture of the text processing part of the weakly supervised learning library. It is implemented using the Python programming language and the PyTorch[5] toolkit for neural networks. Table 3 presents a quick reference on its design and implementation.

[5] https://pytorch.org/

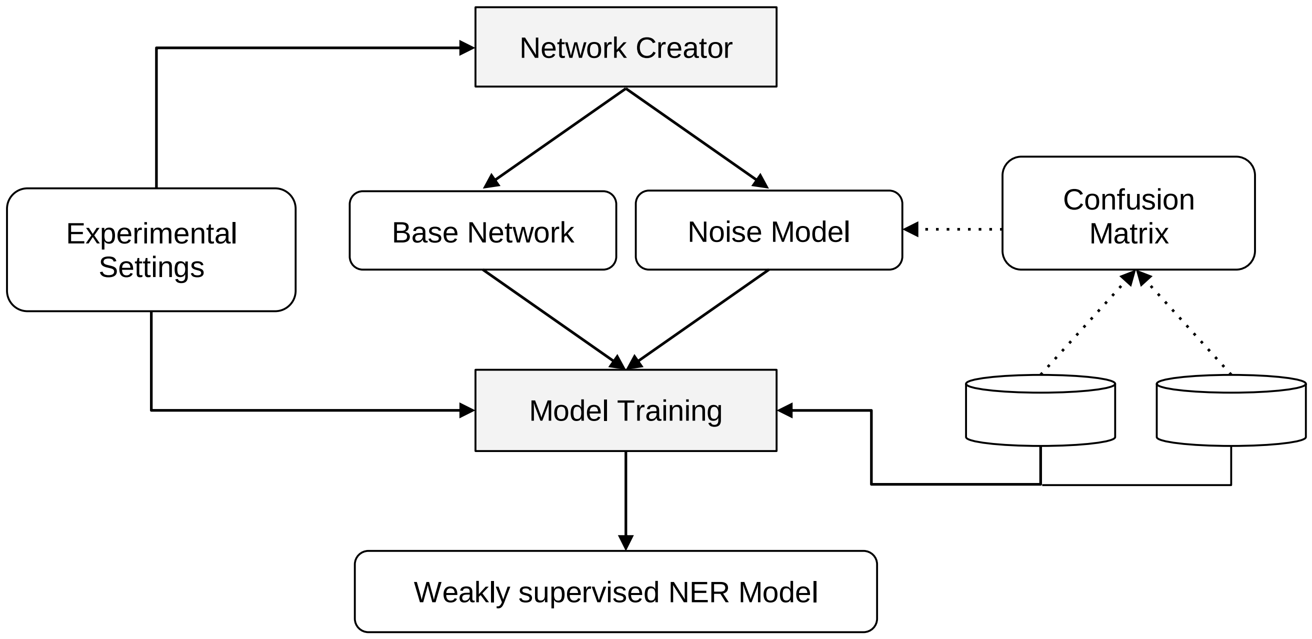

Figure 3: Overview of weakly supervised learning components for text processing.

Table 3: Design and implementation of weakly supervised learning for Named Entity Recognition.

Step	Description	Inputs	Outputs	Implementation notes
1	Load config	Settings file	Experimental settings	Parses command line parameters and loads a JSON-encoded settings file.
2	Create base neural network	Experimental settings	Base network	Creates a BILSTM model for Named Entity labeling
3	Create noise model	Experimental settings, clean and noisy data	Noise model	Clean (= manually annotated) and noisy (weakly annotated) data are used to initialise a noise matrix (confusion matrix) for a noise layer that sits on top of the base network
4	Train network	Experimental settings, Neural models, clean and noisy data	Weakly supervised NER model	Base and noise model are trained in turns, using a configurable batch size and amount of epochs

3. Scientific approach

The initial weakly supervised learning library applies our research outcomes in COMPRISE. The scientific approaches followed for STT and text processing are described below.

3.1. Weakly supervised learning for Speech-to-Text

State-of-the-art STT systems use deep neural network based AMs trained with sequence training objectives like Connectionist Temporal Classification (CTC) or Lattice-Free Maximum Mutual Information (LF-MMI) [6]. The LF-MMI approach requires smaller amounts of transcribed speech data. However, its performance starts degrading on less than 100 h of conversational speech data [6]. In favor of cost effectiveness, semi-supervised training of neural network AMs has been an active topic of research in the STT community. Most common approaches for semi-supervised training of AMs generate automatic transcripts for unsupervised speech data using a seed STT model trained on supervised speech data. Typical approaches perform *best path* decoding of the unsupervised speech and select new speech-transcript pairs based on different filtering schemes (see [7] for relevant references). Interestingly, the LF-MMI approach has been extended to semi-supervised training of AMs [7] and remains state-of-the-art. However, domain mismatch and limited amount of unsupervised speech data are expected to affect its performance.

Similarly, STT systems rely on statistical n-gram LMs or deep neural network based LMs, based on the amount of domain/application specific text data available for training and adapting the LM. Reliable text transcriptions from the target domain/application are required in both approaches. For some applications, existing text data can be exploited to train LMs. However, most novel applications must rely on fully supervised or semi-supervised methods to obtain speech transcriptions for LM. Moreover, it must be noted that cost effective semi-supervised training methods have different end objectives for training AMs and LMs. AM training depends on low-level phone sequences which may even come from STT lattices containing alternate hypotheses (as in the case of semi-supervised LF-MMI). By contrast, LM training requires reliable transcripts (or information) of possible high-level word sequences.

3.1.1 Semi-supervised to weakly supervised training

As compared to semi-supervised training, weakly supervised training methods exploit existing weak labels which are inexpensive and easier to obtain. The weak labels can be noisy predictions, error patterns or high-level labels not directly related to the task at hand. Weak supervision is expected to achieve performance gains beyond those obtained from semi-supervised training, with zero (or minimal) increase in labelling costs. Moreover, these methods can build and improve on top of existing semi-supervised training methods. In the context of STT, starting with semi-supervised training methods is both essential and effective. Seed models can be trained on public domain speech corpora [8] or existing application-specific corpora. The initial COMPRISE library for weakly supervised learning of STT provides two sets of methods for training of STT models, namely training with STT error prediction and weak supervision with dialogue states. It should be noted that the initial library focuses on obtaining reliable transcriptions which can be used for training both STT AMs and LMs.

3.1.1.1 Training with STT error prediction

We propose a new error detection driven semi-supervised training method, which uses an explicit error detection model to tag the reliable word sequences in the hypotheses obtained on the unsupervised speech data. Figure 4 shows a block diagram of the proposed approach. As shown in the figure, supervised training data with accurate speech-transcript pairs are used to train the seed AM and the seed LM. The supervised training data could be a good amount of read speech or a small amount of application-specific data. These seed models are used to decode the unsupervised speech data into STT lattices. STT confusion networks (aka sausages) are obtained from these lattices using Minimum Bayes Risk (MBR) decoding [9].

An error detection model is trained on features extracted from the STT confusion network [10]. Feed-forward neural networks or sequence based Bi-directional Long Short Term Memory (BLSTM) neural networks can be used as the error tagging model. A basic problem with training an STT error detector model is that the confusion statistics and errors made on the STT training set and unseen utterances do not match. Hence, the error detector is trained on a development set which belongs to the target domain.

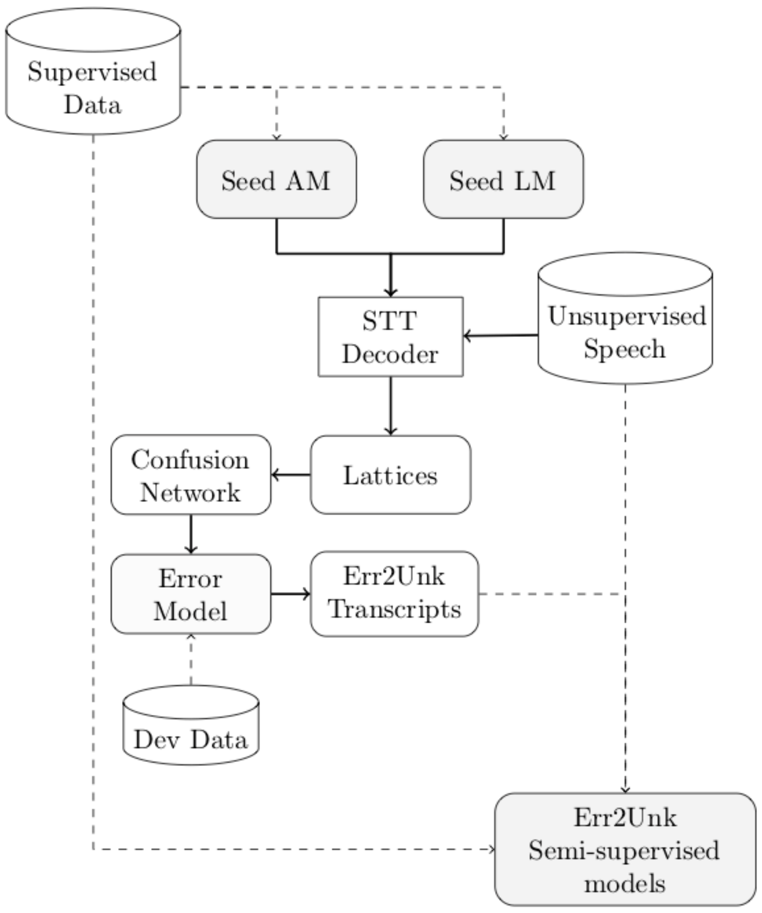

Figure 4: Error detection driven semi-supervised training of STT models.

An error detector model, discussed below, tags the confusion network bins into three classes: <no-error>, <error> and <eps>. Tag <no-error> indicates that the most probable symbol in the confusion bin is correct. <error> represents substitution, deletion and insertion errors due to the most probable symbol. <eps>[6] differentiates a bin that 'correctly hypothesises' no output word as the most probable symbol. We use these tags from the error detector to (a) retain the most probable symbol of a confusion bin tagged as <no-error>, (b) accept the 'no output word' hypothesis of a confusion bin tagged as <eps>, and (c) replace the most probable symbol of a confusion bin tagged as <error> with the <unk> symbol which corresponds to the garbage phone or spoken noise. Hence, we refer to this approach as *Err2Unk* semi-supervised training. Once the Err2Unk transcriptions are obtained for the unsupervised speech data we combine this dataset with the supervised dataset and train new STT models. The AM is trained using the standard LF-MMI approach and the LM is a statistical n-gram model.

3.1.1.2 Weak supervision with dialogue state labels

Human-machine dialogue typically involves a sequence of system-user utterance pairs, representing questions and answers between the system and the user. The dialogue system maintains the history and state of the dialogue to manage the general flow of the conversation. Table 4 shows a sample human-machine dialogue and the possible dialogue states maintained internally by this dialogue system. The ground-truth transcription of the user utterance may not be available for training and improvement of the STT models. However, the corresponding dialogue states may be available, from the dialogue system or through inference on the non-reliable STT transcriptions. Moreover, the dialogue states can be used as weak labels to obtain a more reliable automatic transcription of the user utterance using more complicated methods, which otherwise cannot be used in a live dialogue system.

We propose to use dialogue state adapted LMs to obtain more reliable word sequences in the hypotheses obtained on the unsupervised speech data. Figure 5 shows a block diagram of the proposed approach. Similar to semi-supervised training, a seed AM and a seed LM trained on the supervised dataset are used to decode the unsupervised speech data. Given dialogue states corresponding to utterances in the unsupervised data, pre-trained dialogue state specific LMs are used to rescore the decoded lattices. Hypotheses for the unsupervised speech are obtained from the rescored lattices. We merge this dataset with the supervised dataset and train the dialogue state based weakly supervised models using one of the previously discussed semi-supervised training methods.

Table 4: Example of a human-machine dialogue and possible dialogue states.

Dialogue States	System Utterance	User Utterance
WELCOME	*hi how may I help you*	*cheap restaurant near me*
REQUEST FOOD	*what kind of food do you prefer*	*chinese cuisine*

[6] From epsilon arc (or null arc) in STT and Weighted Finite State Transducer literature.

SUGGEST PLACE	*how about shanghai express*	*sounds good to me*
REQUEST CALL	*should i place a call*	*no tell me the closing time*

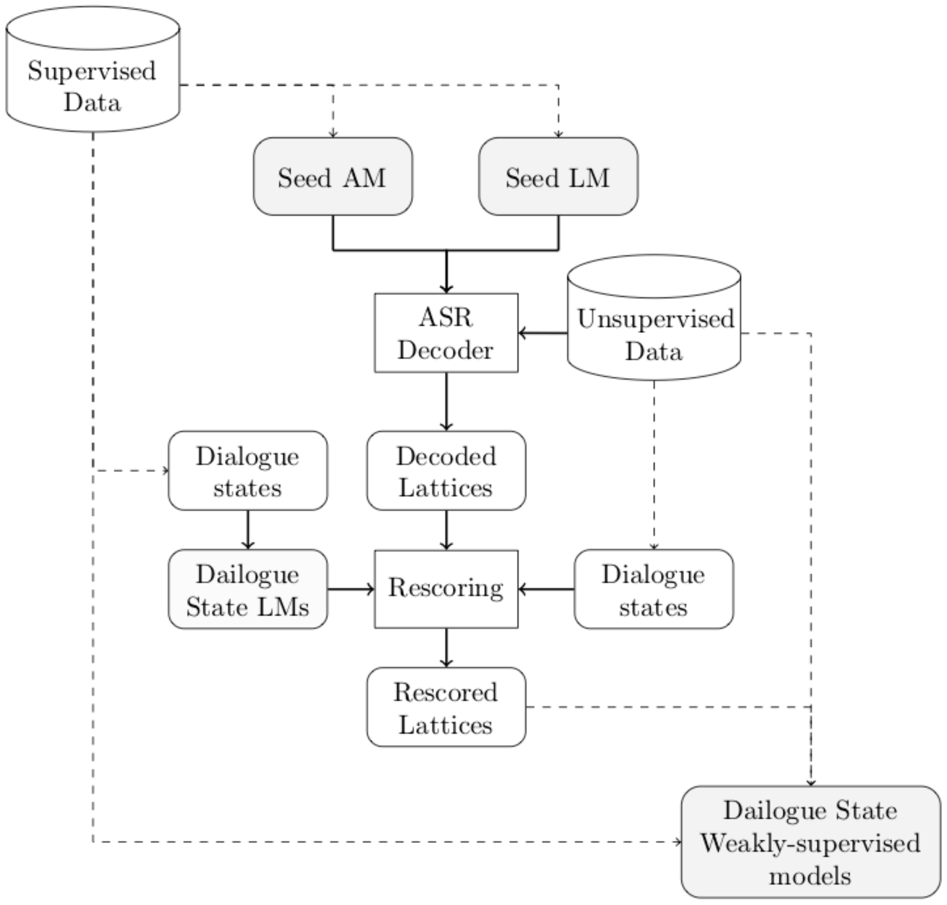

Figure 5: Dialogue state based weakly supervised training of STT models.

3.1.2 Experiments and evaluation

3.1.2.1 Experimental setup

We investigate STT improvements in three different scenarios detailed below. Table 5 briefly presents the different datasets and their splits used in these evaluation setups.

Table 5: Datasets and splits used for evaluation. LS: Librispeech, VM: Verbmobil, LG: Let's Go.

	LS100-VM20	**VM5-VM20**	**LG4-LG19**
Supervised	LS 100 h	VM 5 h	LG 4 h
Unsupervised	VM 20 h	VM 20 h	LG 19 h

Dev	VM 2 h	VM 2 h	LG 6 h
Test	VM 3 h	VM 3 h	LG 6 h

Read speech to conversational speech: Public domain speech data collection efforts have led to a significant amount of read speech in different languages [8]. However, AMs trained on such read speech corpora show degraded performance on conversational speech. Moreover, not all languages have large amounts of read speech datasets to begin with [8], and conversational speech data is limited in amount and mostly untranscribed in the initial stages. We emulate this scenario by using the English read speech corpus Librispeech [11] as our supervised dataset, and English conversational speech from the Verbmobil corpus [12] as the unsupervised data. We denote this setup as *LS100-VM20*. See Table 5 for the amount of data in different splits. We have ensured that there are no overlapping speakers or conversations across Verbmobil splits.

Human-human conversations: We investigate the matched domain limited data scenario wherein both the supervised and unsupervised data belong to the Verbmobil corpus. We denote this setup as *VM5-VM20*. We have ensured that there are no overlapping speakers or conversations across these two subsets. The development and test sets are the same as for LS100-VM20.

Human-machine dialogue: We also analyse the matched domain scenario wherein both the supervised and unsupervised data are human utterances extracted from a human-machine dialogue system. We use subsets of the annotated Let's Go dataset[7] to emulate this setup. We use data corresponding to the first 15 days of October 2008 as the supervised dataset and speech corresponding to the next two and half months as unsupervised data. Data corresponding to the first 15 days and the last 15 days of September 2009 are treated as development and test sets, respectively. We denote this setup as *LG4-LG19*.

3.1.2.2 Evaluation of error detection driven semi supervision

Table 6, Table 7 and Table 8 compare the performance of different methods for semi-supervised training of AMs and LMs on the LS100-VM20, VM5-VM20, and LG4-LG19 setups, respectively. Comparison is in terms of the Word Error Rate (WER) obtained on the development and test sets. We also report the Relative WER Improvement (RWI) with respect to the seed model, which is calculated as

$$\mathrm{RWI} = \frac{\mathrm{SeedModelWER} - \mathrm{GivenModelWER}}{\mathrm{SeedModelWER}} \cdot 100.$$

Additionally, we also include the WER Recovery Rate (WRR) obtained with respect to the seed model, calculated as:

[7] https://dialrc.github.io/LetsGoDataset/

$$\mathrm{WRR} = \frac{\mathrm{SeedModelWER} - \mathrm{GivenModelWER}}{\mathrm{SeedModelWER} - \mathrm{OracleModelWER}}.$$

The WER, the RWI and the WRR of our proposed Err2Unk approach (see Section 3.1.2.1) are compared to those achieved by different models and approaches. These include the seed models, classical semi-supervised training with *best path* decoding of the unsupervised speech, state-of-the-art semi-supervised LF-MMI, and *oracle* (fully supervised) models which have access to ground truth transcriptions of the unsupervised speech. Our Err2Unk approach uses a BLSTM error detection model which achieves micro-average F1 scores of 0.77, 0.84, and 0.81 on the unsupervised speech data of the LS100-VM20, VM5-VM20, and LG4-LG19 setups, respectively. The LMs are trained on the text data available for training AMs. In case of semi-supervised LF-MMI, the word-level hypotheses corresponding to the best path are used to train the LM.

Table 6 shows that semi-supervised LF-MMI performs similarly to classical semi-supervised training in a domain-mismatched, limited data setup. Err2Unk semi-supervised training performs significantly better than the other two semi-supervised approaches. WER improvements across AM training and AM+LM training are consistent. WRRs across these two training scenarios are quite similar. However, relative improvements (RWI) indicate that WER improvements from LM are higher, except for the Err2Unk approach where AM and LM improvements are almost similar.

The evaluation in Table 7 shows that the proposed Err2Unk approach performs best even in a matched-domain setup. WRR comparisons show that the Err2Unk approach can achieve 75% of the possible improvement in AM and 45% of it in AM+LM. Semi-supervised training with the best path performs surprisingly well and better than semi-supervised LF-MMI in this setup. Comparison of the test WERs and RWIs in Tables 6 and 7 indicates that the improvement achieved by semi-supervised training reduces in domain-mismatched scenarios (in Table 6), even though seed models are trained on a larger amount of read speech data.

Table 6: Semi-supervised training on domain-mismatched data (LS100-VM20). Librispeech 100 h (LS100) is used as supervised data and Verbmobil 20 h (VM20) as unsupervised data. AMs are trained with LF-MMI.

Type of Supervision	3-gram LM on LS100 (AM only improvement)				3-gram LM on respective data for AM (AM + LM improvement)			
	Dev WER	Test WER	Test RWI	Test WRR	Dev WER	Test WER	Test RWI	Test WRR
Seed (LS100)	40.50	40.00	-	-	40.50	40.00	-	-
Semi-sup best path	38.96	38.49	3.8%	15%	35.42	35.76	10.6%	24%
Semi-sup LF-MMI	38.15	38.28	4.3%	17%	34.17	35.23	12.0%	27%
Semi-sup Err2Unk	**37.14**	**36.45**	**8.9%**	**35%**	**32.63**	**33.04**	**17.4%**	**39%**
Oracle (LS100+VM20)	30.75	29.63	-	-	21.94	22.20	-	-

Table 7: Semi-supervised training on matched-domain human-human conversations (VM5-VM20). Verbmobil 5 h (VM5) is used as supervised data and Verbmobil 20 h (VM20) as unsupervised data. AMs are trained with LF-MMI.

Type of Supervision	3-gram LM on VM5 (AM only improvement)				3-gram LM on respective data for AM (AM + LM improvement)			
	Dev WER	Test WER	Test RWI	Test WRR	Dev WER	Test WER	Test RWI	Test WRR
Seed (VM5)	39.29	38.56	-	-	39.29	38.56	-	-
Semi-sup best path	32.95	33.05	14.3%	66%	31.92	32.06	16.9%	38%
Semi-sup LF-MMI	34.25	34.91	9.5%	44%	33.76	34.35	10.9%	25%
Semi-sup Err2Unk	**32.19**	**32.26**	**16.3%**	**75%**	**30.83**	**30.91**	**19.8%**	**45%**
Oracle (VM5+VM20)	30.23	30.17	-	-	21.28	21.43	-	-

Table 8 shows the performance on matched-domain dialogue system utterances. The proposed Err2Unk approach continues to give the best performance, although the WER difference is not statistically significant over semi-supervised LF-MMI. LM improvements are limited in this setup, as it can be observed from the RWI and also the WER of the oracle system. We hypothesise that this could be due to the presence of several new named entities in the development and test set.

Table 8: Semi-supervised training on matched-domain human-machine dialogues (LG4-LG19). Let's Go 4 h (LG4) is used as supervised data and Let's Go 19 h (LG19) as unsupervised data. AMs are trained with LF-MMI.

Type of Supervision	3-gram LM on LG4 (AM only improvement)				3-gram LM on respective data for AM (AM + LM improvement)			
	Dev WER	Test WER	Test RWI	Test WRR	Dev WER	Test WER	Test RWI	Test WRR
Seed (LG4)	36.67	36.03	-	-	36.67	36.03	-	-
Semi-sup best path	34.36	33.61	6.7%	32%	33.95	33.05	8.3%	30%
Semi-sup LF-MMI	32.44	31.21	13.4%	63%	32.11	30.94	14.1%	52%
Semi-sup Err2Unk	**32.33**	**30.98**	**14.0%**	**66%**	**32.05**	**30.56**	**15.2%**	**56%**
Oracle (LG4+LG19)	30.01	28.38	-	-	28.31	26.20	-	-

3.1.2.3 Evaluation of dialogue state based weak supervision

Table 9 presents an evaluation of the proposed weakly supervised training of STT models (see Section 3.1.2.2). Weak supervision from dialogue states is used to improve over the previously presented semi-supervised training approaches. Table 9 shows that this weak supervision significantly improves the performance of the best path based semi-supervised training approach (evaluated in Table 8), by contributing both towards AM and LM improvement. A combination of the two proposed approaches (Weak-sup DS, Err2Unk), i.e., error detector driven semi-supervised training and dialogue state based training of AM+LM, gives even better improvements.

Table 9: Weakly supervised training with dialogue state (DS) on human-machine dialogues (LG4-LG19). Let's Go 4 h (LG4) is used as supervised data and Let's Go 19 h (LG19) as unsupervised data. AMs are trained with LF-MMI. DS based LMs are applied in the "AM+LM improvement" results only.

Type of Supervision	3-gram LM on LG4 (AM only improvement)				3-gram LM on respective data for AM (AM + LM improvement)			
	Dev WER	Test WER	Test RWI	Test WRR	Dev WER	Test WER	Test RWI	Test WRR
Seed (LG4)	36.67	36.03	-	-	36.26	35.20	-	-
Semi-sup best path+DS	33.64	32.82	8.9%	42%	33.00	31.65	10.1%	38%
Weak-sup DS, Err2Unk	32.43	30.71	14.8%	70%	31.65	30.04	14.7%	56%
Oracle (LG4+LG19)	30.01	28.38	-	-	27.91	25.92	-	-

3.1.3 Discussion on cost effectiveness

In order to highlight the cost effectiveness of our method we present a quick analysis which translates WER improvements into the number of hours of human transcribed training data that would be required to achieve the same performance with fully supervised training. Figure 6 plots the WER achieved by different systems as a function of the amount of supervised training data. The graph corresponding to supervised AM+LM shows the WER obtained by training the STT models (AM+LM) on different subsets of the Let's Go dataset. Semi-supervised LF-MMI and the proposed Err2Unk semi-supervised training approach use 4 h of supervised data and 19 h of unsupervised data (LG4-LG19), as discussed in Section 3.1.3. The WERs obtained from these methods are mapped to the equivalent number of hours of supervised training data. Note that the development set also comprises human transcribed speech data but it is not counted as part of the supervised training data in this chart, since all the compared methods rely on the development set for performance tuning.

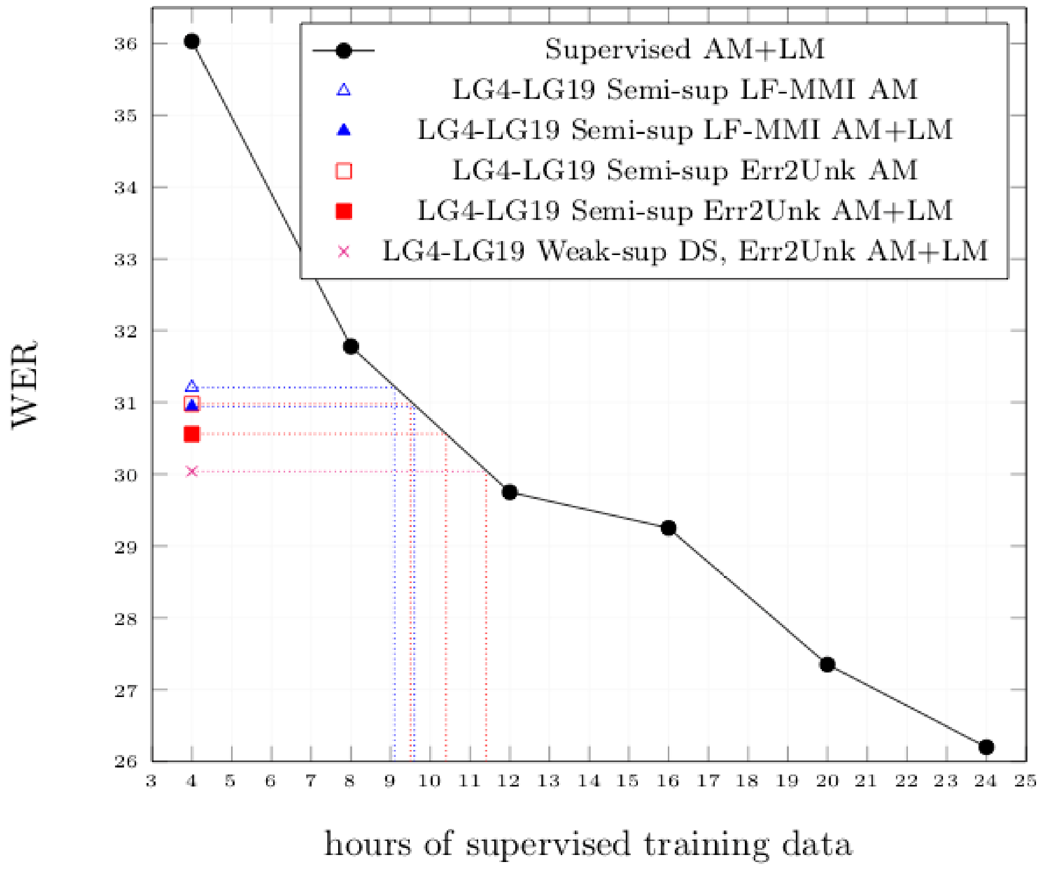

Figure 6: Comparison of all methods in terms of the amount of supervised training data required to achieve a given WER on the Let's Go dataset. The development set is used by all methods and is not included here in the supervised training data.

Figure 6 shows that the proposed Err2Unk approach for semi-supervised AM+LM training achieves a WER equivalent to that obtained with ~10.4 h of supervised training data, whereas it actually uses 4 h only. This implies a 62% relative reduction in the required amount of supervised training data without including the development set, and a 39% relative reduction if we include the development set. Semi-supervised LF-MMI achieves a WER performance equivalent to that obtained with ~9.6 h of supervised training data, which implies 58% and 36% relative reductions without and with the development set, respectively. The combination of the two proposed approaches (Weak-sup DS, Err2Unk) achieves a WER performance equivalent to that obtained with ~11.4 h of supervised training data. This implies 65% and 43% relative reductions without and with the development set, respectively.

3.2. Weakly supervised learning for text processing

We now describe our weakly supervised learning approach for SLU. Generally speaking, before developers can begin training a new model, they need to perform two preparatory tasks:

1. create a sufficiently large collection of in-domain data,
2. annotate the collected data with the desired labels.

Here, we are talking about a *supervised* learning task where the goal is to classify data points into one or multiple predefined categories. Each category has a distinct label (for instance, for the NER task, labels such as LOCATION, ORGANISATION, etc.) from a small set of labels, and annotating means identifying instances of the categories and attaching the correct label to them.

Even if we focus on the task of NER, step 2 is still domain-dependent, since different applications require different types of named entities to be recognised. For instance, in a medical domain, the names of specific conditions or diseases are of central relevance, while such entities will possibly never appear in data for a cooking assistant.

The data collection task (step 1) will become fairly simple once the application has been deployed to a user-base: more and more real-life data is created every time the application is used, and COMPRISE provides the necessary means to make use of this growing data collection in a privacy-preserving manner. However, even before deployment, a certain amount of realistic data will have to be collected to bootstrap the model. While this step is not the focus of our work here, there are a few options on how to create such an initial data collection, including manually creating such data (bearing the risk of not being too realistic), simulations (such as, e.g., Wizard-of-Oz experiments), or using existing data collections, if applicable. In the following, we assume that appropriate data have already been collected and focus on step 2, the annotation task. Note that a potential difference between data collected for bootstrapping and data collected during the actual use of the deployed system is that the former, when done under lab conditions, may not have undergone any privacy transformations. In this initial version of the library, we focus on this condition and will study the effect of WP2 text transformations on weakly supervised learning in subsequent releases.

The basic underlying assumption of weakly supervised learning is that:

1. Manually labeling the data will give high-quality annotations, but is expensive.
2. There are automatic means to create mass annotations cheaply, but imperfectly.

As a consequence of these assumptions, it is often prohibitive for the developer to fully annotate all of the collected data by hand, but manually labeling a small portion is feasible. On the other hand, all data can be annotated automatically but the resulting labels are expected to be wrong every now and then. Annotations that are incorrect at a significant percentage are said to be *noisy*.

This leads to interesting research questions, such as:

- How much data needs to be manually annotated minimally in order to reach a decent performance in the resulting machine learning model?
- How much noise in the automatic annotations can be handled by the weak supervision model?

For the latter, it is clear that in cases when there is *too much* noise in the automatically generated annotations, there will not be any or only very little relevant information to be gained from providing such annotations at all. It is thus interesting to study how robust a weak supervision approach is to the amounts of noise and also to which types of noise.

3.2.1 Approach

Our approach is based on previous work at USAAR in the context of low-resource languages where the problem of non-existent manual annotations is similar albeit differently motivated [2]. The basic idea is to augment a neural-network architecture with a noise layer that explicitly models the noise found in the automatically annotated data. That is, our model consists of two parts, a base model part and a noise model part. The noise layer is only needed during the training and can be dismissed afterwards.

For the task at hand, Named Entity Recognition, our data consist of pairs of words and labels, either clean or noisy. Therefore, let us refer to the clean dataset as $C = \{(x_1, y_1), \dots (x_n, y_n)\}$ and the noisy dataset as $N = \{(x_1, z_1), \dots (x_m, z_m)\}$. Here, the words to be classified are identified by the letter *x*, clean labels are written as *y*, and noisy labels are referred to by the letter *z*. The set of all labels is referred to as *L*.

We then define the base model as a simple multi-label neural network softmax classifier:

$$p(y = i \vee x; w) = \frac{\exp\left(u_i^T h(x)\right)}{\sum_{j=i}^{|L|} \exp\left(u_j^T h(x)\right)}$$

where *h* is a non-linear function or a more complex neural network and *w* are the network weights including the softmax weights *u*. This base model is then extended, as mentioned above, by an additional noise layer to create a noise model, with the idea that the base model is trained only using the clean dataset *C*, and the noise model is trained using only the noisy dataset *N* (see Figure 7).

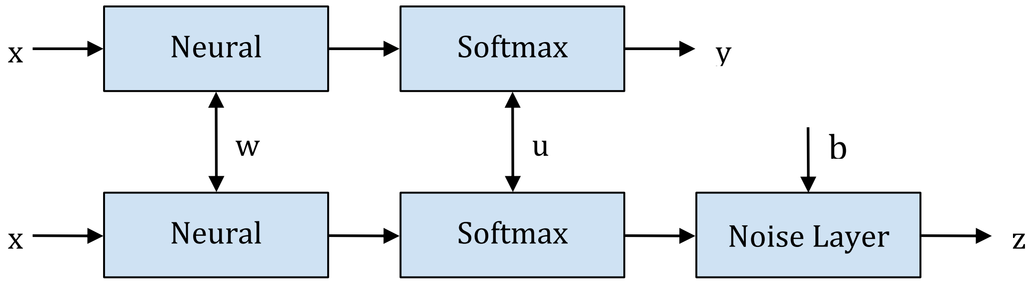

Figure 7: General architecture of our weakly supervised learning approach for NER.

As the vertical arrows indicate, the network and softmax weights are shared between the two models. The noise model θ translates predicted labels to noisy labels: $\theta_{i,j} = p\left(z = l_i \vee y = l_j\right)$ which we estimate as with the help of a confusion matrix *b* and using another softmax function:

$$\theta_{i,j} = \text{softmax}\left(b_j\right)_i.$$

The weights of the confusion matrix are learned, we initialise it by first generating noisy labels for all instances of the clean set *C* employing the same process that is used to generate the labels in *N*. Then, we set the weights by making use of both the clean and noisy data:

$$b_{i,j} = \log\left(\frac{\sum_{t=1}^{|C|} 1_{y_t=l_i} 1_{z_t=l_i}}{\sum_{t=1}^{|C|} 1_{y_t=l_i}}\right)$$

The loss functions for the base and the noise model are the standard cross entropy and negative log-likelihood loss respectively.

In each epoch, the models are trained in alternating fashion, first the base model followed by the noise model. Although we assume that more noisy than clean data is available, each epoch only provides |*C*| data points to the base model and the noise model, respectively. We iterate over *N* in a random fashion to provide different subsets to the noise model in each epoch.

Our method is not the first to model noise using a confusion matrix, see e.g. [3-5]. Many previous works assume that there are no clean annotations available at all. We argue, however, that a small amount of clean annotations is often attainable and can complement the noisy annotations to yield better results overall. A further review of learning in the presence of noisy labels is given in [1].

3.2.2 Data

3.2.2.1 Clean labels

The software for text processing provided in this library has been tested experimentally using the Verbmobil corpus [12], more specifically a subset of the corpus consisting of 27,227 English utterances. Since the corpus does not provide NER labels per se, we used a combination of crowdsourcing and tool-assisted labeling to annotate the corpus: around 20% of the data were annotated by volunteers using the platform Figure Eight[8], while the remaining 80% were first pre-annotated by an automatic tagger (spaCy[9]) and then corrected manually.

As the domain of the used Verbmobil conversations is business meeting negotiations, we employ the five named entities, plus one label for all words that are not named entities (see Table 10).

Table 10: The five Named Entity types used for the Verbmobil annotations.

Label	Description	Example
PER	Person names	*Mr. Miller*
LOC	Locations	*London*

[8] https://www.figure-eight.com

[9] https://spacy.io

ORG	Organisations	*Früh Kölsch brewery*
DATE	Date references	*March*
TIME	Time expressions	*Eight o'clock*
O	All other words	

The words are annotated using the "BIO"-scheme, i.e., for each expression of a named entity of type X, the first word of the expression gets labelled as B-X and all other words (if any) are labelled I-X. The following example illustrates this scheme:

Saturday	*,*	*the*	*fourth*	*,*	*I*	*have*	*a*	*lunch*	*meeting*	*from*	*twelve*	*to*	*two*	*.*
B-DATE	I-DATE	I-DATE	I-DATE	O	O	O	O	O	O	B-TIME	I-TIME	I-TIME	I-TIME	O

The data is split into fixed training, development, test sets as shown in Table 11.

Table 11: Training / development / test split of the Verbmobil data.

split	#sentences	%
train	19151	~70
dev	2846	~10
test	5230	~20

The distribution of labels is roughly the same across all three splits. In Table 12, the "B-" and "I-" labels have been accumulated into single labels for clarity of presentation.

Table 12: Label distributions in the training, development and test datasets.

Label	train (%)	dev (%)	test (%)
PER	0.37	0.30	0.28
LOC	0.69	0.69	0.59
ORG	0.34	0.45	0.37
DATE	8.64	8.57	8.56
TIME	7.01	7.16	7.27
O	82.96	82.83	82.92

3.2.2.2 Noisy labels

For our experiments, we used two different methods to create label noise on the training data, allowing us for a variety of different annotations of the Verbmobil conversations. In the sampling approach, noise is introduced by randomly assigning labels to a choosable subset of the clean data. Complementing this approach is the knowledge-based method, in which external gazetteers are used together with hand-written rules in order to assign labels to the data.

Sampling noise

A straight-forward way to introduce noise into otherwise clean annotations is to replace some portion of the correct annotations with labels that are selected at random. The replacement labels should come from the same set *L* of labels as the clean labels. There are two major parameters in this approach that allow us to control properties of the resulting noisy annotation:

- the portion of the clean annotations to be replaced,
- the sampling distribution $p(l|c)$ over all labels from which to draw the replacements.

Here, the variable *c* represents some context information that could either be globally static or dynamic, i.e., potentially differ from instance to instance. In the simplest case, the distribution is independent of any context, i.e., $p(l|c) = p(l)$. This is the case we will focus on at first.

A natural candidate for the sampling distribution *p* would be the distribution *q* of labels found in the clean annotations, estimated through their relative frequencies. While other candidates are conceivable, for instance a uniform distribution over all labels, there might be some merit in keeping the distribution of noisy labels identical to that of clean data, or at least close to that, since too large discrepancies will likely result in a learned model that performs poorly on the test dataset.

Keeping the label distribution unchanged implies that the individual label counts in the noisy annotations must remain the same as in the clean annotations. Any method that achieves this is akin to randomly shuffling a subset of the clean annotations. That is, (instead of using sampling) create a permutation σ of a random subset $I = \{i_1, \ldots, i_k\}$ of size *k* of the index set $\{1, \ldots, |C|\}$:

$$\sigma = \begin{pmatrix} i_1 & i_2 & \ldots & i_k \\ \sigma(i_1) & \sigma(i_2) & \ldots & \sigma(i_k) \end{pmatrix}$$

Then, define the noisy annotations *N* as:

$$N = \left(C \setminus \{(x_{i_1}, y_{i_1}), \ldots, (x_{i_1}, y_{i_k})\}\right) \cup \{(x_{i_1}, y_{\sigma(i_1)}), \ldots, (x_{i_1}, y_{\sigma(i_k)})\}$$

But, how much noise is actually introduced by shuffling the annotation labels? We define the noise level of *N* with respect to *C* as:

$$\text{noiselevel} = \frac{|\{i \vee y_i \neq z_i\}|}{|C|}$$

The amount of noise does not only depend on the choice of *k*, it is also influenced by the number of i_j for which $\sigma(i_j) \neq i_j$, i.e., the amount of data points that actually receive a label under sigma that is different from their original label. In drastically skewed distributions, where a single label dominates all others in frequency — as is the case for named entity annotations with the "O" label, since most words in ordinary texts are not named entities — the maximum noise level that is possible through shuffling is $\min\left(1{,}2 \cdot \left(1 - f(l)\right)\right)$, where *l* is the dominant label and $f(l)$ its relative frequency in the clean dataset. In our Verbmobil data, the relative frequency of the "O" label is 82.06%, and hence the maximum noise level attainable without changing the label distribution is 17.04%.

If higher noise levels are desired, we need to take different label distributions in the resulting noisy training data into account. Coming back to the original sampling proposal, we can employ the following procedure to create noisy annotations by sampling.

Let $I = \{i_1, \ldots, i_k\}$ be a *k*-subset of $\{1, \ldots, |C|\}$, chosen at random with uniform probability, like before. Then, define the noisy annotation *N* as:

$$N = \left(C \setminus \{(x_{i_1}, y_{i_1}), \ldots, (x_{i_k}, y_{i_k})\}\right) \cup \{\left(x_{i_j}, l_j\right) \vee 1 \leq j \leq k, l_j \sim p\}$$

Or, expressed more algorithmically in the Python programming language, using its NumPy package:

```
def sampling_noise(C, L, p, k)
        # define the index set {0, ..., |C|-1}
        C_indices = numpy.arange(len(C))

        # choose a random k-subset of the index set
        I = numpy.random.choice(C_indices, k, False)

        # create a new label for each data point given by I
        for i in I:
            noisy_label = numpy.random.choice(L, 1, False, p)
            C[i] = (C[i][0], noisy_label)
```

The parameter *k* controls the maximum amount of noise in *N,* with $\frac{k}{|C|}$ being an upper bound for *N*'s noise level which can be expected to be:

$$E(\text{noiselevel}) = \frac{k \cdot \sum_{l \in L} q(l) \cdot (1 - p(l))}{|C|}$$

For the case where $p = q$, the expected noise level is maximised if both distributions give equal probability to all labels (uniform distribution), in which case the expected noise level is $\frac{k}{|C|} - \frac{k}{|L||C|}$. It is not difficult to achieve an actual noise level of $\frac{k}{|C|}$ by altering the sampling procedure above slightly so as to guarantee each newly selected label to be different from the clean label. For that, we insert two additional lines into the for-loop of the above algorithm:

```
# create a new label for each data point given by I
for i in I:
    noisy_label = numpy.choice(L, 1, False, p)
    # make sure the new label is different from the clean one
```

```
while noisy_label == C[i][1]
   noisy_label = numpy.choice(L, 1, False, p)
C[i] = (C[i][0], noisy_label)
```

Of course, this means that the noisy label is no longer selected according to *p*. Rather, the replacements for each label l are based on a distinct distribution p_l given as

$$p_l(x) = \begin{cases} 0, & \text{if } x = l \\ p\dfrac{(x)}{\sum_{y \in L \setminus \{l\}} p}(y), & \text{otherwise} \end{cases}$$

In return, the resulting noise level is now exactly $\frac{k}{|C|}$.

Generating noisy labels without any context information results in annotations that are very unlike the noise introduced by human annotation error. We, thus, also consider a slightly more informed version of the sampling approach that takes two types of context information into account:

- the part-of-speech of the word to be relabeled,
- the label of the preceding word.

The rationale for the first point is that not all parts-of-speech are equally likely to co-occur with named entities. For instance, nouns and compounds have a much higher probability of constituting a location, organisation, or person name than, e.g., prepositions or conjunctions do. Similarly, "I-" labels in the BIO scheme can only follow a "B-" label or another "I-" label of the same type. Therefore, to generate more convincing noisy labels, we can use the conditional probability distribution $p(l \vee pos; prev)$ as the sampling distribution, where *pos* refers to the part-of-speech of the word to relabel and *prev* refers to the label of the preceding word. Again, the distribution can be estimated from the frequencies of occurrence in the clean data. Since not all possible combinations of label, part-of-speech, and previous label are found sufficiently often or are attested at all in our dataset, we first use a simple Laplace smoothing to compute the conditional probability in our experiments:

$$p(l \vee pos; prev) = \frac{f(l, pos, prev) + 1}{f(pos, prev) + |L|}.$$

However, this alone would assign equal probability to all labels for all cases where a specific combination of part-of-speech and previous label do not occur at all in the clean data, i.e., where $f(pos; prev) = 0$. Instead, inspired by Katz’s backoff model, we resort to $p(l)$ in such cases. Using this sampling distribution alone does not prevent unlikely parts-of-speech to be erroneously labelled as a named entity, nor does it always produce consistent BIO-labels. But that is the nature of a probabilistic approach, and it is also not unexpected that noisy annotations do not behave completely like clean annotations. Yet, we also experimented with an alternative way to generate noisy labels that we introduce now.

Knowledge-based noisy annotations

Among the main advantages of the sampling method to creating noise are its flexibility and ease of implementation. However, it requires clean labels to estimate the different sampling distributions and for all situations where only parts of the annotations should be based on noise (i.e., $k < |C|$. In real-world situations, clean labels will usually not be available for the full training set, or else there would be no need for creating a weakly-supervised learning model in the first place. However, it is conceivable that a certain portion of the dataset has been annotated manually. This annotation could then be used to estimate the sampling distribution with which the remaining data could be annotated. The manually annotated portion needs to be sufficiently large to estimate a reliable sampling distribution.

As an alternative, it is possible to leverage external, domain-dependent knowledge to design automatic procedures that provide noisy labels [13]. For instance, in the case of our Verbmobil data, the categories LOC and ORG could be labelled automatically by referring to large databases of locations and organisation names. Similarly, lists of names can be used as the basis for an automatic procedure that identifies PER instances in text. For DATE and TIME, using such a gazetteer seems to be less useful; however, occurrences of such expressions in text can be found by specifically designed scripting rules.

This is exactly the approach we followed for the “knowledge-based annotations”. We refer to such automatic procedures to annotate a dataset (imperfectly) as *weak labelers*. Studying how well the resulting annotations are suited for subsequent weakly supervised learning is especially interesting because it is a realistic methodology that is available to commercial stakeholders at a comparatively low cost.

Following [13], candidate lists were extracted from Wikipedia for PER, LOC, and ORG. In order to achieve higher recall on PER, its list was extended with lists of the most popular first and last names taken from the web. The weak labelers for each of these three categories then identify the longest subsequences in the data and annotate each with the respective label.

One important aspect to pay attention to in this process is how to handle cases where more than one weak labeler matches the same words. This happens for words that are part of more than one list, for instance, “Austin” is both a name and a location and would thus appear in the lists for PER and LOC. Dembowski et al. suggest to not label such instances at all, so as not to risk creating false positive annotations [13]. Here, however, we follow Hedderich and Klakow who instead employ simple heuristics for conflict resolution [2].

Included in the weakly supervised learning library is a tool for weakly labeling data based on matching rules (see Section 4.2). Using this tool, we defined rules to identify expressions of DATE and TIME in the training data. A matching subsequence is labelled only if it has not already been annotated with a different label before. Words that do not get assigned a label by any of the weak labelers, receive the “O” label.

3.2.3 Experiments and evaluation

As discussed above, we have a number of different noisy annotations available for the training set as well as clean annotations for the same training data and also for a development and a test set. The training data consists of 177,375 words but since we want to simulate the situation in which only a small amount of labelled data is available, we only use a randomly chosen 1% of these.

As for noisy data, we compare two variations of the sampling methods described above: sampling solely from the label distribution as inferred from the clean annotations, without any other context information; and sampling from distributions conditioned on the previous label and the part-of-speech of the current label, also inferred from the clean annotations. The part-of-speech information for each word in the training set is automatically generated using the flair NLP framework[10] with its "fast-pos" model.

In addition, we experiment with noisy annotations created through a combination of gazetteers for the PER, LOC, and ORG labels and matching rules for DATE and TIME. The different types of training sets to our disposal are listed in Table 13.

Table 13: Overview of the clean and noisy training data used in our experiments. All data are derived from the Verbmobil corpus.

Name	Description	Noise level
train_clean	1,773 manually annotated words	n.a.
train_noisy_sampled_no_context	train_clean + noisy annotations generated by sampling from an inferred distribution *p(label)*	28.15%
train_noisy_sampled_with_context	train_clean + noisy annotations generated by sampling from inferred distributions *p(label|pos;prev)*	31.26%
train_noisy_kb+rules	train_clean + noisy annotations based on external knowledge	14.90%

Although these datasets were created without any machine learning algorithms, we can compare the three noisy datasets to the clean one in terms of precision, recall and f-score, as is common practice with classifier outputs (see Table 14). Similarly, the noise level is equal to $1 - \text{accuracy}$, a measure also known as *error rate* in the context of measuring classification quality.

As Table 14 illustrates, the knowledge-based method "out-performs" the other two noisy datasets in all measures across all labels.

The simple neural network architecture used in all of our experiments is shown below. The base model, which operates only on the clean portion of our data, consists of a BiLSTM with state-size 300 for input encoding, followed by a linear layer of size 100 and ReLU activation, followed by a second linear layer. We use pre-computed fastText embeddings [14] to encode the words in our dataset. The noise model consists of the same base model with an additional noise layer on top as described above (see Figure 8).

[10] https://github.com/flairNLP/flair

Table 14: Quality of the knowledge-based noisy annotations when compared with the clean annotations, both by individual label and totally.

Label	Precision			Recall			F1-score		
	SNC	SWC	KBR	SNC	SWC	KBR	SNC	SWC	KBR
B-PER	0.00	0.01	0.16	0.00	0.03	0.50	0.00	0.01	0.24
I-PER	0.00	0.00	0.13	0.00	0.01	0.02	0.00	0.00	0.03
B-ORG	0.00	0.00	0.04	0.00	0.01	0.01	0.00	0.00	0.02
I-ORG	0.00	0.00	0.20	0.00	0.02	0.01	0.00	0.00	0.02
B-LOC	0.00	0.2	0.62	0.00	0.11	0.74	0.00	0.04	0.69
I-LOC	0.00	0.0	1.00	0.00	0.02	0.45	0.00	0.00	0.09
B-DATE	0.04	0.25	0.43	0.03	0.28	0.39	0.04	0.27	0.41
I-DATE	0.04	0.10	0.43	0.03	0.06	0.29	0.03	0.08	0.35
B-TIME	0.03	0.20	0.39	0.03	0.27	0.33	0.03	0.23	0.36
I-TIME	0.04	0.20	0.52	0.03	0.15	0.38	0.03	0.17	0.44
O	0.83	0.89	0.92	0.86	0.80	0.95	0.84	0.84	0.94
Macro Avg.	0.09	0.15	0.44	0.09	0.16	0.33	0.09	0.15	0.32
Micro Avg.	0.72	0.70	0.85	0.72	0.69	0.85	0.72	0.70	0.85

On the CoNLL corpus, a standard dataset for NER based on newswire articles, the base model achieves an F-score of around 85%[11].

For the Verbmobil corpus which all of our experiments are based on we first compute a baseline by only training the base model. We use 1,773 (= 1%) of samples with clean labels, and no noisy annotations at all. The model is trained for 100 epochs and then evaluated on the test set, reaching an F-score of 30.03% (averaged over three runs). Here and in the following, the reported F-scores exclude the majority class "O" as is standard practice in Named Entity Recognition research. The relatively low result of 30.03% demonstrates the importance of having enough training data: while manually annotating 1,773 data points could already prove on the border of being too costly for an SME, the classification quality that can be achieved with such a small dataset alone is insufficient.

[11] We follow the tradition in NER of reporting F-scores computed over the non-"O" labels.

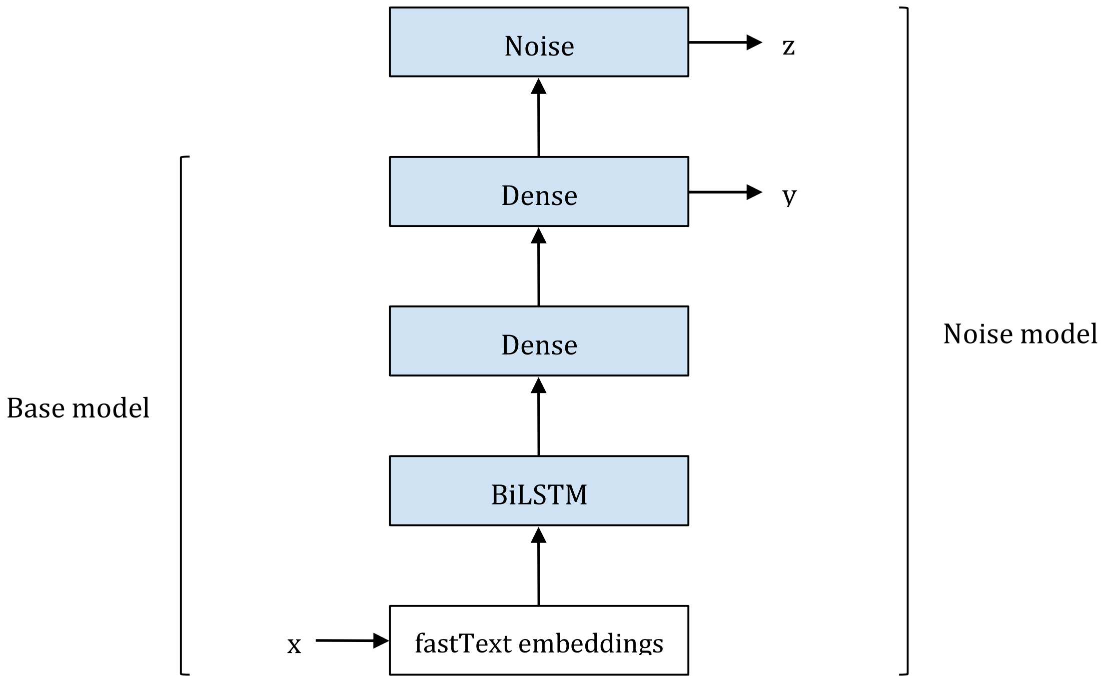

Figure 8: Base and noise model in our weakly supervised NER architecture.

Since we have three different methods available for creating more annotations automatically, we could produce additional baseline conditions by simply adding the noisy annotations to the 1,773 clean annotations and train the base model (without the noise layer) that way. Unfortunately, for both noisy annotations based on sampling, the F-score drops dramatically in this approach to around 2%. In contrast, the noisy annotations generated with gazetteers and rules lead to an F-score of 33.07%, which is in fact slightly better than the initial baseline. This set of noisy annotations contains in fact the smallest amount of noise, yet the drastic performance difference when using sampling noise is not predicted by the differences in noise levels. Instead, the outcome of these baseline experiments hints at a fundamental shortcoming of noise generated by sampling in structural terms, at least when only a small set of clean data can be used for estimating the sampling distributions.

For the actual experiments, we then train our classifier *with* the additional noise layer in the manner described above. As for the baselines, we train each experimental condition for 100 epochs. Table 15 summarises the results for all three noisy datasets and contrasts them with the strongest of our baselines. As we can see, the overly simple sampling method without context fails to even meet the baseline results. However, both the contextualised sampling method as well as the knowledge-based weak labelers lead to much better results.

Although the relative improvements over the baseline are impressive, the recognition rates are still quite low in absolute terms even when keeping in mind that the majority class "O" is not included in these numbers. We thus started exploring alternative methods, one of which we explain in the following section.

Table 15: Results for the weakly supervised NER experiments.

Training data	Precision	Recall	F-score
Baseline (base model only)	35.75	30.77	33.07
train_noisy_sampled_no_context	33.19	29.41	31.15
train_noisy_sampled_with_context	42.29	43.31	42.77
train_noisy_kb+rules	36.82	37.32	37.04

3.2.4 Cross-domain Named Entity Recognition

Is weak supervision the only way to relieve developers of voice-based applications from the cost and effort of annotating large amounts of data in high quality? Or are there alternative options that could possibly even be combined with weakly supervised methods to boost the classification rate further? The general idea to complement the collected data with additional information from external knowledge bases has motivated one of the noisy annotation methods above. While the particular approach did not lead to the best classification results, it did outperform the baseline substantially.

Another possible approach to leveraging external information is the transfer of learned knowledge from a different domain for which a large amount of data already is available to the low-resource target domain in a process called *cross-domain NER*. For instance, the model proposed in [15] tries to utilise cross-domain *language model* training to improve cross-domain NER. The core idea is transferring NER knowledge from the source domain to the target domain by contrasting large raw data in both domains through cross-domain language model training. A more detailed explanation follows.

Given an input sentence, word representations are first calculated through a shared embedding layer. Then a set of task- and domain-specific BiLSTM parameters is calculated through a parameter generation network. Finally, respective output layers are used for different tasks and domains. We have started initial experiments with applying this technique to the COMPRISE task at hand, therefore we quickly summarise the original method, which we use as a baseline, and introduce our own extensions so far.

To get the embeddings, given an input $x = [x_1, x_2, \dots, x_n]$ from a source-domain training set or target-domain, each word x_i is represented as the concatenation of its word embedding and the output of a character level CNN.

Then a bi-directional LSTM layer is applied to the embedding vector$v = [v_1, v_2, \dots, v_n]$. To transfer knowledge across domains and tasks, a parameter generator network is introduced, by decomposing the parameters θ of the NER or LM task on the source or target text domain into the combination $\theta = f(W, I_d, I_t)$ of a set of meta parameters W, a task embedding vector I_t $(t \in \{ner, lm\})$ and a domain embedding vector I_d $(d \in \{src, tgt\})$, so that domain and task-correlations can be learned through similarities between the respective domain and task embedding vectors.

Furthermore, standard CRFs are used as output layers for NER. The network is trained by optimising the NER and LM objectives:

$$L_{ner} = -\frac{1}{|D_{ner}|}\sum_{n=1}^{N} log\left(p(y^n \vee x^n)\right)$$

$$L_{lm} = -\frac{1}{2|D_{lm}|}\sum_{n=1}^{N}\sum_{t=1}^{T} log\left(p^f(x_{t+1}^n|x_{1:t}^n)\right) + log\left(p^b(x_{t-1}^n|x_{t:T}^n)\right)$$

where $D_{ner} = \{(x^1, y^1), (x^2, y^2), \ldots\}$ is the training data for NER and $D_{lm} = \{(x_1^0, x_2^0, \ldots, x_T^0), \ldots\}$ is the language modeling training data.

For now, we take the CoNLL-2003 English NER data as the source domain data, as this is the dataset used in the original publication. For the target domain, a science and technology dataset is collected and labelled in [15]. One of the next steps will be to move to Verbmobil data for compatibility and possible combination with our previous experiments. CoNLL-2003 contains four types of entities: PER (person), LOC (location), ORG (organisation) and MISC (miscellaneous). For the science and technology dataset, 620 articles from CBS SciTech News were collected and annotated following the CoNLL-2003 standard. Applying the aforementioned method in this setting leads to an F-score of 73.59%.

We extend this base approach with two contributions. First, we make use of a masked language model, and second we incorporate latent representations of the labels themselves. While these experiments are preliminary, we are able to achieve improvements over the model described above which we consider our baseline. Both contributions are explained now.

Pretrained models like BERT [16] have recently achieved state-of-the-art scores in the NER task. As another competitive approach, we employ the pretrained BERT language model adapted for the NER task and compare it with the baseline model discussed above. Following [16], to do so, we apply a linear classifier on top of BERT, fine tune it on the source domain and finally test it on the target domain.

To make knowledge transformation from source domain to target domain even more efficient, we add a latent representation of each label to the inputs of the BERT model. Therefore, we also consider another model, which is obtained by augmenting the BERT model with the predefined embeddings for each label. The idea behind this approach is that the model learns the concept of each entity and therefore knowledge transfer is more efficient. An overview of the model is depicted in Figure 9.

A natural question is how to get the representation of labels in the latent space. One simple solution is to take the average of embeddings of the tokens corresponding to each label and use the resulting vector as the embedding for that category. To reflect the statistics in the data, we also weight the contribution of each word by its frequency in the dataset. We then compute the most similar words to these averaged embeddings to see how they are distributed in the latent space. The results for the CoNLL-2003 dataset are depicted in Table 16.

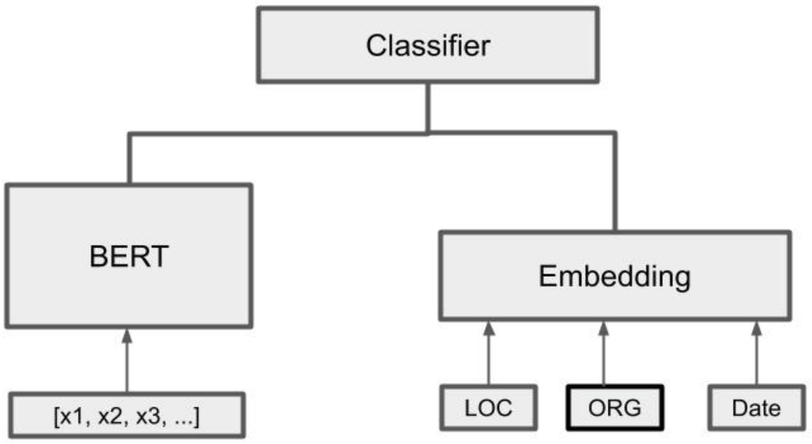

Figure 9: Adding label embeddings to the BERT model.

Table 16: The seven most similar words to the label embedding for the four CoNLL labels.

LOC	ORG	PER	MISC
England	MLB	James	American
Germany	NFL	Scott	German
Europe	Government	Thomas	French
France	NBC	John	Russian
Australia	Army	Smith	British
USA	Yankees	Bennett	Chinese

We compare the F-score of all approaches in Table 17. Empirically, we found that using only the embeddings for "LOC", "PER" and "ORG" results in a higher performance. This can be understood from the fact that tokens with the "MISC" label have more diversity and therefore finding a common representation between source and target domains is more difficult.

As it is evident from Table 17, the BERT model outperforms the method proposed in [15] and adding the label embeddings to BERT further improves the performance. These F-scores are considerably higher than in our weakly supervised learning experiments, but it should be kept in mind that the settings of both experiments differ in multiple ways. The most crucial next step for our research will thus be to unify the two approaches to allow for better comparison.

Table 17: F-scores of the two experiments compared to the baseline.

Model	F-score
Baseline	73.59%
BERT	74.0%
BERT + label embeddings	74.5%

4. Software library

4.1 Library for weakly supervised learning of Speech-to-Text[12]

The initial version of the weakly supervised library for STT provides two main components which represent the two approaches presented above, namely:

1. STT Error Detection driven Training (Err2Unk)
2. Weakly Supervised Training based on Dialogue States

These library components focus on obtaining reliable transcriptions of un-transcribed speech data which can be used for training both STT AMs and LMs. The AMs can be of any type, although we chose the state-of-the-art Chain models in our examples. Statistical n-gram LMs are chosen over other possible LMs to support limited data scenarios.

Readers interested in the high-level design and experimental evaluation of these two components are directed to Section 2.1 and Section 3.1, respectively. This section provides details on typical usage of these two components.

4.1.1 Prerequisites

- This library will re-use binaries and scripts from the Kaldi toolkit. So, you should have Kaldi pre-installed on your system.
- Speech datasets for training STT models, including:
  - (small amount of) transcribed speech data. As demonstrated in Section 3.1, it could be an existing read speech corpus or a few hours of domain/application specific speech corpus.
  - (more) un-transcribed speech data.
  - a development set containing application specific transcribed speech data
- Err2Unk based training requires:
  - the Keras Python library to train neural network models for STT error detection.
  - the kenlm Python module to extract language model related features for error detection.

[12] https://gitlab.inria.fr/comprise/speech-to-text-weakly-supervised-learning

- Dialogue state based training requires:
    - the SRILM toolkit. If you have installed Kaldi, you can install the SRILM toolkit with the *tools/extras/install_srilm.sh* script in your Kaldi installation.
    - a speech dataset from a human-machine dialogue system where a dialogue state corresponding to each human utterance is already available, for example the Let's Go dataset. One could also use human-human conversations or any other speech dataset with any kind of weak (but relevant) utterance level labels.

4.1.2 Setup

- Ensure that you have a working Kaldi installation.
- Modify the softlinks *steps* and *utils* in this directory to point to *egs/wsj/s5/steps/* and *egs/wsj/s5/utils/*, respectively, in your Kaldi installation.
- Modify the path of *KALDI_ROOT* and modify (or remove) the path to *kaldi_lm*, *SRILM* and *sox* tools in path.sh
- Modify *cmd.sh* if you are using a different execution queue for Kaldi.

4.1.3 Typical usage steps

4.1.3.1 Err2Unk based training

Err2Unk based semi-supervised training of STT models will typically involve following steps.

Step 1. Train seed STT models
- Supervised training data with reliable speech-transcript pairs are used to train the seed AM and LM. Note that this step can be skipped if you already have pre-trained AM and LM.
- A sample Kaldi recipe to train the seed AM and LM on a subset of the Let's Go dataset is made available in the *egs/* directory.

Step 2. Prepare for STT Error Detection
- The seed AM and LM are used to decode the unsupervised speech and development set into STT lattices. A sample script is available in *egs/local/* if you are relying on the sample recipe from Step 1.
- Obtain STT confusion networks from the lattices decoded on the unsupervised speech and development set. The COMPRISE library assumes confusion networks are in Kaldi sausage format. Assuming your lattices are generated by Kaldi (as *lat.*.gz*), you can use our script to generate STT confusion networks as follows:

  bash local/err2unk/getSaus.sh lattice_dir graph_dir lm_wt

  graph_dir is the directory used by the Kaldi decoder, and *lm_wt* is the LM weight which gives the best development set WER. Note that STT confusion networks, aka sausages, are generated in the *lattice_dir/sau/* directory, referred to as *saus_dir* in the next steps.

Step 3. Train the STT Error Detector

- Align the development set confusion networks to the corresponding reference transcriptions

  bash local/err2unk/sausAlign.sh saus_dir graph_dir ref_text

  'ref_text' is the reference transcription file in Kaldi format. The output is saved to a text file *saus_dir/saus_bin-best_with-heps.hyp.align* which is referred to as *saus_ref_align* below.

- Extract relevant features and labels from the development set confusion networks

  python local/err2unk/errdet/saus_feats_for_train.py saus_dir saus_ref_align lm_arpa graph_dir dev_saus_feats_n_labs

  lm_arpa is the LM in ARPA format, and *dev_saus_feats_n_labs* is the output file containing features and labels extracted from the confusion networks, which will be used in the next command. Note that the error detector is trained on the application-specific development set.

- Train a Bi-directional Long Short Term Memory (BLSTM) based error tagger

  python local/err2unk/errdet/train_3c_error_tagger_on_dev.py dev_saus_feats_n_labs err_model_dir

  err_model_dir stores the resulting error tagger model. Feedforward neural network based error detectors can also be tried with *local/err2unk/errdet/train_3c_error_mlp_on_dev.py*.

Step 4. Get unsupervised speech transcripts

- Extract relevant features from the unsupervised speech confusion networks, obtained in Step 2.

  python local/err2unk/errdet/saus_feats_for_predict.py saus_dir lm_arpa graph_dir unsup_saus_feats

  unsup_saus_feats is the output file containing features extracted from the confusion networks, which will be used in the next command.

- Tag STT errors on the unsupervised speech confusion networks

  python local/err2unk/errdet/tag_with_3c_tagger.py err_model_dir unsup_saus_feats unsup_error_preds

  unsup_error_preds is a text file containing the error predictions.

- Get Err2Unk unsupervised speech transcripts

  bash local/err2unk/getErr2UnkTranscripts.sh saus_dir graph_dir unsup_error_preds > unsup_text

  unsup_text are the output transcriptions in Kaldi format.

Step 5. Retrain STT models

- Prepare a new data directory, combining supervised and unsupervised data, for training new models

  bash local/err2unk/prepareNewDataDir.sh unsup_text old_sup_data_dir old_unsup_data_dir new_data_dir

  old_sup_data_dir contains *wav.scp* and *utt2spk* used for training the seed AM in Step 1, *old_unsup_data_dir* contains *wav.scp* and *utt2spk* used for decoding unsupervised speech in Step 2, and *new_data_dir* will contain the combined data directory for training new models. Note that this script can be extended to combine *feat.scp* and *cmvn.scp* to avoid repeating feature extraction.

- Train a new AM and LM on the new combined data directory using a Kaldi recipe similar to Step 1.

4.1.3.2 Dialogue state based training

Dialogue state based weakly supervised training of STT models will typically go through the following steps.

Step 1. Train seed STT models

- Supervised training data with reliable speech-transcript pairs are used to train the seed AM and LM. This step can be skipped if you already have pre-trained AM and LM.
- A sample Kaldi recipe to train the seed AM and LM on a subset of the Let's Go dataset is made available in the *egs/* directory.

Step 2. Decode unsupervised speech to lattices

- The seed AM and LM are used to decode the unsupervised speech into Kaldi STT lattices (*lat.*.gz*). A sample script in available in *egs/* if you are relying on the sample recipe from Step 1.

Step 3. Train dialogue state LMs

- Train dialogue state specific LMs

  bash local/dsLMs/trainDialogStateLMs.sh old_lang_test_dir utt_dialog_state_csv ds_lm_dir

  old_lang_test_dir was created during training of seed models (Step 1) and should contain files *words.txt* and *G.fst*, *utt_dialog_state_csv* is the training set 3-column CSV file of form *utt_id,transcript,dialog_state*, and *ds_lm_dir* will contain the dialogue state specific LMs. This script uses the *unk* symbol and a count threshold *minDsCnt* on the minimum number of utterances in a dialogue state. Dialogue states with fewer utterances than this count are ignored and these utterances will resort to the seed LM (*G.fst*) in *old_lang_test_dir*.

- Train interpolated dialogue state specific LMs *bash local/dsLMs/trainInterpolatedDialogStateLMs.sh ds_lm_dir old_lm_arpa int_ds_lm_dir*

old_lm_arpa is the ARPA LM corresponding to the seed LM, and *int_ds_lm_dir* contains the interpolated dialogue state specific LMs. Interpolated dialogue state specific LMs perform better than the dialogue state specific LMs created by the previous command. But the previous command is essential to obtain *ds_lm_dir*.

Step 4. Rescore unsupervised lattices

– Reorganise the old lattice archives into dialogue state specific lattice archives

bash local/dsLMs/reorgLattices.sh data_dir utt_dialog_state_csv old_lat_dir int_ds_lm_dir new_lat_dir

data_dir contains the *wav.scp* file, *utt_dialog_state_csv* is a 3-column CSV file of form *utterance_id,transcript,dialog_state* (without any transcript for unsupervised speech), *old_lat_dir* was created after decoding with seed models and should contain Kaldi format lattice archives (*lat.*.gz*), *int_ds_lm_dir* was created in Step 3, and *new_lat_dir* will contain the reorganised lattice archives ready for rescoring with Kaldi.

– Rescore unsupervised lattices with interpolated dialogue state specific LMs

bash local/dsLMs/rescoreDsLattices.sh old_lang_test_dir int_ds_lm_dir data_dir new_lat_dir rescored_lat_dir

old_lang_test_dir was created during training of seed models (Step 1) and should contain files *words.txt* and *G.fst*, *int_ds_lm_dir* was created in Step 3, *data_dir* should contain reference transcriptions in Kaldi format if you want to computer the WER, *new_lat_dir* contains the reorganised lattice archives ready for rescoring, and *rescored_lat_dir* will contain the dialogue state LM rescored lattice archives

– Get best path transcripts on unsupervised speech

bash local/dsLMs/getBestPathTranscripts.sh rescored_lat_dir words_file lm_wt word_ins_penalty unsup_text

words_file is the *words.txt* file used by the seed models (e.g., in 'old_lang_test_dir'), *lm_wt* is the LM weight which gives the best development set WER, *word_ins_penalty* is 0.0, 0.5 or 1.0 (whichever gives the best development set WER), and *unsup_text* contains the best path transcripts on unsupervised speech.

Step 5. Retrain STT models

– Prepare unsupervised data for training new models

bash local/err2unk/prepareNewUnsupDataDir.sh unsup_text old_sup_data_dir old_unsup_data_dir new_data_dir *old_sup_data_dir* contains *wav.scp* and *utt2spk* used for training the seed AM in Step 1, *old_unsup_data_dir* contains *wav.scp* and *utt2spk* used for decoding unsupervised speech in Step 2, and *new_data_dir* will contain the new combined data directory for training new models. This script can be extended to combine *feat.scp* and *cmvn.scp* to avoid repeating feature extraction.

- Train new AM and LM on the new combined data directory using a Kaldi recipe similar to Step 1.

4.2 Library for weakly supervised learning of text processing[13]

The text processing part of the library consists of the actual weakly supervised learning implementation as well as a number of auxiliary tools. This section describes the necessary steps to install and use them.

4.2.1 Prerequisites

All tools in this part of the library are implemented using the Python programming language, version 3.7. Four external packages are required:

- NumPy
- PyTorch
- fastText
- Flair

These can be installed in a number of ways, using standard Python package installation methods, such as, e.g. "pip":

- *pip install numpy*
- *pip install torch*
- *pip install fasttext*
- *pip install flair*

The training of the machine learning models in this part of the library can be run on the CPU, however, it is recommended to use PyTorch together with the NVIDIA CUDA[14] framework to leverage GPU-based training.

4.2.2 Configuration

Relevant settings for training a weakly supervised NER model are stored in configuration files in JSON[15] format, consisting of a flat JSON object with the following entries:

- **NAME** - The name of the experiment — must match the filename of the configuration file.

[13] https://gitlab.inria.fr/comprise/speech-to-text-weakly-supervised-learning

[14] https://developer.nvidia.com/cuda-zone

[15] https://www.json.org/

- **PATH_TRAIN_CLEAN** - The path to the file containing the manually annotated training data.
- **PATH_TRAIN_NOISY** - The path to the file containing the automatically generated training data.
- **PATH_DEV** - The path to the file containing the manually annotated validation set data.
- **PATH_TEST** - The path to the file containing the manually annotated test set data.
- **DATA_SEPARATOR** - The (whitespace) string used to separate *word, true label, predicted label* in the output format for the CoNLL evaluation script.
- **WORD_EMBEDDING** - The path to fastText word embedding file "cc.en.300.bin".
- **LABEL_FORMAT** - Either *"io"* or *"bio"*, according to which labeling convention was used to annotate the training data.
- **CONTEXT_LENGTH** - The number of words to the left and the right of the word to be labelled that make up the context window.
- **LSTM_SIZE** - The output size of the BiLSTM.
- **DENSE_SIZE** - The output size of the first Dense layer.
- **DENSE_ACTIVATION** - The name of the activation function for the first Dense layer (*"relu"* or *"sigmoid"*)
- **BATCH_SIZE** - The number of samples per batch.
- **EPOCHS** - The number of epochs to train for.
- **USE_NOISY** - must be *True*.
- **USE_IDENTITY_MATRIX** - *True*, if the noise matrix should be initialised as the identity matrix instead of the way described above. Useful for debugging.
- **SAMPLE_SEED** - An integer used to initialise NumPy's random number generator.
- **SAMPLE_PCT_CLEAN** - A value between 0 and 1, specifying the percentage of the full clean dataset to be used (e.g., 0.01)
- **SAMPLE_PCT_NOISY** - A value between 0 and 1, specifying the percentage of the full noisy dataset to be used (e.g., 1.00)
- **NUM_WORKERS** - A positive integer will turn on multi-process data loading with the specified number of loader worker processes.
- **REPORT_INTERVAL** - For a positive integer *i*, logging information is printed out every *i*-th epoch.

4.2.3 Data format

Our implementation uses fastText word embeddings for all words in all datasets. Depending on the size of the datasets, generating these embeddings can take quite a long time. Therefore, we use Python's pickle package which allows us to export objects to files, and to recreate them at a later point by re-importing them. All datasets are thus encoded as Python dictionaries with three entries:

1. *instances* - a list of data points, encoded using class *Instance*.
2. *embedding_dim* - the size of the word embedding vectors. This is currently 300.
3. *remove_label_prefix* - a boolean value indicating whether or not the "B-" and "I-" prefixes of the NER labels should be removed. This value is currently hard-coded to be *True*.

As it can be cumbersome to create these pickle files, we provide a command line tool that can convert annotated data in text format to the correct pickle format expected by the NER software. The format expected by that tool is tab separated values (TSV format) where each line of the file contains a word, followed by a TAB character, followed by its label. Sentence boundaries are marked by empty lines.

4.2.4. Running the code

The main script for running an experiment is “ner.py” which can be invoked as follows:

python3 ner.py [<options>] <settings>

<*settings*> refers to a JSON configuration file that contains all the relevant information for the experiment (see Section 4.2.2). In addition, a number of command line options can be provided. Most of these, except for *--config-dir* and *-h* override the respective values from the JSON configuration file, allowing for quick setup changes without the need to edit the latter. The available options are:

- **-h** – Abort normal operation and print a help screen that explains the options for the script.
- **--config-dir=<PATH>** - Specifies the path to the directory containing the JSON configuration file.
- **--epochs=<EPOCHS>** - The number of epochs to train for.
- **--sample-pct-clean=<float>** - A value between 0 and 1, specifying the percentage of the full clean dataset to be used (e.g., 0.01)
- **--sample-pct-noisy=<float>** - A value between 0 and 1, specifying the percentage of the full noisy dataset to be used (e.g., 1.00)
- **--use-identity-matrix=<bool>** - *True*, if the noise matrix should be initialised as the identity matrix instead of the way described above. Useful for debugging.

The script will then proceed to construct the neural networks, load the data, and start the training process. The progress of the training is logged to the terminal as given by the *REPORT_INTERVAL* setting in the configuration file. At the end of the training, the resulting model is automatically evaluated against the test set in terms of accuracy, precision, recall and F-score overall as well as for each individual label.

4.2.5 Auxiliary tools

Besides the main code, the initial weakly supervised learning library also comes with a number of handy tools, mostly for working with TSV files (see Section 4.2.3):

- **Io2bio.py / bio2io.py**
  - ❖ USAGE: *python3 io2bio.py <infile.tsv> <outfile.tsv>*
    USAGE: *python3 bio2io.py <infile.tsv> <outfile.tsv>*
    - ▪ <*infile.tsv*> - The input file in tab-separated format, annotated according to the IO/BIO scheme.
    - ▪ <*outfile.tsv*> - The output file in tab-separated format, annotated according to the BIO/IO scheme.
  - ❖ The scripts convert the annotation scheme used in the input file from the IO scheme

to the BIO scheme, or vice versa. Note that the conversion BIO -> IO drops information as the boundaries between adjacent annotation of the same type get lost. Similarly, the conversion IO -> BIO assumes that only the first one in a sequence of labels of the same type is a "B-" label.

– **create_pickle.py**
  - ❖ USAGE: *python3 create_pickle.py <label-set> <label-format> <input.tsv> <output.pickle>*
    - ▪ *<label-set>* - The string *"verbmobil".*
    - ▪ *<label-format>* - *"io"* for training sets, *"bio"* for validation and test sets.
    - ▪ *<input.tsv>* - The input file, containing the annotated data in tab-separated format.
    - ▪ *<output.pickle>* - The output file in .pickle format as expected by the main program.
  - ❖ This script creates .pickle files as needed by the main program *ner.py*.

– **date_time_rules.py**
  - ❖ USAGE: *python3 date_time_rules.py <input.tsv> <output.tsv>*
    - ▪ *<input.tsv>* - The input file containing the data to be annotated in TSV format.
    - ▪ *<output.tsv>* - The output file with additional date/time annotation in TSV format.
  - ❖ This script identifies date and time expressions in the input data according to a set of rules. Matching words are annotated in BIO format, unless this would overwrite an existing non-"O" annotation. The result is written to the output file.

– **eval_annotations.py**
  - ❖ USAGE: *python3 eval_annotations.py <true-annotations> <predictions> [<label-bodies>]*
    - ▪ *<true-annotations>* - The path to the file containing the ground-truth annotations in TSV format.
    - ▪ *<predictions>* - The path to the file containing the annotations to evaluate against the ground-truth annotations in TSV format.
    - ▪ *<label-bodies>* - An optional parameter specifying the labels used without "B-" and "I-" prefixes. Default value: *PER, ORG, LOC, DATE, TIME*
  - ❖ This script prints precision, recall, and F-score values for each label. It also prints macro and micro averages and global accuracy. Unlike the standard CoNLL script, the "O" label is also considered in the computation of averages.

– **merge_annotations.py**
  - ❖ USAGE: *python3 merge_annotation.py <file1.tsv> <file2.tsv> [<file3.tsv> …]*
    - ▪ *<fileN.tsv>* - The files whose annotations should get merged. All files must contain the same words in the same order and use the same label set.
  - ❖ This script merges together the annotations given by many files and writes the result to the file *merged.tsv* in the current working directory. For each word, if all annotations label the word as "O", so will the output. Otherwise, the first non-"O" label will be used.

– **sample_merge_annotations.py**
  - ❖ USAGE: *python3 sample_merge_annotations.py <clean.tsv> <noisy.tsv> <noise-level> <merged.tsv> [<force-clean-interval>]*
    - ▪ *<clean.tsv>* - The path to the file containing manually annotated data.
    - ▪ *<noisy.tsv>* - The path to the file containing noisy annotations.
    - ▪ *<noise-level>* - A floating point value between 0 and 1.
    - ▪ *<merged.tsv>* - The output file that combines annotations from *<clean.tsv>* and *<noisy.tsv>*.

- *<force-clean-interval>* - An optional argument of the from *<from>:<to>* where *<from>* and *<to>* define an interval of word indices.

❖ This script creates a new noisy annotation of the words found in the file *<clean.tsv>* by mixing the clean annotations found in the same file with noisy annotations provided by the file *<noisy.tsv>*. For each word, either the clean or the noisy label is selected at random so as to reach the desired noise level. For a continuous stretch of the words it is possible to force the selection of clean labels by providing the *<force-clean-interval>* parameter: *<from>* specifies the first and *<to>* the last index of the words in said stretch of words. The result is written to the file *<merged.tsv>*.

– **sample_noise.py**

❖ USAGE: *python3 sample_noise.py <clean-in.tsv> <noisy-out.tsv>*
- *<clean-in.tsv>* - The input file, containing clean annotations in TSV format.
- *<noisy-out.tsv>* - The output file.

❖ This script samples 1% of the labels found in the input file to create an estimate of the label distribution $p(label)$. Then it creates a noisy annotation for all words in the input file by sampling that distribution. The result is written to the output file.

– **sample_noise_prev_pos.py**

❖ USAGE: *python3 sample_noise_prev_pos.py <clean-in.tsv> <noisy-out.tsv>*
- *<clean-in.tsv>* - The input file, containing clean annotations in TSV format.
- *<noisy-out.tsv>* - The output file.

❖ This script samples 1% of the labels found in the input file to create an estimate of the label distribution $p(label|prev;pos)$. To this end, the flair NLP package is employed to automatically generate part-of-speech labels for each sentence in the input data. The script then creates a noisy annotation for all words in the input file by sampling that distribution. The result is written to the output file.

5. Summary and outlook

The COMPRISE initial library for weakly supervised learning provides software tools for weakly supervised training of speech-to-text and named entity tagging in text.

The tools for training STT models are based on two main approaches proposed in the COMPRISE project, namely training guided by STT error predictions and weak supervision with dialogue states. Experimental evaluation on three different limited speech data scenarios, including domain mismatched and matched conditions, has demonstrated the effectiveness of these methods in terms of significant reduction in word error rate of STT conversion. Moreover, a combination of these two approaches gives further word error rate reductions. The evaluation also presented an analysis on error rates and amount of training data corresponding to fully supervised STT and those for the proposed weakly supervised training methods. This analysis shows that significant cost reductions can be expected from these initial methods proposed in COMPRISE. This should further encourage the use of the libraries provided for weakly supervised STT training.

Future work on weakly supervised learning for STT AMs will explore methods to foster relevant word hypothesis and phone sequences in error regions. Future work on STT LMs will explore neural network based language models which represent words by continuous embedding vectors and are more promising for domain adaptation tasks. Additionally, they can be tightly coupled with weak labels like dialogue states, intents and named entities. Limited target domain data and STT errors will be the two main challenges to be addressed in this direction.

The proposed STT training approaches and libraries can benefit from the weakly supervised text processing methods proposed for named entity recognition tasks. The current dialogue state based weak supervision of STT models relies on existing dialogue state labels. In practice, these dialogue states are available from the dialogue system after inference on the non-reliable ASR transcriptions. Future work will involve a coordination between the weakly supervised training methods for STT and text processing to jointly exploit their benefits.

Our work on text processing has so far focused on the NER task. This will change in future revisions to embrace a greater range of relevant tasks. We have spent considerable effort with the generation of appropriate noisy labels. To this end, we have explored different sampling methods and a knowledge-based method. The used “noise level” metric which is in other contexts known as “test error rate” has not proved helpful in deciding which type of noise to use for our weakly supervised learning experiments as they did not correlate with the final prediction results.

Surprisingly, we found that a simple sampling method that takes a weak form of context into account performs best in our experiments using only 1% of clean data (Verbmobil corpus). Still, the overall performance is low, with F-scores under 50%.

Therefore, we have started looking into alternative or complementary approaches. In particular, we look at cross-domain NER to leverage external knowledge in better ways than our first noise model could make use of. This is an avenue we plan to explore further in the future, possibly together with related work on “few-shot learning”.

However, most importantly, we will perform a more in-depth error analysis to be able to make informed decisions on how to best proceed in our research.

One further dimension that has deliberately been left out for this initial version of the library is that weakly supervised learning in COMPRISE should also be applicable to data collected after a voice-application developed with the COMPRISE SDK has been deployed. In that case, however, the data collected will have been privacy-transformed first. It will thus be an interesting additional challenge how noise not only in the labels but also in the data itself can be dealt with in the training of new models.

Our software is made publicly available under an Open Source license. The two parts of the initial weakly supervised learning library can be accessed here:

- **Speech-to-text**: https://gitlab.inria.fr/comprise/spoken-language-understanding-weakly-supervised-learning
- **Text processing:** https://gitlab.inria.fr/comprise/speech-to-text-weakly-supervised-learning

References

[1] Benoit Frenay and Michel Verleysen. Classification in the presence of label noise: A survey. In IEEETransactions on Neural Networks and Learning Systems, 25(5):845–869, 2014.

[2] Michael A. Hedderich and Dietrich Klakow. Training a Neural Network in a Low-Resource Setting on Automatically Annotated Noisy Data. In Proceedings of the Workshop on Deep Learning Approaches for Low-Resource NLP, Melbourne, Australia, July 2018.

[3] Alan Joseph Bekker and Jacob Goldberger. Training deep neural-networks based on unreliable labels. In Proceedings of the 2016 IEEE International Conference on Acoustics, Speech and Signal Processing, pages 2682–2686, 2016.

[4] Jacob Goldberger and Ehud Ben-Reuven. Training deep neural-networks using a noise adaptation layer. In Proceedings of the 5th International Conference on Learning Representations (ICLR). Toulon, France, April 24-26, 2017.

[5] Sainbayar Sukhbaatar, Joan Bruna, Manohar Paluri, Lubomir Bourdev, and Rob Fergus. Learning from noisy labels with deep neural networks. In Proceedings of the 3rd International Conference on Learning Representations, (ICLR), Workshop Track Proceedings, San Diego, CA, USA, May 7-9, 2015

[6] D. Povey, V. Peddinti, D. Galvez, P. Ghahremani, V. Manohar, X. Na, Y. Wang, and S. Khudanpur, Purely sequence-trained neural networks for ASR based on lattice-free MMI, in Proc. Interspeech 2016, 2016, pp. 2751–2755.

[7] V. Manohar, H. Hadian, D. Povey, and S. Khudanpur, Semi-supervised training of acoustic models using lattice-free MMI, in 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), April 2018, pp. 4844–4848.

[8] Mozilla common voice. [Online]. Available: https://voice.mozilla.org/

[9] H. Xu, D. Povey, L. Mangu, and J. Zhu, Minimum bayes risk decoding and system combination based on a recursion for edit distance, Computer Speech & Language, vol. 25, no. 4, pp. 802 – 828, 2011.

[10] Y. Tam, Y. Lei, J. Zheng, and W. Wang, ASR error detection using recurrent neural network language model and complementary ASR, in 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), May 2014, pp. 2312–2316.

[11] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, Librispeech: An ASR corpus based on public domain audio books, in 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015, pp. 5206–5210.

[12] S. Burger, K. Weilhammer, F. Schiel, and H. G. Tillmann, Verbmobil: Foundations of Speech-to-Speech Translation. Springer Berlin Heidelberg, 2000, ch. Verbmobil Data Collection and Annotation, pp. 537–549.

[13] Julia Dembowski, Michael Wiegand, and Dietrich Klakow. Language independent named entity recognition using distant supervision. In Human Language Technologies as a Challenge for Computer Science and Linguistics. Proceedings of the 8th Language & Technology Conference, Poznań, Poland, November 17-19, 2017, pp. 68–72.

[14] Piotr Bojanowski and Edouard Grave and Armand Joulin and Tomas Mikolov. Enriching Word Vectors with Subword Information. In Transactions of the Association for Computational Linguistics, vol. 5, 2017, pp. 135–146.

[15] Jia, Chen, Xiaobo Liang, and Yue Zhang. Cross-Domain NER using Cross-Domain Language Modeling. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. Florence, Italy, July, 2019, pp. 2464–2474.

[16] Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of NAACL-HLT 2019, Minneapolis, Minnesota, June, 2019, pp. 4171–4186.