LREC 2020 Industry Track
Language Resources and Evaluation Conference
11–16 May 2020

Industry Track

PROCEEDINGS

Khalid Choukri (ELRA/ELDA, France), Bente Maegaard (University of Copenhagen, Denmark),
Nicoletta Calzolari (Institute for Computational Linguistics «A. Zampolli», CNR, Italy; ELRA, France),
(eds.)

Proceedings of the LREC 2020 Industry Track

Edited by: Khalid Choukri, Bente Maegaard, and Nicoletta Calzolari

ISBN: 979-10-95546-71-9
EAN: 9791095546719

For more information:
European Language Resources Association (ELRA)
9, rue des Cordelières
75013 Paris
France
http://www.elra.info
Email: lrec@elda.org

Introduction

These are the proceedings of the Industry track at LREC 2020. As stated in the track description[1], Human Language Technologies (HLT) have reached a high level of maturity to become increasingly important parts of our lives. These technologies have emerged from decades of collaborations between academic and industrial research organizations often with financial and research support from the public sector; collaborations are made possible by the unique strengths of both communities and a set of shared practices (algorithms, evaluation methods and the corresponding challenges, language resources, and the like). But despite this, there are substantial differences between research in academic and industrial settings.

In contrast to academic research, industrial speech and language technologies pose unique challenges of scale when real applications are to be deployed on the market; language resources from industry may imply different sizes, demand different algorithms or validation and quality control methodologies than in academic settings; and the practices of academic and industrial settings may converge on distinct methods for the same problem; industrial systems and practices may pose ethical challenges not necessarily present in academic settings.

The initial plan was to have one full conference day dedicated to addressing the industrial R&D in the HLT domains we target, reviewing the current state of the art and future trends. Unfortunately, because of the COVID-19 pandemic, LREC 2020 had to be cancelled, as well as the full industry track day. We considered that it was difficult to go virtual for such an interactive day which by nature should trigger discussions and debates between the participants. In particular since the intention was that the technical presentations were to be backed up by an exhibition set-up within the traditional HLT village organized by ELRA.

To pay tribute to the authors who have submitted papers, the Industry Track Committee has also decided to publish these papers that have been thoroughly examined by the Track organizers.

The organization of an industry track will continue at the next LREC, in 2022, and the current authors will be invited to re-submit updates of 2020 activities if appropriate which then can be examined by a larger committee assessing their added value with respect to state of the art.

The Organizing Committee:

Khalid Choukri (ELRA/ELDA, France)
Bente Maegaard (University of Copenhagen, Denmark)
Nicoletta Calzolari (Institute for Computational Linguistics «A. Zampolli», CNR, Italy, & ELRA, France)

[1] https://lrec2020.lrec-conf.org/en/conference-programme/industry-track/

Organizers:

Khalid Choukri (ELRA/ELDA, France)
Bente Maegaard (University of Copenhagen, Denmark)
Nicoletta Calzolari (Institute for Computational Linguistics «A. Zampolli», CNR, Italy, ELRA, France)

Table of Contents

Proceedings of the LREC2020 Industry Track , pages 1–5
Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020

Spoken Medical Prescription Acquisition Through a Dialogue System on Smartphone: Perspective of a Healthcare Software Company

Ali Can Kocabiyikoglu[†], François Portet[⋆], Jean-Marc Babouchkine[†], Hervé Blanchon[⋆]
[†] Calystene SA, 38320 Eybens, France
[⋆]Univ. Grenoble Alpes, CNRS, Grenoble INP-LIG F-38000 Grenoble France
{a.kocabiyikoglu, jm.babouchkine}@calystene.com
{francois.portet, herve.blanchon}@imag.fr

Abstract

Industrial Medical Practice Management Software (PMS) have appeared in health institutions to reduce medication errors which affect several million people worldwide each year. However, practitioners must enter information manually into PMS which decreases the time devoted to care. In this paper, we describe the approach and some experiments of the implementation of an initial spoken dialogue system in this low-resourced domain in an industry-oriented setting. The main objective is to provide a natural language interface as an alternative to typing prescriptions in PMS. We highlight some of the difficulties of using deep-learning systems in an industrial context and discuss how these systems could be used while enabling a full traceability. To overcome the lack of annotated speech data, we present a way to generate aligned data for machine-learning systems and discuss the limitations of using artificial data generation. We report on the findings of human evaluation conducted on the initial prototype with 2 medical experts and 2 naive users and discuss the results of each module of the dialogue system from an industrial perspective.

Keywords: Spoken Dialogue Systems, Natural Language Understanding, Health Informatics, Natural Language Processing

1. Introduction

Mobile applications, internet of things, big data and digital health are affecting permanently healthcare domain and all of its actors. In this transformation process, health information systems are becoming increasingly complex as a result of the diversity of the digital tools available to patients and the healthcare professionals. After the earlier dialogue systems that simulated conversations between doctor-patients (Weizenbaum, 1966; Colby et al., 1971) and then that focused on commercial interactions such as advanced traveler information systems (Bobrow et al., 1977; Price, 1990), there has been an increasing interest on dialogue systems developed for health-related purposes. While Calystene SA is proposing healthcare informatics solutions for hospitals since 1992 in France, following this trend, we also have taken interest in dialogue systems as a part of our R&D program.

If most industrial dialogue systems have been based on some sort of expert rule-based models (Wallace, 2009), recent research on dialogue systems focused on neural approaches for dialogue systems (Wen et al., 2017; Ultes et al., 2017; Williams et al., 2017a) and led to end-to-end dialogue systems trainable by using a dataset of textual human dialogues (Ultes et al., 2017; D'Haro et al., 2020). Availability of deep learning frameworks and large datasets allowed training of such systems. Hence, for task-oriented dialogue systems, instead of creating domain specific semantics for each new task, these systems learn an intermediate semantic representation from data which is supposed to fit better the needs of the task.

However, from an industrial perspective, the goal is also to build agile, predictable, sustainable and scalable software. Hence, statistical approaches must provide control over the pipeline. Especially for the healthcare domain, without using any intermediate control mechanism, inferring directly an output from an input representation can have serious consequences.

Developing a dialogue system in the health care domain implies specific requirements, in particular to meet the standards of security and confidentiality of health data, but also the standards of healthcare software conformity and effectiveness relating to the use medical practice software. In this paper, we share our experience of the modeling process of a hybrid dialogue system which uses recent approaches for NLU and dialogue management while incorporating rule-based software PMS which would allow a control mechanism for the medical domain.

2. Overview of the Dialogue System

A classical dialogue architecture is based on a modular system where each component is responsible for a specific task (Williams et al., 2016). The different components of a dialogue system are the following : automatic speech recognition (ASR), spoken language understanding (SLU), dialogue state tracking (DST), dialogue policy, natural language generation (NLG) and text-to-speech (TTS). Since our objective is to deliver a prototype as a mobile application, we have to take into consideration several aspects regarding the human-computer interaction and other details which are not related to the Natural Language Processing (NLP) domain.

We approach the medical prescription understanding problem as a dialogue task in which the utterance initiated by the user must be understood, disambiguated and completed through goal oriented dialogue. This way, missing information about the prescription could be completed through dialogue turns. When connected to a patient profile, prescription assistance software could warn the practitioner for potential adverse drug events (ADE) or adverse drug interactions, etc. The example described Figure 1 illustrates this strategy.

(1) **Spoken Language Understanding**
Prescriber: Metoprolol 200 mg one-half tablet once a day at night during dinner

(2) **Disambiguation and Information Filling:**
(METOPROLOL 200 mg, coated tablets, extended-release, route oral) (freq-ut: everyday,freq-startdate: immediately)

(3) **Requesting precision from the prescriber:**
System: Please specify the duration of the prescription
Prescriber: For one week

(4) **Proposition of a structured prescription:**
METOPROLOL 200 mg, tablets, route of administration oral. One-half tablet at night during dinner, starting from today for 1 week. Do you confirm?

(5) **Checking for drug interactions and patient history:**
System: Contraindication detected, the patient's file shows that the patient has had arthritis. Do you want to add this prescription to the patient's file?
Prescriber: No

Figure 1: Overview of the general approach

3. Challenges & Approach

Most of the dialogue systems created for health purposes focus on preventive health and medical data collection of patients, especially in the context of mental health problems (Fitzpatrick et al., 2017) which raises a lot of concerns in terms of privacy, ethics and effectiveness of the proposed solutions.

3.1. Low-resourced domain

One of the major challenges for biomedical NLP is the lack of freely available datasets for developing machine learning algorithms. There are very few datasets that are used in NLP domain which are composed of natural language medical prescription. For example, the challenges i2b2, which took place in 2009, involved extraction medication information from electronic health records (Uzuner et al., 2010). As a part of this competition, 696 health records were released with 17 documents annotated by medical experts and 251 documents annotated by the scientific community.
Another dataset which is widely used in the medical domain is the MIMIC-III corpus which is an extension of MIMIC-II providing a massive amount of data also used by the NLP community (Johnson et al., 2016). However, this dataset is not annotated medical prescription-wise.
Another issue in NLP is that for languages other than English, the situation is even worse, since the only paper we found was (Deléger et al., 2010), who applied techniques used for the i2b2 Shared Task to a French dataset extracted from 17,412 French EHRs. This dataset cannot be made available. We are not aware of any speech dataset related to medical prescription that would be available to the community.
For a spoken dialogue system, a corpus of dialogues is required for training and evaluating the systems. Since we were not able to find any corpus for this task, we approached the dialogue modeling by using interactive learning (Bocklisch et al., 2017). Regarding the SLU aspect, our approach for generating initial training data was to extract medical prescriptions from a medical textbook and to complement it with prescriptions generated by a context-free grammar. Although the artificial data generation techniques lacks of naturalness, it is capable of providing a large coverage (cf. (Kocabiyikoglu et al., 2019)).

3.2. ASR and NLU Quality in the Pipeline

Most frequent use of speech recognition in clinical context has been speech recognition software for dictation (Kumah-Crystal et al., 2018). However, most of the earlier ASR systems were abandoned quickly due to recognition errors (Blackley et al., 2019). Recent systems transcribe conversational medical speech with around 20% Word Error Rate (WER) (Edwards et al., 2017; Chiu et al., 2017). Most of these systems are using cloud platforms which raises questions about the privacy of the data, especially knowing the fact that when proposing a solution to a client, the client is not necessarily aware about how the data is processed.
For a modular dialogue system, since the output of a module is feed as input into another module, errors appearing in the earlier modules of the system is propagated through other components and result in poor performances (Li et al., 2017). This is why the ASR stage and NLU stage must be as error-free as possible. Even though ASR systems are prone to errors, a spoken dialogue system can benefit from human intelligence by implementing implicit repetitions and visual confirmation of information on the smartphone application. Also, for a given utterance when the confidence level of the system is low, dialogue systems could implement fallback actions such as explicit confirmation to make sure that recognized utterances are validated by the user.
However, creating an ASR system for health-related purposes is a complex process and require good quality conversational speech data. For demonstration purposes, we have started using the cloud based speech recognition system available on Android smartphone. A recent study on major ASR services on medical conversational data show that Google ASR produce on average 40% WER on healthcare related conversational speech (Kodish-Wachs et al., 2018). For medical prescriptions, the entries are more clear and concise which allowed the use of an ASR cloud engine for demonstration purposes. However, in order to scale the dialogue system at production-level, one should take into account the performance of ASR and the impact on the NLU system. Ideally, before scaling process, it would be interesting to test toolkits such as Kaldi (Povey et al., 2011) which can run the ASR system locally even on Android platform or to explore end-to-end SLU system (Desot et al., 2019).

3.3. Medical Components of the Pipeline

Healthcare applications are subject to strict conformity checks often required by governments. In USA, the HIPAA

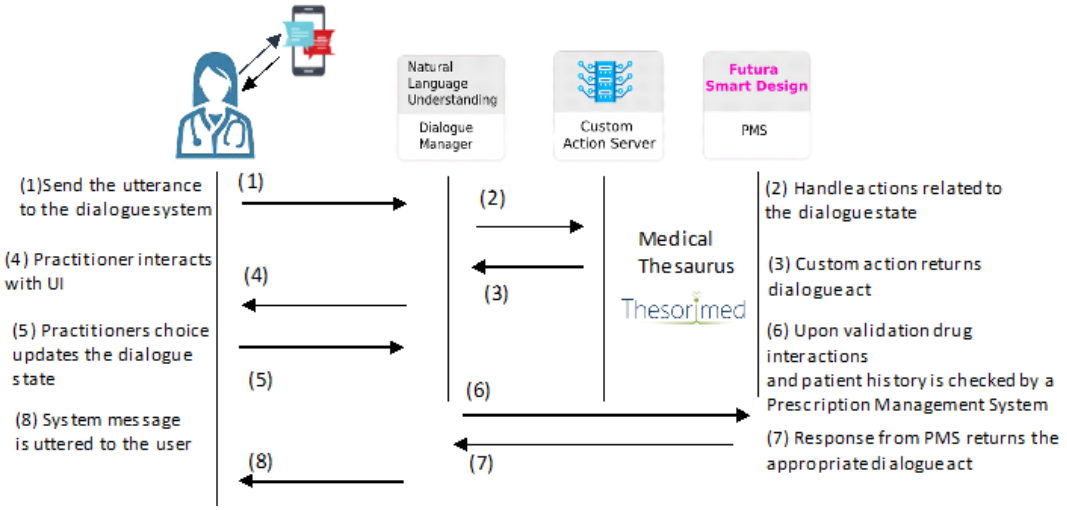


Figure 2: Pipeline of dialogue servers and external services

(Edemekong and Haydel, 2019) requires all healthcare facilities to implement strict rules to protect the confidentiality and integrity of patient information. In a similar way, in France, medical applications should be certified by National Authority for Health (HAS).

One of the crucial aspects of conformity is to provide a seamless flow of information with medical knowledge bases. Healthcare databases are updated regularly (daily or weekly) and provide information about drugs and their interactions. For example, in the case of refusal of a marketing authorization from the government, medical applications should be capable of modifying the applications by disabling the prescription of some drugs.

Our PMS, Futura Smart Design® is certified by HAS and is used at over 50 health care facility in France. Our knowledge base, is based on Thesorimed®, a large databases on drugs, which is updated regularly by the French National Health Insurance (Ameli). We are planning two levels of verification to ensure a secure information validation process. In order to do so, the processing pipeline involves different levels of verification using several medical knowledge bases. The overview of the different components that incorporates expert knowledge is illustrated Figure 2.

At first (1), when a prescription intent is inferred from an utterance, slot-fillers are extracted (2) to be associated to common dispensation codes (UCD) of drugs (3). If no drugs could be associated from the extracted semantic frames, system suggests to restart the prescription process. If there are more than one potential drugs corresponding to semantic frames, the user is provided with a list of drugs to choose and the process continue until all missing information is inferred (4 and 5). At the second stage, when the necessary prescription information is complete, we plan to send this structured data to PMS (6). For a given patient file, PMS handles the validation process of the prescription and give information about drug interactions, patient allergies, and so on (7 and 8). Until the prescriber validates the prescription or cancel it.

3.4. Training Procedure and Traceability

Over the last decade, systems based on deep neural networks surpassed other methods in most of the existing literature in many fields. Availability of large datasets, reduction of costs of cloud computing platforms and continuously evolving deep learning frameworks allowed this change especially in academic studies. Compared to academia, the adoption of fully statistical methods have been somewhat slower in industry. The reason is once a dialogue system is deployed, it should be a part of continuous integration (CI) and continuous deployment (CD) at any point. However, the process for developing, deploying, and continuously improving deep neural networks is more complex compared to traditional software (Sculley et al., 2015).

In domains where the output of a system can have serious consequences, a system should be fully tracable. In case of an error, one should be able to pinpoint easily the source of an error. Decoupling tasks in a modular system allows this partly by creating logs of each process.

In industry, it's not uncommon to 'hack' a solution by manually modifying a behavior in a software or correcting an entry in the database when there is a specific need coming from a client. A neural network is often viewed as a black box in the sense that while it can approximate any function, its structure does not provide any insight of the function being approximated. Thus, quick-fix solutions and even adjusting the system according to new data requires the retraining of the whole system. Their behavior is often complex and hard to predict, harder to test, explain and maintain (Sculley et al., 2015). A hybrid approach tackles this problem by allowing to train/adapt only the module that is concerned by the changes (Williams et al., 2017b). We are using the freely available Rasa X [1] toolset which allows this by providing a simple web interface that allows to modify the NLU, dialog scenarios and the domain definition from a web interface.

[1] http://www.rasa.com/

	Task Sucess Rate	Average Dialogue Turns	NLU (f-measure)	WER (ASR)	Drug Association Rate	Average Time Elapsed
medical experts	45%	1.56	0.75	3.40%	0.62	30 seconds
naive users	16.6%	1.54	0.43	17.35%	0.65	35 seconds

Table 1: Results of the human evaluation of the dialogue system

4. Experimental Results

In order to have an early feedback about the prototype, we performed an human-based experiment. This experiment had a double objective: collect speech dialogue corpus in French using the dialogue system and evaluate the dialogue modeling which extends our previous work of the evaluation of our NLU system (Kocabiyikoglu et al., 2019). For the evaluation process, we have contacted two medical experts and two naive users for prescribing medicine using our mobile dialogue system.

Prescribing medicine is not an easy task for a naive user, even when the information to utter is provided. Therefore, we have prepared two procedures, one for medical experts and another for naive users. To avoid reading behavior from the experts, they were given a textbook of therapeutics which presents clinical cases and for which medical prescription is presented in a non-natural language way (B. Gay, 2009). Thus they had to abstract the prescription before uttering it. For naive users, however, another therapeutics textbook for students in medicine was given (Perrot, 2015). All prescriptions information was explicitly presented so they did not need medical knowledge. Hence, reading behavior could not be avoided.

In order to challenge the system and obtain various examples, the textbook prescriptions were ranked according to their complexity. Each participant had to make 10 medical prescriptions using the mobile application a headset and microphone in a silent room. For medical experts, they had to try at least two challenging examples. In total, 40 dialogue were collected from 2 medical experts and 2 naive users with 10 prescription each. The implementation process does not include the interactions with the PMS and focused on the prescription and drug association.

Results of the evaluation are summarized in the table 1. The global task success rate (ratio of validated prescriptions) which describes if the prescription has been completed or not is low for both medical experts and naive users.

It can be seen that for medical expert, the ASR Word Error Rate (WER) is very good while the NLU stage exhibits a fair f-measure of 0.75 which stays in line with our previous study on our NLU performance (Kocabiyikoglu et al., 2019). The picture is far less good regarding naive users.

A behavior which is common for both type of users is the low dialogue turn (about 1.5). In fact, the dialogue stopped quickly because the system often responded "drug not found". This is illustrated the low score of Drug Association rate (about 40% of error). Another problem was due to the difficulty of recognizing frequency (e.g., every weeks) and duration (e.g., for the next two weeks). These elements have been reported as been difficult to extract in the i2b2 challenges as well (Uzuner et al., 2010) and are due to a the lack of training data of intermediate interactions such as precising the duration or the dosage of the prescription.

In case the drug is associated correctly, the prescription process takes around 20-30 seconds which is quite reasonable with respect to the typing procedure.

Overall, this quick evaluation shows that although some components gives satisfying performances the overall pipeline is not robust enough for the expert (the target user of the product). The evaluation of the modular architecture permits to identify the necessary improvement such as the NLU component (needs to be trained with examples for precising duration and frequency of the prescription) the drug identification as well as the dialogue management. Indeed, the system has been trained using cooperative scenarios. However, a good number of dialogues entered a fallback loop because of the lack of uncooperative scenarios.

The evaluation also showed that including non expert in the process emphasize the difference in behavior and language with the expert. Indeed, the poor performance of the naive users were mainly due to some formulations that were less technical and more familiar which differed significantly from most of the training examples. Although very limited in size, this evaluation shows the importance of performing experiment including target users.

5. Conclusion

This paper presents an approach to make oral medical prescription possible for an industrial prototype through dialogue on a smart phone. We discussed the trade-off between fully statistical end-to-end systems and the need for control, maintenance, traceability and privacy in a real industrial setting. An initial working pipeline prototype has been developed using a mixture of inference models acquired by machine learning, expert system and professional knowledge bases. The evaluation of this initial prototype showed the importance of keeping a modular architecture to identify the components that need improvement and emphasized the dependence of machine learning technique on data; data that are very often unavailable in industry setting. This call for more research in machine learning to develop methods which could benefit both from expert knowledge and a reduced amount of data.

References

B. Gay, P.-L. Druais, P. A. T.-D. (2009). *Thérapeutique en médecine générale*. apnet.

Blackley, S. V., Huynh, J., Wang, L., Korach, Z., and Zhou, L. (2019). Speech recognition for clinical documentation from 1990 to 2018: a systematic review. *Journal of the American Medical Informatics Association*, 26(4):324–338.

Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., and Winograd, T. (1977). Gus, a frame-driven dialog system. *Artificial intelligence*, 8(2):155–173.

Bocklisch, T., Faulkner, J., Pawlowski, N., and Nichol, A. (2017). Rasa: Open source language understanding and dialogue management. *arXiv preprint arXiv:1712.05181*.

Chiu, C.-C., Tripathi, A., Chou, K., Co, C., Jaitly, N., Jaunzeikare, D., Kannan, A., Nguyen, P., Sak, H., Sankar, A., et al. (2017). Speech recognition for medical conversations. *arXiv preprint arXiv:1711.07274*.

Colby, K. M., Weber, S., and Hilf, F. D. (1971). Artificial paranoia. *Artificial Intelligence*, 2(1):1–25.

Deléger, L., Grouin, C., and Zweigenbaum, P. (2010). Extracting medication information from French clinical texts. In *MEDINFO 2010*, pages 949–953, Amsterdam.

Desot, T., Portet, F., and Vacher, M. (2019). SLU for voice command in smart home: comparison of pipeline and end-to-end approaches. In *IEEE Automatic Speech Recognition and Understanding Workshop*, Singapore.

D'Haro, L. F., Yoshino, K., Hori, C., Marks, T. K., Polymenakos, L., Kummerfeld, J. K., Galley, M., and Gao, X. (2020). Overview of the seventh dialog system technology challenge: Dstc7. *Computer Speech & Language*, page 101068.

Edemekong, P. F. and Haydel, M. J. (2019). Health insurance portability and accountability act (hipaa). In *StatPearls [Internet]*. StatPearls Publishing.

Edwards, E., Salloum, W., Finley, G. P., Fone, J., Cardiff, G., Miller, M., and Suendermann-Oeft, D. (2017). Medical speech recognition: reaching parity with humans. In *International Conference on Speech and Computer*, pages 512–524. Springer.

Fitzpatrick, K. K., Darcy, A., and Vierhile, M. (2017). Delivering cognitive behavior therapy to young adults with symptoms of depression and anxiety using a fully automated conversational agent (woebot): a randomized controlled trial. *JMIR mental health*, 4(2):e19.

Johnson, A. E., Pollard, T. J., Shen, L., Li-wei, H. L., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L. A., and Mark, R. G. (2016). Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035.

Kocabiyikoglu, A. C., Portet, F., Blanchon, H., and Babouchkine, J.-M. (2019). Towards spoken medical prescription understanding. In *2019 International Conference on Speech Technology and Human-Computer Dialogue (SpeD)*, pages 1–8. IEEE.

Kodish-Wachs, J., Agassi, E., Kenny III, P., and Overhage, J. M. (2018). A systematic comparison of contemporary automatic speech recognition engines for conversational clinical speech. In *AMIA Annual Symposium Proceedings*, volume 2018, page 683. American Medical Informatics Association.

Kumah-Crystal, Y. A., Pirtle, C. J., Whyte, H. M., Goode, E. S., Anders, S. H., and Lehmann, C. U. (2018). Electronic health record interactions through voice: a review. *Applied clinical informatics*, 9(03):541–552.

Li, X., Chen, Y.-N., Li, L., Gao, J., and Celikyilmaz, A. (2017). Investigation of language understanding impact for reinforcement learning based dialogue systems. *arXiv preprint arXiv:1703.07055*.

Perrot, S. (2015). *Thérapeutique pratique 2015*. MEDLINE.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., et al. (2011). The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*.

Price, P. (1990). Evaluation of spoken language systems: The atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., and Dennison, D. (2015). Hidden technical debt in machine learning systems. In *Advances in neural information processing systems*, pages 2503–2511.

Ultes, S., Barahona, L. M. R., Su, P.-H., Vandyke, D., Kim, D., Casanueva, I., Budzianowski, P., Mrkšić, N., Wen, T.-H., Gasic, M., et al. (2017). Pydial: A multi-domain statistical dialogue system toolkit. In *Proceedings of ACL 2017, System Demonstrations*, pages 73–78.

Uzuner, Ö., Solti, I., and Cadag, E. (2010). Extracting medication information from clinical text. *Journal of the American Medical Informatics Association*, 17(5):514–518.

Wallace, R. S. (2009). The anatomy of a.l.i.c.e. In Robert Epstein, et al., editors, *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer*, pages 181–210. Springer Netherlands.

Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.

Wen, T.-H., Vandyke, D., Mrkšić, N., Gašić, M., Rojas-Barahona, L. M., Su, P.-H., Ultes, S., and Young, S. (2017). A network-based end-to-end trainable task-oriented dialogue system. In *EACL 2017*, pages 438–449, Valencia, Spain.

Williams, J., Raux, A., and Henderson, M. (2016). The dialog state tracking challenge series: A review. *Dialogue & Discourse*, 7(3):4–33.

Williams, J. D., Asadi, K., and Zweig, G. (2017a). Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. In *ACL 2017*, pages 665–677, Vancouver, Canada.

Williams, J. D., Asadi, K., and Zweig, G. (2017b). Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. *arXiv preprint arXiv:1702.03274*.

Proceedings of the LREC2020 Industry Track , pages 6–9
Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020

Promises and Disappointments of Semantic Analysis of Speech-To-Text Applied to Call Center Conversations in an Industrial Setting

Ruslan Kalitvianski, Emmanuelle Dusserre, Muntsa Padró
ELOQUANT
5 allée de Palestine, 38610 Gières
{ruslan.kalitvianski, emmanuelle.dusserre, muntsa.padro}@eloquant.com

Abstract

Recent progresses in speech-to-text technologies, and the marketing hype they create, lead to believe that speech-to-text has become robust enough for reliable semantic analysis of textual transcripts of phone conversations to have a chance to be feasible. This paper describes our experience as a mature provider of semantic analysis of written text in French with analysis of uncorrected machine transcripts of real-life call center two-speaker conversations. We argue that, although much progress has indeed been made, many difficulties remain before French conversation transcripts can be exploited to their full potential.

Keywords: speech-to-text, semantic analysis, opinion mining, classification, call center

1. Introduction

Our team has been developing commercial solutions for semantic analysis for text in French for many years. We propose machine learning and expert-based systems for generic and bespoke multilabel classification, opinion extraction and topic modelling. Since the acquisition of our start-up by a company specialized on customer relations management solutions, the services we provide are shaped on the direction of extracting information from the different channels we propose to our client companies to communicate with their customers.

One of these channels, and probably the most important today for us, is the voice channel. Our company offers a platform for managing call centers with multichannel interactions (phone, emails, chat…), phone calls being nowadays more than 80% of the communications managed. Thus, the amount of data generated from the conversations of our customers with their costumers is huge. To enhance the service we give to our customers, it is a natural step to experiment with the use machine analysis of conversation recordings to automatically determine which topics are being discussed, how the conversation is evolving, what are the emotions expressed by the customer, etc.

There is a profusion of providers of speech-to-text (STT) solutions on the market for the French language. Nuance, Vocapia, Bertin IT, Allo-Media, IBM Watson, Google Speech-to-Text, to name some. A few are specialized in phone conversations, but most offer various models trained for different purposes.

We are aware of numerous works on deep learning-based architectures that have considerably advanced the state of the art in speech transcription, but in this paper, we are interested in mature commercial solutions, being agnostic of the underlying technologies.

This paper describes our experience as an experienced provider of semantic analysis for French with analysis of uncorrected machine transcripts of real-life call center two-speaker conversations. First, we describe the rationale that a company such as ours could have for using semantic analysis of machine transcripts, then we describe the difficulties and obstacles we encountered when trying to apply our technology on this data.

2. Promises of STT: Massive analysis of an abundant resource

Call centers abound with recordings of conversations, which represent a big wealth that companies could exploit to better serve their customers and improve the efficacity of their contact centers. Nevertheless, these data are not as readily exploitable as written text, therefore good STT systems are crucial for the development of these applications.

In what follows we present some interesting tasks that could be performed with a quality speech transcription.

2.1 What STT providers can do

Other than raw speech-to-flow-of-words transcription, many STT systems provide other features:

- "Speaker diarization is the task of determining "who spoke when"" (Anguera et al., 2010). Most commercial STT providers include automatic or semi-automatic identification of the number of speakers in a conversation and attribution of speech turns, performing a kind of speech segmentation, which is useful when the audio is single channel (mono). This task may also involve speaker gender identification.
- Word timestamping and word transcription confidence scores: some systems (Vocapia, Bertin IT) assign a timestamp to each transcribed token. This allows to align tokens with the audio, a useful feature when developing an interactive audio player that displays the transcription. More expert users may benefit from the confidence score $\in [0, 1]$ that these systems assign to each token. Other STT providers (Nuance) can output a lattice of transcription candidates which allows NLP experts to select the best transcription path using a custom language model.
- Several systems (Nuance, Vocapia, Bertin, Google STT) can incorporate custom vocabularies for correct transcriptions of persons' names, product names, places, etc. Some accept a simple list of words, other allow to specify a category for each custom word (product name, locality, etc).
- A few systems (Google, Vocapia) have begun adding automatic punctuation to their output, which renders the transcript more natural to read.

2.2 Indexation

The most basic use for speech transcripts is term indexation of the audio. This allows searching among and within the conversations for mentions of specific terms. This is a need that has been expressed by some of our customers during our brainstorming workshops.

2.3 Semantic analysis

We provide Web platforms that allow to graphically display results of semantic analyses on an interactive dashboard (Dusserre *et al.*, 2020). One can thus combine results of different analyses to select, for example, opinions expressed about one or several products or topics. The purpose of these platforms is to provide our clients with an overview of the vocal interactions between their agents and their customers, since the volumes of the data, coupled with the time it takes to listen to a call, vastly exceed the humanly analyzable. Thus, a more ambitious use is machine analysis of the transcripts for:

- Speaker role identification: based on the diarization performed by STT, determine who of the speakers is the client and who is the agent. In a general setting, one cannot systematically say, for instance, that the first person to speak is the agent that is replying to an incoming call. An analysis of what is said is necessary to determine who is who in the conversation. This will allow both filtering in the Web dashboards, and analyses tailored to each type of speaker.
- Call topic extraction: extract the topics addressed in each call, either for statistical analysis over large collections of calls (to determine the most frequent topics and their diachronic evolution) or to classify the call (i.e. for indexing, routing, etc.). This task can be approached in two ways:
  i. Unsupervised topic modelling: given a significant amount of calls, extract the most frequent topics that are relevant to the current domain. This allows for an evolutive modelling of the calls for each costumer and to discover new topics.
  ii. Supervised classification: given a predefined set of categories, use a classifier to determine the topics of a call. This allows to track over time a set of categories that we know are important for the customer/domain.
- Opinion mining: extract feedback given by the customer (positive or negative opinions, action requests, threats, etc.). This would allow to perform real-time alerting about unhappy customers, study the opinions related to each topic or category, etc.
- Domain-specific named entity recognition: detect the mentions of a given product or specific entity to gain insights into its popularity, understand the satisfaction level related to it, etc.
- Automatic summary generation and action item detection: this has been attempted by the CALO Meeting Assistant Project (Tur *et al.*, 2010) for English, and, more recently by the REUs project for French (Patel *et al.*, 2019).

2.4 Good/bad practice identification

Our client companies' agents say that the mere fact of reading one's own conversation transcript al-lows them to better themselves by an a posteriori analysis of how an interaction.

An advanced semantic analysis may also help identify the specific rhetoric used by the agent that allowed him or her to close a deal, or, on the opposite side, to see what went wrong in an interaction.

Many companies have scripts that their agents must adhere to (for instance mentioning a product's name N times during an advertisement), and semantic analysis could be used to check for script conformity.

2.5 Combining NLP extracted information with structured data

Information extracted from audio can be combined with that extracted from other data sources, which can be especially useful in a multichannel customer relation management platform. Some examples of other data can be event information (how long was the conversation, did the customer already contact the company before, etc.), customer information (age, gender, etc.), closed questions answers (e.g. satisfaction note the customer gave in a survey), among others. Thus, we can imagine several applications of combining these kind of data with data extracted from conversations: profiling of customers and agents to assign them to the best suited agent, prediction of satisfaction rate given the contents of the conversations, computing automatically first call resolution rate, etc.

3. Difficulties

Our first attempt at semantic analysis of conversation transcripts was for a client company that routinely recorded their pre-sale and customer support conversations via the several call center platforms they employ. These are conversations recorded in France, whose length ranged from less than a minute to over 40 minutes long.

We agreed to transcribe conversations that were between 1 and 10 minutes long and transcribed over 10000 such audios over a period of one year. All audios are single channel, 8KHz, 64 Kbit/second, encoded with G711.

3.1 Choice of STT provider

The first obstacle was selecting the best provider of STT for French. We had to construct a gold standard of transcription on the data of the client, a process that is very time-consuming: on average, transcribing 4,5 minutes of audio required an hour of work per person.

The task was distributed to 20 participants which allowed us to collect 90 minutes of transcribed speech. Instead of making our participants work from scratch, we chose to give them machine transcripts from a promising STT tool, that they would correct via a text editing tool. *A posteriori*, all participants felt that having a machine "pre-transcript" was useful, and accelerated the transcription process.

3.1.1 The issue of evaluation

We chose to evaluate several systems using the word accuracy metric (WAcc = 1 - Word Error Rate[1]). We normalized input texts by stripping punctuations (if any), lowercasing words, removing some disfluency markers

[1] https://www.vocapia.com/glossary.html

("euh", "hum", etc.) as well as newlines. The texts to be compared are therefore streams of words. We chose not to evaluate speaker diarization, because was of lesser priority.

We evaluated a total of ten models of five leading STT providers. Given that the conversations centered on the client's products (heating appliances), we built a lexicon of client-specific terms, and used it as a parameter for systems that accepted vocabulary customization.

On twenty audios of 4.5 minutes each, the lowest observed average word accuracy rate was 0.47 and the highest was 0.73. The median WAcc was 0.696. The highest average WAcc was obtained with a system that used a phone conversation model supplemented with the client's vocabulary.

It is important to note that merely focusing on average performance occults the performance of the system in the best and worst cases, and, more generally the variability in performance. Thus, the system with the best average WAcc transcribed only 10% of the audios with a WAcc at or above 0.8, whereas the second-best transcribed 15% of the audios with a WAcc at or above that threshold, but conversely it produced less transcriptions that reached the 0.7 WAcc threshold.

3.1.2 Lexical fidelity vs task usefulness

Lexical fidelity may not be the best or the only measure of the performance of a STT system. We decided to perform and extrinsic evaluation by comparing the results of our analyses on the gold standard transcriptions with those on the automatic transcriptions using the two best STT providers with their best configuration. We evaluated three analyses that produce sets of words or categories:

- Concept extraction, which extracts nouns and nominal groups that are significant to the domain using a terminology extraction system based on Sclano and Velardi (2007).
- Categories, which are client-specific themes, based on a ML classifier (which uses as features the word, the lemma, noun phrases, and semantic annotations coming from domain-specific gazetteers), and hand-written rules based on the TokensRegex formalism (Chang and Manning, 2014).
- Domain-specific Named Entities, which are simply a list of our client's products and their synonyms.

Table 1 demonstrates that a system that performs better at transcription (average WAcc = 0.73) may reveal itself to be less useful for semantic analysis than a system with a slightly worse performance (average WAcc = 0.699).

		F-measure		
		Concepts	Categories	Domain NEs
Best system	Average	**0.51**	0.79	0.69
	Median	**0.52**	0.85	0.73
Second-best system	Average	0.50	**0.86**	**0.83**
	Median	0.48	**0.92**	**0.86**

Table 1: performance of three semantic analyses on machine transcripts by the two best systems (as determined by their average WAcc), with human transcripts analyses as reference.

These results show that, paradoxically, analyses performed on the output of the best system show less agreement with analyses of human transcripts of the same audios than analyses of the output of the second-best system. The discrepancy is likely due to differences in the abilities of these systems in integrating customer-specific vocabulary into the transcription model. More generally, that shows the importance of performing this kind of evaluation on top of core task evaluations, since they may reveal that the most useful systems for industry purposes may not be the ones that have best scores on paper.

3.2 Insufficient audio quality

Real-time telephony is a low-latency transmission whose only requirement is human intelligibility of speech. Signal is deformed at many steps, including capture (low quality microphones), encoding, transmission (loss of packets), mixing (fusion of two speaker channels into one).

Moreover, the system we selected based on best average performance does not attempt automatic punctuation, which results in a flow of words that seem odd when read, even if correctly transcribed, partly because they lack prosodic features such as pauses.

Thus, the indexing and reading promise of STT appears only partially fulfilled. Reading a transcript is often a tedious task, because of the reasons above, and because one must guess what words were actually uttered when one reads an erroneous portion, which is a cognitively demanding task.

3.3 Insufficient transcription quality

Upon subjective evaluation of transcription quality, it turned out that a WAcc of 0.7 corresponded to a rather low-quality transcription. Most of the correct words were trivial words, there were portions that were not transcribed, often at the beginning of a speech act.

Moreover, the system we selected based on best average performance does not attempt automatic punctuation, which results in a flow of words that seem odd when read, even if correctly transcribed, partly because they lack prosodic features such as pauses.

Thus, the indexing and reading promise of STT appears only partially fulfilled. Reading a transcript is often a tedious task, because of the reasons above, and because one must guess what words were actually uttered when one reads an erroneous portion.

3.3.1 Speaker diarization

This is a necessary step when one has mono audio (as we did) and wants to build a speaker classification system to determine which speaker is the client and which one is the agent. The two typical errors we observe with speaker diarization are:

- Incorrect identification of the number of speakers: in our context two is the correct number, but sometimes more are found, or, when the voices of the two speakers are similar, only one. Some systems can take the maximum number of speakers as a parameter, others cannot.
- Incorrect speech turn boundary placement, which attributes words to the wrong speaker, making the

transcript less comprehensible and confusing the speaker role identification system.

3.3.2 Impact of noise and errors

As mentioned before, there are many errors on the automatic transcriptions. This means that the semantic analysis systems that we build to extract information have to be very robust to these errors. This implies that it is very challenging to develop rule-based or Machine Learning systems that correctly model a very variable set of contexts. Here, we must deal with the already rich and variable spoken language with noise added by the STT system.

Furthermore, French is especially difficult to analyze because of its high degree of homophony. It is crucial to have a good language model, and to adapt it to each domain, since each customer will have a set of specific words (names of products, for example) that are unknown by the system or that appear in unexpected contexts. In our experience, this lexical and contextual tuning are mandatory to improve the quality and, most importantly, task usefulness of the transcriptions. Adding words is an option for many STT system, but it would also be useful if words that are observed in machine transcriptions and are unlikely in the client's context could be manually removed from the model's vocabulary.

3.4 Analyzing text vs analyzing speech

A difficulty that is not necessarily related to the quality of the transcription is the vast difference between written one-time comments and spoken conversations.

3.4.1 Disfluencies of speech

We use linguist-defined rules that describe specific lexical and syntactic patterns to detect opinions. Spoken language is much less precise than written comments, and it is much harder to write patterns that match discourse filled with disfluencies, interruptions and transcription errors.

3.4.2 Loss of prosody

Much emotional information is conveyed by prosody, which could be used to increase confidence in opinions detected in the text. Thus, to correctly detect opinions, studying not only lexical items, but also the prosody seems mandatory. None of the systems we tried supplied this information.

3.4.3 Expressions of opinions

Usual sources for opinion mining are surveys or reviews that are rich on opinionated expressions since this is their purpose. Detecting opinions in an oral conversation is much more difficult, for several reasons:

- As mentioned, a much emotional information is conveyed by prosody, not on the language itself.
- The purpose of many conversations is often to factually discuss topics, not necessarily to express satisfaction or unsatisfaction. Thus, the opinions are found much less frequently, mixed with a lot of other information, and often expressed implicitly.
- Given the oral nature of the speech and the errors from STT systems, the sentences are rarely syntactically complete, thus, opinion extraction systems based on syntactic patterns only rarely match.
- Expressions such as "good", "very well" or "ok" often imply an opinion in written reviews but are used as mere acquiescence in phone conversations.

4. Conclusion

As a call center management software experts and NLP experts, we are highly interested on exploit-ing STT solutions to process the big amount of data that our customers produce. We believe that the call center software of the future will allow companies to better serve their customers by automatizing as much as possible the study of the needs, expectations, and satisfaction of the customers.

The promise of STT systems is to accurately transform audios into written texts, which would al-low their browsing, indexing and detailed analysis.

In this paper, we have presented some of the applications that we can imagine of such a transcription and analysis, but also the current limitations discovered in practice when applying current state-of-the-art commercial solutions to our real-world data. Namely, even if we can extract some information as the topics of a conversation, the quality of the transcription is often not good enough to be read without listening to the audio.

In our domain, we observed two leverages that can improve the quality of STT systems: the quality of the audio, which in our case is limited by the phone lines, and the use of in-domain lexica to tune the STT models. These are, of course, known issues for researchers, but what is sometimes ignored is the difficulty in real life to obtain (or create) audio in good quality and good lexical resources.

The lessons learnt from our experiences is that big attention has to be given to these factors before setting up a project, and that there is still a long way to go to be able to fully analyze call conversations, but we remain optimistic about the possibility of reaching this goal and to industrializing se-mantic analysis of this channel.

5. Bibliographical References

Anguera, X., Bozonnet, S., Evans, N., Fredouille, C., Friedland, G., & Vinyals, O. (2012). Speaker diarization: A review of recent research. *IEEE Transactions on Audio, Speech, and Language Processing*, *20*(2), 356-370.

Chang, A. X., & Manning, C. D.: TokensRegex: Defining cascaded regular expres-sions over tokens. *Technical Report CSTR 2014-02*. Department of Computer Science, Stanford University (2014)

Dusserre, E., Kalitvianski, R., Ruhlmann, M., & Padró, M. (2020). Analyse sémantique de transcriptions automatiques d'appels téléphoniques en français. *Actes de la 6e conférence conjointe JEP, TALN, RÉCITAL, Volume 4: Démonstrations et résumés d'articles internationaux*.

Patel, N., Lannes, M., & Pradel, C. (2019, July). Patrons linguistiques pour l'extraction de tâches dans des transcriptions de réunions. In *Actes de PFIA 2019, pp. 158–166*. PFIA.

Sclano, F., & Velardi, P.: TermExtractor: a Web Application to Learn the Shared Terminology of Emergent Web Communities. In: R. J. Gonçalves, J. P. Müller, K. Mertins, M. Zelm (Éd.): Enterprise Interoperability II, pp. 287-290. Springer London (2007)

Tur, G., Stolcke, A., Voss, L., Peters, S., Hakkani-Tur, D., Dowding, J., ... & Frederickson, C. (2010). The CALO meeting assistant system. *IEEE Transactions on Audio, Speech, and Language Processing*, *18*(6), 1601-1611.

Proceedings of the LREC2020 Industry Track , pages 10–14
Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020

Industrial ASR Troubleshooting Tool

Oleksandr Solop, Filip Sawicki, Andrzej Jeziorski, Marcin Sikora
Michał Junczyk, Tomasz Zietkiewicz
Samsung Research Institute Poland (SRPOL)

Abstract

During the development of an industrial scale Automatic Speech Recognition (ASR) system significant quantities of evaluation data have to be analyzed in order to make positive adjustments to the ASR subsystems. The scale and variety of the evaluation data, as well as the complexity of the ASR system architecture makes both manual improvements and automatic model boosting a challenging task. We present a system created to facilitate more effective improvement of the ASR recognition rate with the help of a web tool that provides aggregation, quality review and statistical analysis of ASR performance evaluation data in a human readable format. This paper presents an overview of the system and some exemplary use cases.

Keywords: ASR, error rate, tokenization, alignment, data pipeline, evaluation data

1. Introduction

Evaluation and development of industrial scale, machine learning based systems like Automatic Speech Recognition (ASR) is a challenging task due to quickly evolving product requirements and dependencies between data and system components, often with hidden feedback loops. ASR systems are evaluated in an iterative manner and as a result produce large amounts of evaluation data, which is hard to explore manually. One of the approaches to increase the efficiency of development is to focus on the issues which most significantly impact performance.

The ASR Troubleshooting Web Tool is made to address the above challenge. The tool aggregates tokens extracted from evaluation data and produces a pool of words that have a high probability of being attributed to a common error. These tokens are called ”troublemakers”. ASR experts can use the tool to mark issues which are hard to automatically detect and suggest corrections to models by monitoring top troublemakers in each ASR system version.

2. Literature Review

The need exists for industrial systems that continuously improve ASR, as seen in the pending US patent by call analytics firm Marchex Incorporated. The patent specifies a system that automatically provides faster and more accurate ASR for telephone calls. In addition to automated model refinement the cited patent provides a human interface for the manual review of transcriptions. Our system not only provides efficient way to detect quality issues in test data, but also provides an extended view of evaluation results thanks to data statistics, aggregated around tokens. (Marchex, Inc., 2018)

In more general terms, our work addresses problems related to machine learning pipelines deployed in a production environments. Many academic publication have been made on the topic. For example D. Sculley et al. specify how technical debt creeps it's way into machine learning systems. (Sculley et al., 2015) Our solution mitigates some of the technical debt issues listed in the article, e.g. the increased cost of analysing an individual model due to it's dependency on models preceding it in the pipe. B. Nushi et al. go more in depth into the problem of troubleshooting component-based machine learning systems and propose a human-in-the-loop approach to model refinement.(Nushi et al., 2017) In their case study on Automated Image Captioning they rely on crowdsourcing microtasks to fine-tune components. Our approach provides a boarder set of information and allows experts to make sweeping adjustments to the pipeline elements.

3. Materials and Methods

3.1. ASR evaluation data overview

An ASR test case (ASR TC) is a pair consisting of an audio/speech recording and a corresponding transcription. An organized collection of ASR TCs in a standardized format, with relevant meta-data (e.g. language, domain, acoustic conditions etc.) constitutes an ASR database (ASRDB). An Automatic Speech Recognition (ASR) system generates an orthographic text representation (hypothesis) to an audio input (audio recording). The quality of an ASR system is evaluated by comparing the similarity of the hypothesis generated for a given audio recording to a manually created transcription (reference) of the corresponding

recording, across a given set of ASR TCs. Depending on the ASR subsystem being evaluated, either a basic reference or a normalized reference is used. The basic reference contains only tokens and is used to evaluate the performance of Language Models (LM), while the normalized reference is used for Inverse Text Normalizer (ITN) evaluation, therefore it should resemble text written by humans and respect language specific normalization rules related to numeric expressions formatting, spelling of named entities, punctuation etc.
In order to generate ASR hypotheses resembling text written by humans a combination of various modules is used (see next section).

3.2. ASR system overview

The ASR system is comprised of a long short-term memory (LSTM) Acoustic Model (AM), Grapheme-to-Phoneme (G2P) model, n-gram Language Model (LM), optional n-gram Personal Language Model (PLM) and Inverse Text Normalizer (ITN) post-processing modules. All the models have been trained using *proprietary* data. An in-house built decoder has been used to generate 1-best hypothesis. The hypothesis is then processed by different ITN submodules, including:

- Preprocessor - removes tags inserted by decoder (such as start and end of utterance tags), normalizes casing by the decoder output
- Inverse Number Normalizer (INN) - performs inverse normalization of numbers by replacing numerals with appropriate numbers.
- Fixers - perform string substitutions, either ones defined by language experts as regular expressions patterns or performed automatically, by statistical models trained on ASR evaluation data.

ITN modules are connected in the form of a pipeline. Each of them can be selectively turned off. During the ASR evaluation ITN is tested multiple times with different modules turned on and off, to check them separately and provide a way to spot the exact cause of possible errors. This, together with the PLM module also being turned on and off, leads to production of different ASR hypotheses. ITN specification in the industrial ASR system needs to be quickly adapted to evolving user needs, language conventions, market trends and system architecture. Given the inter-subjective nature of language, hidden/complex organizational dependencies with sometimes conflicting requirements, it's necessary to manage the ITN functional requirements as a common "ASR text normalization protocol", available to all stakeholders, including product owners, ASR developers, testers and language experts responsible for creating references.

3.3. Data Aggregation

In order to provide a human readable view of the complex ASR evaluation pipeline, the system uses data from ASRDB evaluation sets and corresponding ASR hypotheses generated with various system settings. If a hypothesis has a word recognition rate of $x_{wrr} = 1$ compared to the normalized reference (ref-n), then it is considered to be correct, else there is a mismatch and the pair is marked as incorrect.

The Troublemaker Tool was designed using the ETL (Extract, Transform, Load) concept to generate the best data pipeline performance using available resources. The system saves only structured, filtered and polished outcomes instead of large amounts of ASR test results in various formats.

System ETL steps
1. Data loading, validation and filtration
2. Collection of mismatched tokens
3. Calculation of aggregated statistics on tokens
4. Extraction of troublemakers
5. Comparison of mismatched utterance pairs
6. Concordance and stability metric preparation
7. Saving processed data to database

During the data processing stage, the ETL system collects and counts all tokens that appear in mismatched utterance. These tokens are grouped together and used to calculate aggregated statistics. Based on the gathered metrics our system selects troublesome tokens that have both high fail frequency and the greatest influence on ASR performance. Fixing errors in tokens with the highest combination of total count, stability and frequency yields the most significant improvements in ASR. We have created several metrics to make detection of these tokens easier (see section "Troublemaker Metrics").

To make an ASR improvement suggestion with the Troublemaker, the reference and hypothesis has to be compared to find differences. Our tool makes use of a sequence matching algorithm to reject equivalent tokens from both sentences. A pairwise sequence alignment algorithm is then applied to find the best matching tokens based on the received similarity score. Moreover the entropy value for each token is calculated. It shows how spread out the error is and thus gives information on how meaningful the potential fix would be.

3.4. Troublemaker Metrics

TokenCount: the number of ASR TCs containing a given token. This metric helps to prioritize work by providing information about the "troublemaker" token's impact based it's popularity. x is equal to 1 if the token is present in the sentence i and 0 otherwise. n is the total number of ASR TCs in the evaluation set.

$$\text{TokenCount} = \sum_{i}^{n} x_i$$

FailsCount: the number of incorrect hypotheses related to the token. Incorrect hypotheses are those with a word recognition rate $wrr < 1$. $x_{i;wrr<1}$ is equal to 1 when the token in present in the sentence and the sentence is from a failing ASR reference-hypothesis pair.

$$\text{FailsCount} = \sum_{i} x_{i;wrr<1}$$

Frequency: Ratio of incorrect hypotheses to all ASR TCs containing the token. Shows an estimate of how often the token is failing.

$$\text{Frequency} = \frac{\text{FailsCount}}{\text{TokenCount}}$$

Entropy: Measure of the level of uncertainty associated with a given token. F is the Frequency of the token failing.

$$\text{Entropy} = -\left(\text{F} \cdot \log(\text{F}) + (1 - \text{F}) \cdot \log(1 - \text{F})\right)$$

Fail Coefficient: a modified frequency metric that reduces the number of most common tokens shown and include less frequent ones. Higher values indicates bigger impact on ASR system performance. User can sort troublemakers by this field to work with the most important defects first.

$$\text{Fcoeff} = \text{Frequency} \cdot \log(\text{TokenCount})$$

FCoeffR: the logarithm of the token's rank among all of the tokens based on Fcoeff values.

EntropyR: the logarithm of the token's rank among all of the tokens based on Entropy values.

Wrank: a weighted rank based on a combination of Fail Coefficient and Entropy ranks. The metrics tries to balance between token frequency, replacement stability and overall incorrectness. It is used as a main metric for comparison of different ASR system versions. The lower the rank, the more the token is worthy of the user's attention. A_1 and A_2 are adjustable weight constants that effect the ratio with which FCoeff and Entropy effect Wrank.

$$\text{WRank} = (A_1 \cdot \text{FCoeffR}) \cdot (A_2 \cdot \text{EntropyR})$$

4. Triage Process

Every ASR data report contains a change rate based on the aggregated troublemakers *Wrank* metric, which makes it easy to compare performance of any pair of ASR system versions. This simplifies results analysis, because it allows the review of data by portions.

Language experts have access to multiple tables describing top tokens per hypothesis. These tables include all metrics, which are used to locate troublemakers that are the most cost-efficient to fix, and the number of reported issues related to them. Moreover, our tool can represent two kinds of troublemakers: the reference troublemakers (tokens, which ASR rarely recognizes correctly) and hypothesis troublemakers (tokens, which ASR outputs too often). This error type separation and issue detection prevents the ASR system from over-fitting and under-fitting on these troublemakers.

After the user selects a troublemaker to work with, the concordance view opens, enabling the checking of details about a troublemaker's word stability, for example all mismatched tokens with their probabilities.The concordance view also contains a pool of random utterance pairs (reference and hypothesis) containing the selected troublemaker. A single example row consists of information about WRR, a button for audio recording playback and comparisons of hypotheses from other stages of the ASR pipeline. All troublemaker words and potential replacement words are presented in a joint column to facilitate the process of issue analysis.

Reference	wrr	Hypothesis
The cat sat on the mat	0.77	The dog sat on the mat
The big cat on the mat	0.65	The big dog sat on the mat
	...	

Figure 1: Concordance view example.

An expert can report issues related to the troublemaker token using the report panel. In this panel, one can quickly assign the troublemaker to several categories, depending on the issue type. Additionally one can provide a detailed description of the problem and tag it with other relevant tokens. Different issue categories are then automatically routed to appropriate developer teams which are specialized in fixing the particular problems. An exemplary set of problem classes for a typical ASR system has been described below. This set can differ vastly, depending on the ASR system architecture.

Audio: a recording may or may not meet quality requirements, depending on the purpose of a particular ASRDB. For example, audio with an above normal

distortion level, can be reported if present in a regression test set which supposed to contain only clean audio data. In other contexts it can be a useful example of a real usage scenario. Other fairly common issues are excessive cross-talk, incomplete or unintelligible speech.

Reference: since ASRDBs references are prepared by language experts, they are prone to human errors. The typical problems are: incorrect normalization, typos and the unnecessary transcription of background speech.

AM: poor recognition in certain acoustic conditions, systematically substituted tokens with similar pronunciation or sensitivity to background speech – these all can be attributed to a problem with AM.

Reference	wrr	Hypothesis
le président de l'égypte	0.25	les présidents de l'égypte
président de france	0.67	présidents de france
...		

Figure 2: Concordance AM issue example

Grapheme-to-Phoneme (G2P): improper phonetization can lead to errors, especially when it comes to Named Entities (NE).

Reference	wrr	Hypothesis
hi bixby	0	hi pixy
bixby was ist das für ein lied	0.33	xd was ist das für ein lied
...		

Figure 3: Concordance G2P issue example

LM: a broad category, which can be as simple as a lexicon typo, or more complex, signaling a broader issue with NEs, problems with number recognition, and many others.

Reference	wrr	Hypothesis
ponme un álbum de camilo sesto	0.83	ponme un album de camilo sesto
ponme un álbum de joe dassin	0.83	ponme un album de joe dassin
...		

Figure 4: Concordance LM issue example

ITN: a fairly well defined category consisting of general normalization, capitalization and punctuation issues.

Reference	wrr	Hypothesis
playstation 4	0	play station cuatro
enciende la play playstation	0.33	enciende la play play station
...		

Figure 5: Concordance ITN issue example

5. System architecture

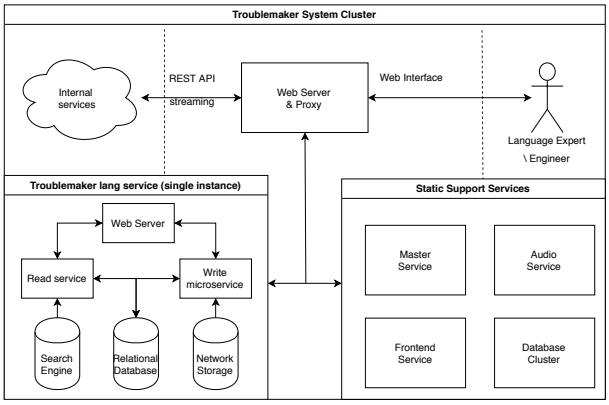


Figure 6: System architecture.

5.1. Microservice architecture

The Troublemaker Tool follows a microservices design pattern. In front of the system sits a web server that acts as an API gateway, reverse proxy, offloader and load balancer for incoming REST requests. Next there are two main groups of services. Static Support Services serve as gateway aggregators, health checkers, frontend interfaces, data management systems and provide other common utilities, like audio streaming, logging and issue reporting. Scalable Language Services consist of multiple instances of read and write microservices deployed for one specific language.

6. Conclusion

The presented system aids the development of industrial scale ASR by extracting ”troublemaking” tokens that most frequently cause errors in the ASR system or reference data, presenting information in a convenient format for ASR experts and streamlining the generation of reports. Given the early stage of development, we’re not able to measure the impact of the system on the process in a quantitative way yet. However the qualitative feedback received so far is encouraging to pursue further development and research in this area.

7. Bibliographical References

Marchex, Inc. (2018). Automatic speech recognition (asr) model training. US Patent 20180315417.

Nushi, B., Kamar, E., Horvitz, E., and Kossmann, D. (2017). On human intellect and machine failures: Troubleshooting integrative machine learning systems. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., and Dennison, D. (2015). Hidden

technical debt in machine learning systems. In C. Cortes, et al., editors, *Advances in Neural Information Processing Systems 28*, pages 2503–2511. Curran Associates, Inc.

Proceedings of the LREC2020 Industry Track , pages 15–20
Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020

Measuring the Polarity of Conversations between Chatbots and Humans: a use Case in the Banking Sector

Guillaume Le Noé-Bienvenu, Damien Nouvel, Djamel Mostefa
OrangeBank, Inalco, OrangeBank
guillaume.lenoe.bienvenu@gmail.com, damien.nouvel@inalco.fr, djamel.mostefa@orangebank.com

Abstract

This paper describes a study on opinion analysis applied to both human to chatbot conversations, but also to human to human conversations with data coming from the banking sector. Applying a polarity classifier model to conversations provides insights and visualisations of the satisfaction of users at a given time and its evolution. We also conducted a study on the evolution of the opinion on the conversations started with the chatbot and then transferred to a human agent. This work illustrates how opinion analysis techniques can be applied to improve the user experience of the customers but also detect topics that generate frustrations with a chatbot but also with human experts.

Keywords: Text Mining, Opinion Analysis, Chatbot, Polarity

1. Introduction

1.1. Scope and Aim

Orange Bank is a mobile bank launched in late 2017 and for which the main channel of communication with its customers is Djingo, a text chatbot. Available 24/7 by chat, Djingo, Orange Bank virtual advisor, is the customers first point of contact. Since the launch of Orange Bank in November 2017, more than 2,5 million conversations have been initiated by our clients with Djingo (an average of 100,000 conversations per month), 50% of which are handled entirely by the virtual advisor (without any redirection to the Customer Relationship Centre). Since Djingo is the first point of contact of Orange Bank clients, all chat conversations with a human agent started with Djingo. We are hence able to measure the evolution of the polarity within the same conversation between a customer and Djingo and then between the customer and the human operator.

In this context, opinion mining may be used to deliver in real time an understanding of the customer relationship for a given service. It could also be used to detect annoyance, irritation or angriness at an early stage of the conversation with Djingo in order to quickly redirect the user to a human expert. In this situation, opinion mining is also useful to detect topics and to provide insights about customer's satisfaction.

Our work focuses on the evolution of customer's opinion, both on conversations or messages within conversation. We implemented an opinion detector that has been evaluated, and plugged into the history of online conversations between customers and chatbot or human support desk. This work provides the customer support service visualisations of the evolution of customer's satisfaction depending on themes, as well as information on how much the bot and humans give satisfaction to the customers.

1.2. State of the Art

1.2.1. Opinion Analysis

Whereas a lot of work has been done in the opinion analysis field, most of it was directed towards product reviews, e.g. identifying the sentiment linked to the aspects of an object or its entities (Liu, 2012), but a few work was done towards written conversations, especially with a chatbot. (Hancock et al., 2019) used the estimation of user satisfaction to improve the learning process of the chatbot. Tools to work on polarity and emotions based on rules such as VADER (Hutto and Gilbert, 2014) or SentiWordNet (Esuli and Sebastiani, 2006) are freely usable, but remain only for the English language. For French, resources are also available, such as the CANÉPHORE Corpus (Lark et al., 2015), but remain mostly specific to tweets. In this paper, we present a few cases (mostly graphs) in which opinion analysis could help giving valuable information with written talks. We focus on the polarity, defined by Zhang and Ferrari (2014) as the property of a text being positive, negative or neutral.

1.2.2. Text Classification

Text classification is a well known task in NLP, and a reasonably efficient technique to perform it consists of using a TF-IDF (Salton and Buckley, 1988) representation of the data combined with a support vector machine classifier (SVM) on it. This approach has since be giving satisfactory results. (Joachims, 1998; Pang et al., 2002; Lilleberg et al., 2015) Deep learning methods can also be used for text classification. In particular, convolutional neural networks obtain very high scores for text classification (Kim, 2014), but require more time and examples for training. Also, the winners of many challenges in NLP for the French language used TF-IDF+SVM models as the one used for DEFT 2015 (Thierry Hamon, 2015) or during the Hackatal 2018 [1]).

1.3. The Djingo Chatbot

Djingo is Orange Bank's conversational agent, available 24/7 for its 3,000 daily users. It is able to understand 390 intentions and has more than 1,000 answers adapted to the user's needs. Djingo is used both as a Frequently Asked Questions (FAQs) system (products marketed e.g. withdrawal fees, time to deliver a cheque book, etc.) and as an assistant to perform actions related to the customer account (ordering a cheque book, blocking the card, etc.). FAQ-oriented answers are usually the same for all customers, whereas requests performing an action trigger an operation

[1] https://hackatal.github.io/2018/

that depends on the account.

For example, if a user wishes to order a checkbook, Djingo will check if the user is identified, if there is currently no checkbook order, if the user can order it, and so on. At each step, depending on the elements received through a programmatic interface (APIs), Djingo provides the user with an appropriate answer. During the conversation, themes and intentions are detected by the IBM Watson module. To date, there are about 60 themes: Orange-Bank, app-site-info, app-site-problem, insurance-info, termination insurance, etc. Conversations can include several themes. If the user asks a question that Djingo does not have the answer to, or detects that the user is unable to make himself understood, he suggests that the user should be redirected to an advisor.

2. Opinions for messages and conversations

2.1. Chatbot Corpus

The corpus used in this article consists of 1,566,060 unique conversations from November 2017 to March 2019, containing 5,775,227 messages. Most of the messages sent by the users contain a small number of words (around 4.6 words per message) and are often describing the question using simple words. The size of the lexicon is quite important with around 144k entries due to important number of misspellings and typos.

2.2. Annotation

As we focus on the polarity of messages, we built a gold-standard, by manually annotating 3,053 randomly picked user messages from the corpus. Each message is considered as positive, negative or neutral, following the 2015 DEFT annotation guide (Thierry Hamon, 2015).

The annotation was made by two different annotators, giving a Cohen's kappa coefficient of 0.72. One particular issue during the annotation process was the case of greeting messages. We notice that in our data set, the user uses greetings for 83.96% of the conversations with a human agent, and only 18.99% of those with the chatbot. This gives us a clear indication of the behaviour of the user depending on the interlocutor. From an opinion perspective, we then assumed those greetings were positive and annotated them accordingly.

Table 1 gives examples of annotated data.

Unsurprisingly, the annotations are unbalanced: 5.01% of the messages are positive, 73.96% of them neutral and 21.03% negative. This was expected as users usually come with problems and questions regarding bank services and operations. Indeed, the company wants to maximise the satisfaction of users at the end of the interaction, while limiting the number of agents hired for this task.

2.3. Classification

This annotated data set was then divided over a train (4/5) and test parts (1/5). The train data was then pre-processed by computing a TF-IDF transformation. We tested several classical machine learning models using the sklearn API (Buitinck et al., 2013). Results are reported in Table 2.

Message (*translated*)	Annotation
Merci orange pour les 80 euros *Thank you orange for the 80 euros*	positive
Merci, bonne soirée *Thank you, have a nice evening*	positive
OK, super ! *Okay, great!*	positive
Je souhaiterai ouvrir un compte *I'd like you register an account*	neutral
Savoir si ma demande a été traitée *Find out if my request has been processed*	neutral
Quelles sont vos offres pour les étudiants ? *What are your offers for students?*	neutral
Cela ne repond pas a la question *This doesn't anwser the question*	negative
Non merci je suis très contrariée *No, thanks, I'm very upset.*	negative
Vous servez à rien *You're useless.*	negative

Table 1: Example of annotated messages

ML classifier	**Precision**	**Recall**	**F1**
SVM	0.90	**0.81**	**0.85**
MaxEnt	**0.92**	0.75	0.82
Nultinomial Naive Bayes	**0.92**	0.63	0.70
SGDClassifier	0.91	0.79	0.84

Table 2: Performance of Opinion Classifier (macro)

As the SVM classifier provides the best F1 score, we ran a grid search on several parameters to optimize this model configuration. We obtained an average 0.85 F1 macro score (0.91 F1 micro). The neutral class obtains the best score (0.95 F1), while positive and negative classes have much lower F1 scores (0.82 and 0.76, respectively). Those results were obtained using the NLTK TweetTokenizer (Bird et al., 2009), without any other preprocessing (no lemmatization, case is kept as it is) and linear kernel for the SVM. Finally, the model was used to classify all messages of the corpus.

3. Conversation Polarity by Themes

3.1. Rules to Predict Conversations Polarity

To have a global view of user experience, one needs to compute an opinion score for each conversation. As the data was annotated by messages, simple rules were implemented to predict the polarity of an entire conversation based on the opinion of its messages. A conversation is then:

- **neutral** when all messages are such,
- **positive** when at least one of its messages is such and the remaining is neutral or positive,
- **negative** when at least one of its messages is such and the remaining is neutral or negative,
- **mixed** otherwise.

	Number of messages	%	Number of conversations	%
Positive	460,744	3.98	190,057	7.30
Neutral	9,903,323	85.50	1,746,296	67.07
Negative	1,218,890	10.52	541,549	20.80
Mixed	_	_	125,641	4.83
Total	1,1582,957	100	2,603,543	100

Table 3: Proportion of messages and conversations in the corpus

Using these simple rules, table 3 shows the proportion of messages and conversations in the corpus. These rules allowed us incidentally to get strongly oriented conversations (e.g. a conversation where nearly all of its messages are negative would be very negative).

3.2. Histogram

The first representation we get from this labelling is the proportions of the conversation classes (positive, negative, neutral and mixed) depending of the detected themes. Figure 3 (annex) shows those proportions for December 2018. For instance, *the app_site* theme (related to the behaviour of the Bank's application) has more than 50% of its conversations being negative where the *cheque* theme remains globally neutral, this can be explained by the fact that this operation is rarely problematic. The representation of polarity gives us a rough idea of where to improve the user's experience. This type of plot can also be drawn for a different time scale (year, day, etc.).

3.3. Heatmap

In the previous section, we presented a way of drawing the proportions of the conversation classes for a particular timelapse. However, this type of plot does not give us information about the evolution of this proportions across a time scale. E.g. on Figure 3, the *app_site* theme has a strong part of negative conversations but one can wonder if those proportions were similar through the year, whether it was due to a temporary failure, or if it was a general trend.
In order to represent a potential evolution of those proportions, we proposed a heatmap showing this evolution of the opinion by theme. To get a polarity score as a single numerical value for each case, a rule was implemented, consisting of adding the neutral and positive proportions of conversation and subtracting the negative. This was given by the following formula:

$$PS(th,t) = \frac{N(neu,th,t) + N(pos,th,t) - N(neg,th,t)}{NTotalConversations(th,t)}$$

Where

- *th*: the theme of the conversation
- *t*: a date
- *N(pol, th, t)*: the number of polarity (pol as negative, positive or neutral) conversations of the theme *th* at time *t*

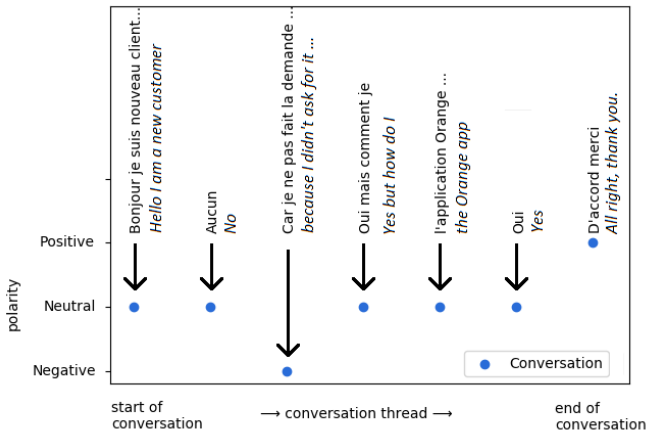


Figure 1: Single Conversation Polarity Graph

- *NTotalConversations(th, t)*: the total number of conversations of the theme *th* at time *t*

Figure 4 (annex) reports the heat map from November 2017 to March 2019. The bluer the case is the higher proportion of positive conversations the corresponding theme has. Conversely the red cases indicate negative conversations. One can then watch the changes in the proportions of cases throughout the months. For instance, we clearly see that the *Bonus* theme in March 2018 had its lowest polarity score, but its polarity score increased in the next few months. As in the previous section, this plot can also be drawn for a different time scale.

3.4. Graph of Polarity

We have been then studied the way polarity of messages changes for a single conversation, especially when the user switches from a chatbot to an agent. In order to have a visual output, we converted the polarity (negative, neutral, positive) of each message of the conversation to an integer (0 for negative, 1 for neutral, 2 for positive). This provides us with a list of integers that we can plot on a basic polarity graph, as reported in Figure 1.
Since the conversations do not have the same length (different number of user messages), we converted the lists of integers representing the polarity of the user messages into lists of floats of fixed size. The size of the output lists can be modified as an optional parameter.[2] We then compute the average of each point of the list. Figure 2 show the result of the output with a padding of dimension 20.
On Figure 2, we first notice that for both types of users (redirected and non-redirected or full IA), the conversation starts with the same polarity (neutral) on average. After the first third of the conversation, people who are not redirected see the polarity of their conversation stagnate around a value slightly below neutral, while people who will be redirected see the polarity of their conversation decrease until an agent takes over. As soon as people are cared for by

[2]Code available at `https://github.com/GuillaumeLNB/perso/blob/master/rounding.py`

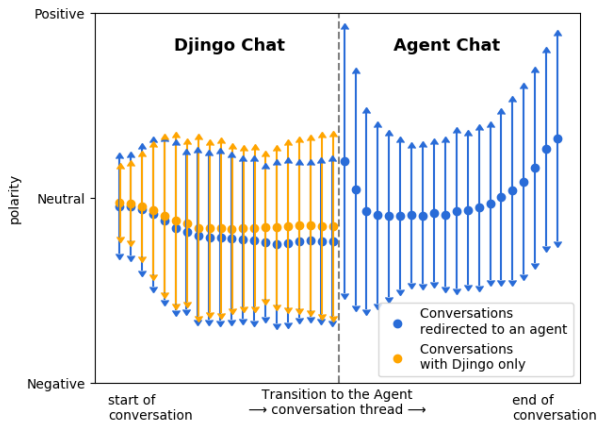


Figure 2: Polarity graph

a counsellor, the polarity of the conversation takes a more positive trend (signs of politeness such as "hello" are labelled as positive and are more present in conversations with a human being). This is followed by a more neutral phase, which generally corresponds to the advisor's information gathering. At the end of the conversation, the trend is clearly becoming positive, we hypothetize that satisfying solutions are being proposed by the human agent.

4. Discussion

There are however some limitations to the approaches discussed in this paper. First of all, the classification is based on annotation, and it is quite difficult to annotate into only three polarity classes. In the example: "*Mon épouse est décédé et je souhaite réaliser une demande de succession / My wife has died and I want to make a succession request*", the user of the conversational agent reports a past event as well as the willingness to take action. However, the part "*Mon épouse est décédé / My wife died*" would have been annotated as negative, while the part "*je souhaite réaliser une demande de succession / I wish to make an estate application*" would have been annotated neutral. A new class "positive-negative mix" could have been used as in DEFT 2018 [3], but would have required a much more subtle and fine-grained annotation work.

Secondly, polarity is useful information, but does not indicate the subjectivity of the message. There is a significant difference between a user complaining about a particular Orange Bank service (example: *Ma carte bancaire ne marche pas / My credit card doesn't work*, negative polarity) and a dissatisfied user without a specific reason being stated (example: *Orange c'est vraiment de plus en plus pourri ! / Orange is really getting crap!*, negative polarity).

Thirdly, the transition from the polarity of the messages to the polarity of the conversation was carried out with a rule-based approach, creating a mixed class. This class does not take into account the intensity of certain messages. In the example in Table 4, the conversation has a mixed polarity

Message (*translated*)	Predicted Polarity
bonjour, *hello,*	positive
association loi 1901 peut elle ouvrir un compte chez vous? *Can a association loi 1901 open an account with you?*	neutral
compte + association oi 1901 *account + aossociation 1901 [l]aw*	neutral
je ne parle pas aux robots, connards *I don't talk to robots, assholes.*	negative

Table 4: Example of a conversation classified as mixed where it should have been negative

(presence of positive and negative), but remains very negative by the presence of the last message. An annotation at the level of the conversation would probably have classified this conversation as negative, but would not have made a difference between this very negative and a less negative conversation.

Finally, the thermal map display gives us an overview of the evolution of the polarity, but does not detail the reasons of this variation. In addition, we did not find a correlation for all themes between their monthly polarity scores and their redirection rates. We are wondering if this metric is suitable for comparing these data.

5. Conclusion

In this paper, we have presented several applications of opinion analysis on chatbot conversations. By developing a model for polarity analysis (positive, negative, neutral) using standard machine learning algorithms, we were able to use the data to highlight trends. A real corpus of more than 1.5 million of conversations between Orange bank customers and Djingo was used for this study. For privacy and confidential reasons, this corpus can not be shared at that time but it may be released in the future after anonymization of all personal data.

This analysis allowed us to look at which topics of the conversational agent show the most customer satisfaction or dissatisfaction, on a time scale. It also provides the opportunity to bring out very focused conversations (very positive or negative) from the corpus for educational purposes for customer relationship centre agents. Finally, this tool makes it possible to obtain a quantification of the customers' opinions on the spot.

6. Bibliographical References

Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly, Beijing.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122.

[3] https://perso.limsi.fr/pap/DEFT2018/annotation_guidelines/index.html

Esuli, A. and Sebastiani, F. (2006). Sentiwordnet: A publicly available lexical resource for opinion mining. In *In Proceedings of the 5th Conference on Language Resources and Evaluation (LREC 06*, pages 417–422.

Hancock, B., Bordes, A., Mazaré, P., and Weston, J. (2019). Learning from dialogue after deployment: Feed yourself, chatbot! *CoRR*, abs/1901.05415.

Hutto, C. J. and Gilbert, E. (2014). Vader: A parsimonious rule-based model for sentiment analysis of social media text. In Eytan Adar, et al., editors, *ICWSM*. The AAAI Press.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751.

Lark, J., Morin, E., and Peña Saldarriaga, S. (2015). Canéphore : un corpus français pour la fouille d'opinion ciblée. In *Actes de la 22e conférence sur le Traitement Automatique des Langues Naturelles*, pages 418–424, Caen, France, June. Association pour le Traitement Automatique des Langues.

Lilleberg, J., Zhu, Y., and Zhang, Y. (2015). Support vector machines and word2vec for text classification with semantic features, 07.

Liu, B. (2012). *Sentiment Analysis and Opinion Mining*. Morgan & Claypool Publishers.

Pang, B., Lee, L., and Vaithyanathan, S. (2002). Thumbs up: Sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 79–86, Stroudsburg, PA, USA. Association for Computational Linguistics.

Salton, G. and Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, August.

Thierry Hamon, Amel Fraisse, P. P. P. Z. C. G. (2015). Analyse des émotions, sentiments et opinions exprimés dans les tweets: présentation et résultats de l'édition 2015 du défi fouille de texte (deft), 06.

Zhang, L. and Ferrari, S. (2014). Intensité et polarité : un modèle opératoire articulant plusieurs travaux linguistiques. In *Langue française, /4 (num 184), p. . DOI : 10.3917/lf.184.0035.*, pages 35–54.

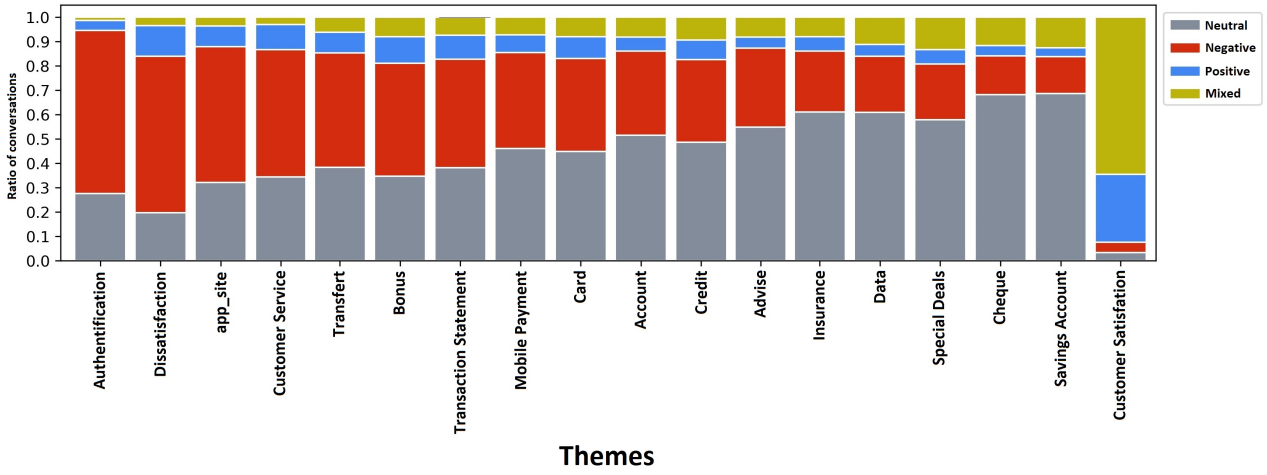


Figure 3: Basic polarity histogram

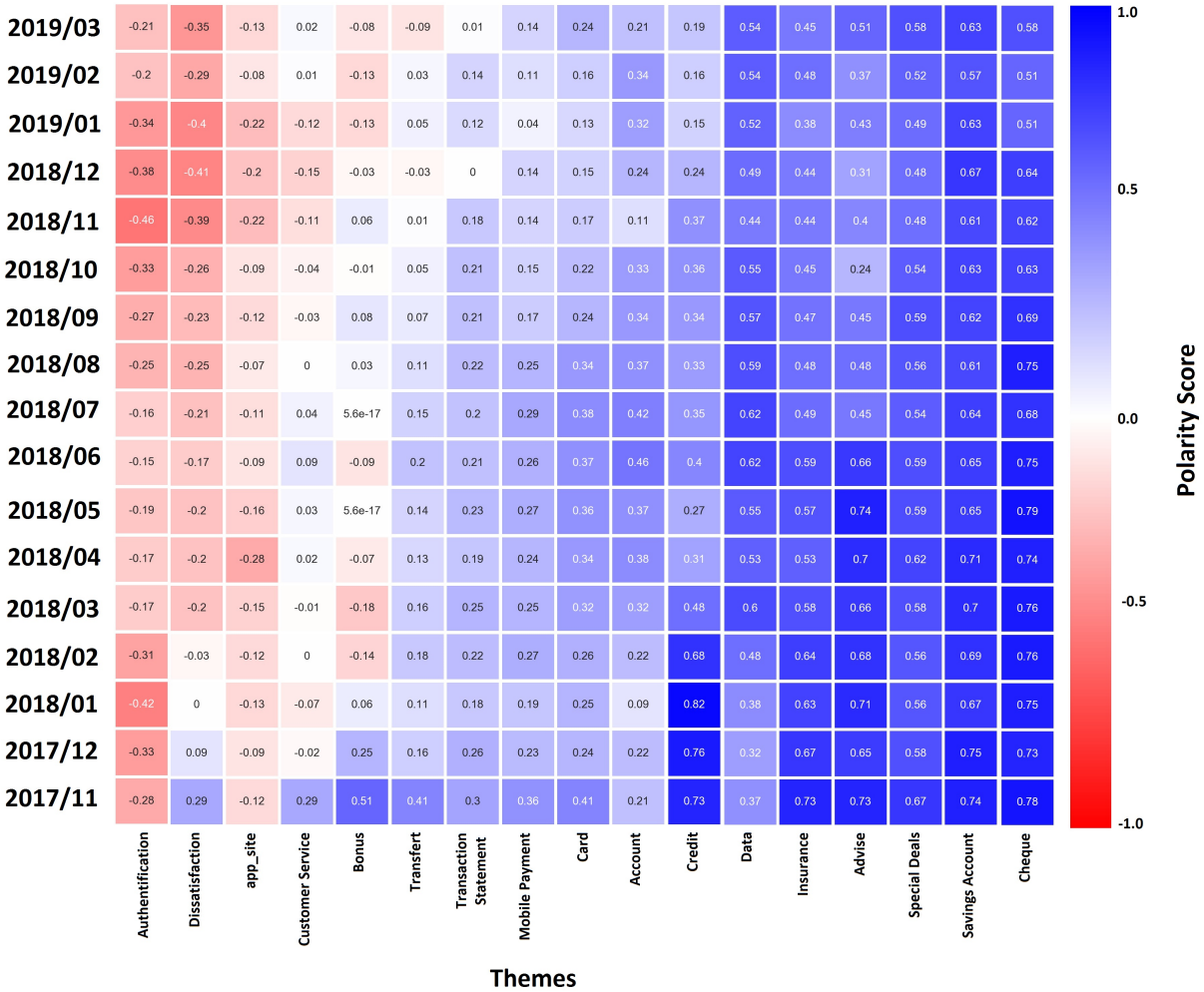


Figure 4: Heatmap of polarity

Proceedings of the LREC2020 Industry Track , pages 21–25
Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020

On the Importance of Text Classification Pipeline Components for Practical Applications: A Case Study

Andrej Švec, Katarína Benešová, Marek Šuppa
Slido
Bratislava, Slovakia
{asvec, kbenesova, msuppa}@slido.com

Abstract

The worlds of academia and industry have different priorities for machine learning models. In the academic world, the model's performance is often the main focus, whereas finding the balance between the model's performance, resource requirements, and the ease of its deployment is often deemed more important in the production environment of the industry. In this paper we consider a real world text classification problem, compare the specifics of different parts of natural language processing pipelines and investigate their contribution to the final model's performance. We also take into consideration the practical aspects of the model's use and deployment, such as the size of the model and preprocessing time. Our case-study shows that in this particular scenario the performance of simpler models can be on par with the more complex ones. We find this result valuable, as simpler and smaller models are normally also easier to deploy in practice, e.g. in a serverless environment. To showcase the practical usefulness of our final model, we deploy it to AWS Lambda and show that its execution time in this environment scales linearly with the input text's length.

Keywords: text classification, NLP pipeline, production environment, serverless

1. Introduction

Text classification is one of the most common use-case of Natural Language Processing (NLP) models. The choice of a model type and architecture varies significantly from task to task. It is also greatly affected by the requirements for the final model. In production environments, these often comprise training time, latency and resource requirements (for instance CPU, memory or model size).

Many academic and state-of-the-art works propose solutions incorporating different steps of preprocessing, data augmentation and large model architectures, rendering them time-consuming and resource-intensive to develop and train (Strubell et al., 2019). On the other hand, there is a great demand for simplicity and ease of setup in the production environment, which renders some of the academic-world solutions unsuitable for real-world use.

In this paper we compare several approaches to a specific instance of text classification. Given its practical applicability, we also investigate the deployment of the final model in a serverless environment.

1.1. Use-case prediction task

We compare these approaches in a real-world multi-class classification task. The task is to distinguish between different types of use-cases in which an online Q&A application[1] is used.

We call a single usage of the Q&A application an *event*. Thus, the task is to predict the use-case of an event by looking at the data associated with it. A use-case in this context refers to the types of events at which this Q&A system gets used, such as a conference, a team meeting or a lecture at a university.

Formerly the events used to be classified into use-cases by a set of 118 expertly chosen keyword-based regular expressions matched against the event name. This approach is based on the assumption that a keyword (for instance *conference*) in the event name implies that the Q&A system was used at a specific type of event (in this case a conference). A sample of these regular expressions is shown in Table 1. The main disadvantage of this approach lies in its low coverage (around 33%), as not every event name contains a keyword. For example, an event with the name *LREC 2020* would not be classified as *conference*, despite being an abbreviation for *Language Resources and Evaluation Conference 2020*.

Use-case	Regular expressions
Conference	`/conference/`, `/summit/`
Company meeting	`/all(\s*)hand/`, `/staff(.+)meet/`
Team meeting	`/team(\s*)meet/`, `/team(\s*)sync/`
Learning	`/workshop/`, `/l&d/`

Table 1: A sample of regular expressions used for use-case classification, based on a match against the event name.

1.2. Serverless environment

Serverless (Function-as-a-Service) environments provide a simple way of deploying code to production environments by providing an abstraction over many standard operational concerns. They are particularly suitable for stateless applications, such as small machine learning models (Ishakian et al., 2018). Furthermore, they require minimal maintenance, as the cloud infrastructure takes care of provisioning their computational resources, as well as ensuring that they only run when necessary. On the other hand, there are numerous practical limitations as to what can be currently deployed in a serverless environment. These are mainly related to stor-

[1] In our case the Q&A application is called Slido, reachable at `https://slido.com`

age (250 - 500 MB) or RAM (2 - 3 GB), forcing the models to be small in size and to also have a small RAM footprint.

2. Related work

Text classification (sometimes also referred to as text categorization) is a widely studied technique in Natural Language Processing (Aggarwal and Zhai, 2012). Due to its great versatility, numerous practical tasks can be viewed as text classification problems. This results in a wide variety of applications, ranging from linguistically-inspired problems, such as Word Sense Disambiguation (Raganato et al., 2017), classification of sentiment or opinion polarity of documents (Liu, 2015), to direct downstream applications in which text classification can be helpful, such as online suicide prevention (Desmet and Hoste, 2018).

In broad terms, the text classification pipeline can be viewed as a three stage process: at first the *pre-processing* part ensures that the inherent noise gets removed from the raw input text, then the *feature extraction* part converts the clean input text into representation that is then taken as the input of the final *classification* part, in which a classification algorithm predicts the category of the text using the provided representation.

The standard approaches in *pre-processing* relevant to the presented work include tokenization (Verma et al., 2014) and utilization of pre-trained models for extracting part of speech tags (Batanović and Bojić, 2015) as well as named entities (Trask et al., 2015) from the input text and using this information in further parts of the pipeline.

There exist a multitude of options for feature extraction, such as Bag-of-Words, TF-IDF (Jones, 1972), word embeddings, e.g. word2vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014), FastText (Bojanowski et al., 2017) and various contextualized word representations (Peters et al., 2018). A different approach to representing text is called sentence embeddings. These aim to provide semantically meaningful vector representations of sentences, comparable using cosine distance. Some of the popular methods include Skip-Thought vectors (Kiros et al., 2015), InferSent (Conneau et al., 2017) and the Universal Sentence Encoder (Cer et al., 2018).

A sizable body of work has been published on the topic of classification algorithms. Popular choices include logistic regression (Lee and Liu, 2003), Naïve Bayes Classifier (Kim et al., 2006), Support Vector Machines (Kwok, 1998) as well as various approaches based on neural-networks (Joulin et al., 2017), particularly in combination with Deep Learning approaches (LeCun et al., 2015).

3. Dataset

Our dataset is composed of 80 000 events. An event consists of various types of interactions with the Q&A application, incorporating mainly questions posted by the audience to the presenter, as well as the live polls used by the presenter to find out the opinion of the audience about a topic.

Only a subset of the interaction data has been selected for the purpose of our task. A preview of such data can be seen in Figure 1. The event data can be categorized as follows:

- **Aggregated features.** These comprise high-level information about the events, for instance the number of participants, the duration of the event or the number of posted questions and activated polls.
- **Textual features.** The texts of the questions and polls belonging to the events. All the texts are in English language. We exclude the event name from the texts to prevent the model from essentially extracting the regular expressions from the event name.

Aggregated: participants=95, n_questions=52, duration_days=1, . . .
Questions: "How much time / hours have you saved so far in these 23 processes?", "How do you pay invoices? Do you have any automated process?", . . .
Polls: "You are from", "How did you like this talk?", "What do you want me to elaborate more on?", . . .
Label: Conference

Figure 1: An example of data about event.

Text length. The length of a question is limited to 160 characters in most cases but a few events use a maximum length of up to 300 characters. The number of questions ranges from zero to a few thousands, with a median of 16. A poll consists of the question asked to the audience and either poll options or audience replies (for a multiple-choice poll or open-text poll, respectively). A poll question can have up to 256 characters. The number of polls in an event ranges from zero to a few thousand, with a median of 1.

The distribution of number of tokens per event is shown in Figure 2. It suggests that the texts we are trying to classify can easily contain hundreds of words.

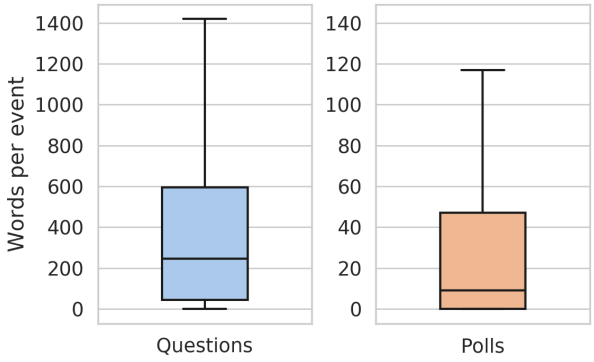


Figure 2: The distribution of the number of words in concatenated questions and polls in our dataset.

Labels. The events are assigned a use-case based on the matching regular expressions described in Section 1.1. We exclude events that are assigned more than one use-case, as they are not numerous and doing so would needlessly turn our task into multi-label classification.

Our task considers four use-cases: *company meeting* (37% of data), *conference* (31%), *learning* (17%) and *team meeting* (15%). The main deficiency of these classes is the inevitable blending of team meetings into other use-case cat-

egories. Since some companies are smaller and some are larger, a meeting with 20 attendants can either be a team meeting or a whole company meeting depending on the size of the company. We also note the inherent limits of this labeling scheme, as an event where a team is participating in a workshop is both a team meeting and a learning event.

Train-test split. We split the dataset into train and test parts using a stratified split with the 80 : 20 ratio.

4. Method

We experiment with multiple aspects of NLP pipelines. In order to measure the importance of an aspect, we conduct an ablation study by keeping the model fixed and only altering the studied aspect. Model performance is measured by calculating the accuracy and the micro-averaged F_1 score of the model on the test set.
Two types of models were used in our experiments:

- **FastText model** (Joulin et al., 2017). The inputs to the model are concatenated and preprocessed texts of all questions and all polls. The model does not consider the aggregated features.
- **Two-stage pipeline model.** The first step produces a document embedding for both questions and polls and in the second step we take the embeddings along with the aggregated features and classify them using a single-layer softmax neural network. The document embedding layer is trained on a concatenation of preprocessed questions and polls. The training of this layer takes place separately from the rest of the network.

4.1. Preprocessing

To assess the effect of various preprocessing approaches, we study their effect on the final performance, using the **FastText model** for the classification part. We focus on the following areas of preprocessing:

Tokenization. Three types of tokenizers are compared: simple whitespace with special characters striping, spaCy (Honnibal and Montani, 2017) and Bling Fire[2].

Part-of-speech (POS) tags. We use the spaCy POS tagger to assign POS tags to each token. The tokens are then concatenated with their POS tag, e.g. `work|NOUN` or `work|VERB`, and used for training similarly to the `sent2vec` method (Trask et al., 2015).

Named entities (NE). The spaCy NLP pipeline can also find NEs in a text. We leverage this information to join the tokens associated with a NE into a single token and concatenate the joined token with the NE type, e.g. `01-04-1997|DATE`. It is also possible to use both POS and NE tags simultaneously. If this is the case, NE tags take precedence over POS tags.

4.2. Embedding types

We also experiment with different types of embeddings used in the **two-stage pipeline** model:

- **Fast sentence embeddings (FSE)** (Borchers, 2019). We base the FSE embeddings on FastText word embeddings (Bojanowski et al., 2017) that are then composed into higher-level embeddings. The FSE library allows for three composition methods: Deep Averaging Network (DAN) (Iyyer et al., 2015), Smooth Inverse Frequency (SIF) (Arora et al., 2017) and Unsupervised Smooth Inverse Frequency (USIF) (Ethayarajh, 2018).
  The main advantage of the FSE embeddings is their fast computation speed.
- **Universal sentence encoder (USE) embeddings** (Cer et al., 2018). We experiment with multiple flavors of pre-trained USE models available on TensorFlow Hub[3]. The flavors of the USE model differ mainly in size and performance, but also in the maximum number of tokens considered by the model: the *large* flavor based on the Transformer architecture (Vaswani et al., 2017) supports a maximum of 128 tokens whereas the *default* flavor based on DAN is unbounded.
  The advantage of the USE model is that is was trained on a large corpus resulting in a fair coverage of different domains. However, unless the model is fine-tuned on the target domain, its performance can be lower than that of a dataset-specific model.
- **FastText embeddings.** We use the average of the FastText tokens in the texts to create the sentence embedding as specified in (Joulin et al., 2017).
  A possible advantage of the FastText embeddings is that they can be trained in a supervised manner.

5. Results

In this section we present and discuss the results of different experiments described in Section 4. We group these results according to the aspects described above.

Method	Size	Time	Acc	F_1
Whitespace	-	4.5s	0.656	0.642
Bling Fire	1.4MB	12.6s	0.650	0.628
spaCy	80KB	6m 1s	0.657	0.642
spaCy+POS	3.7MB	42m 34s	0.654	0.636
spaCy+NE	4.0MB	1h 1m	0.656	0.644
spaCy+POS,NE	7.7MB	1h 26m	0.653	0.635

Table 2: Performance of the different preprocessing methods in combination with the FastText model on the test set. We also report the size of the preprocessing model and the time needed for preprocessing of the training set due to their implications for practical usage.

5.1. Preprocessing

Table 2 shows the effect of testing various preprocessing approaches on the **FastText model**. The simple method incorporating a whitespace tokenizer and punctuation striping is competitive with more complex methods like Bling

[2] `https://github.com/microsoft/BlingFire`

[3] `https://tfhub.dev/google/collections/universal-sentence-encoder/1`

Fire or spaCy tokenizers. Furthermore, the latter methods require increased preprocessing time. When they do POS and NE tagging on top of tokenization, they can be up three orders of magnitude slower than the baseline.
Models trained on data with tags (especially POS) also take longer to converge, which can be seen in Figure 3. We hypothesize that this can be caused by the model needing more epochs to cope with increased vocabulary size.

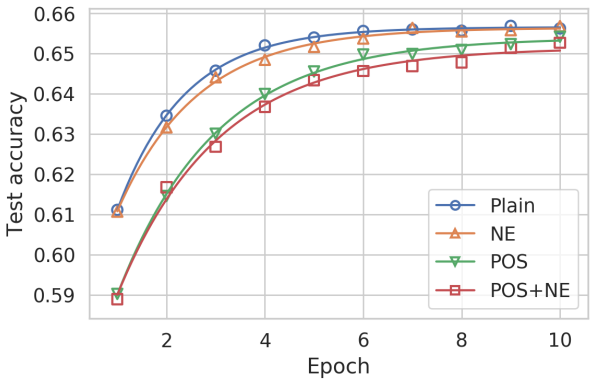


Figure 3: Test accuracies of models using various token tags at different points of the training process.

5.2. Embedding types

Table 3 shows the performance comparison of the **two-stage pipeline model** using different types of embeddings trained on texts preprocessed using whitespace tokenizer and punctuation stripping.

Embedding type	Dim	Size	Acc	F_1
FSE DAN	300	2.5GB	0.590	0.595
FSE SIF	300	2.5GB	0.585	0.593
USE default	512	1.0GB	0.575	0.584
USE large	512	850MB	0.557	0.562
FT unsupervised	150	1.3GB	0.584	0.590
FT supervised	**20**	**180MB**	**0.645**	**0.647**

Table 3: Performance of the different models on the test set along with the respective embedding dimensions. FT stands for FastText.

Despite using the largest embedding dimension and being trained on the largest corpus, the USE model achieves the lowest performance on our task. We hypothesize that this is due to its training corpora being quite different from the one available for our task.
We note that the *default* DAN-based USE flavor performs better than the *large* Transformer-based flavor. This seems to be due to the Transformer limiting the maximum number of tokens to 128. With the median of 285 words per event, the words necessary for discrimination between classes may not be considered by the Transformer.
The FSE and FastText unsupervised models reach higher performance than USE. We think that this is because they are trained on our data and manage to capture its specifics.
The model based on supervised FastText embeddings reaches the highest performance. We assume this stems from the fact that the supervised embeddings are tailored to the task. As a result, the embedding layer filters out unimportant words and makes the task of the classifier simpler.

Embedding size. We observe that unsupervised embedding models require much higher embedding size than the supervised model. It seems that only a fraction of the features captured in the unsupervised embeddings is finally useful for the classification task.

5.3. Discussion and Deployment

The results listed in Section 5 show that the F_1 score is the highest in case of the two-stage pipeline model which uses supervised FastText embeddings. On the other hand, a simple FastText model (Table 2) trained end-to-end consistently reaches higher accuracy than various two-stage pipelines (Table 3). We find this particularly notable, as it utilizes a subset of the available data: the model only considers the texts of questions and polls associated with an event. Furthermore, thanks to its simplicity and small size (300 MB in RAM), we managed to deploy this model in the AWS Lambda severless environment. As Figure 4 shows, its execution time scales linearly up to 25 000 tokens.

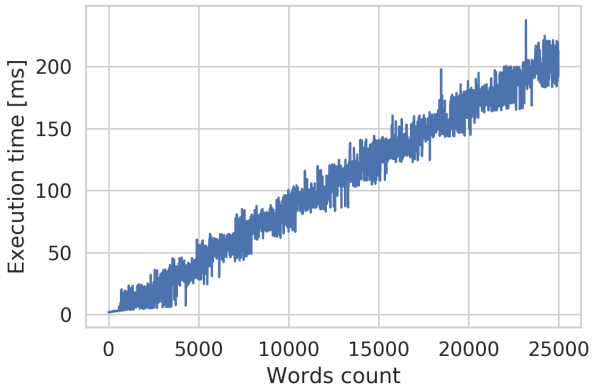


Figure 4: Execution time of the FastText model deployed in AWS Lambda as a function of the input text's length.

6. Conclusion

In this work we present a case study of a text classification task on longer texts. We evaluate the performance of various preprocessing and feature representation approaches, and show that an end-to-end trained FastText model is able to match the performance of more complex pipelines. To showcase its practicality, we deploy it to AWS Lambda.
As the resulting model is still quite large, quantization methods could be used to further reduce its size (Joulin et al., 2016). Another interesting avenue of future research would be distilling the text classification model back to regular expressions (Bui and Zeng-Treitler, 2014).

7. Bibliographical References

Aggarwal, C. C. and Zhai, C. (2012). A survey of text classification algorithms. In *Mining text data*, pages 163–222. Springer.

Arora, S., Liang, Y., and Ma, T. (2017). A simple but tough-to-beat baseline for sentence embeddings. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Batanović, V. and Bojić, D. (2015). Using part-of-speech tags as deep-syntax indicators in determining short-text semantic similarity. *Computer Science and Information Systems*, 12(1):1–31.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Borchers, O. (2019). Fast sentence embeddings. `https://github.com/oborchers/Fast_Sentence_Embeddings`.

Bui, D. D. A. and Zeng-Treitler, Q. (2014). Learning regular expressions for clinical text classification. *Journal of the American Medical Informatics Association*, 21(5):850–857, 02.

Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., John, R. S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., et al. (2018). Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*.

Desmet, B. and Hoste, V. (2018). Online suicide prevention through optimised text classification. *Information Sciences*, 439:61–78.

Ethayarajh, K. (2018). Unsupervised random walk sentence embeddings: A strong but simple baseline. In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 91–100, Melbourne, Australia, July. Association for Computational Linguistics.

Honnibal, M. and Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Ishakian, V., Muthusamy, V., and Slominski, A. (2018). Serving deep learning models in a serverless platform. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 257–262. IEEE.

Iyyer, M., Manjunatha, V., Boyd-Graber, J., and Daumé III, H. (2015). Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1681–1691, Beijing, China, July. Association for Computational Linguistics.

Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. (2016). Fasttext.zip: Compressing text classification models. *CoRR*, abs/1612.03651.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2017). Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431. Association for Computational Linguistics, April.

Kim, S.-B., Han, K.-S., Rim, H.-C., and Myaeng, S. H. (2006). Some effective techniques for naive bayes text classification. *IEEE transactions on knowledge and data engineering*, 18(11):1457–1466.

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302.

Kwok, J. T.-Y. (1998). Automated text categorization using support vector machine. In *In Proceedings of the International Conference on Neural Information Processing (ICONIP*. Citeseer.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Lee, W. S. and Liu, B. (2003). Learning with positive and unlabeled examples using weighted logistic regression. In *ICML*, volume 3, pages 448–455.

Liu, B. (2015). *Sentiment analysis: Mining opinions, sentiments, and emotions*. Cambridge University Press.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Raganato, A., Camacho-Collados, J., and Navigli, R. (2017). Word sense disambiguation: A unified evaluation framework and empirical comparison. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 99–110.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Trask, A., Michalak, P., and Liu, J. (2015). sense2vec-a fast and accurate method for word sense disambiguation in neural word embeddings. *arXiv preprint arXiv:1511.06388*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

Verma, T., Renu, R., and Gaur, D. (2014). Tokenization and filtering process in rapidminer. *International Journal of Applied Information Systems*, 7(2):16–18.

Proceedings of the LREC2020 Industry Track , pages 26–30
Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020

Hybrid Tagger – An Industry-driven Solution for Extreme Multi-label Text Classification

Kristiina Vaik, Marit Asula, Raul Sirel
TEXTA OÜ
kristiina@texta.ee, marit@texta.ee, raul@texta.ee

Abstract

This paper presents an industry-driven solution for extreme multi-label classification with a massive label collection. The proposed approach incorporates a large number of binary classification models with label pre-filtering and employs methods and technologies shown to be applicable in industrial scenarios where high-end computational hardware is limited. The system is evaluated on an Estonian newspaper article dataset which contains almost 2000 unique labels and has shown to perform over 80 times faster than applying all the binary models of the entire label set without negative impact on prediction scores.

Keywords: text classification, extreme multi-label classification, data processing workflows

1. Introduction

The interest of automating certain tasks in the news industry and other sectors dealing with large volumes of textual data has been growing rapidly. Today these tasks often rely on human's judgement and manual assignment which consumes a lot of resources. However, the market has shown a higher demand on using already existing data and NLP technologies efficiently. One of such tasks is extreme multi-label text classification, i.e automatically assigning a list of most relevant labels based on the content of the text from a large label set. In the industry multi-label text classification can be used for many different applications, such as describing the subject of a news article by assigning a general topic (e.g. *politics*, *history*, *sports*, etc.) or add specific keywords (e.g. *NATO*, *Donald Trump*, etc.) based on the relevant content. Such scenarios produce an increasingly large number of labels to predict making it a challenge for extreme multi-label classification settings to synchronously handle massive label sets.

We propose an industry-driven solution, *Hybrid Tagger* (HT), for extreme multi-label classification combining supervised and unsupervised text processing methods which have been proven to be applicable in the industrial settings. HT is part of TEXTA Toolkit[1] – an open-source framework for building and executing machine learning pipelines and analysing textual content. HT categorizes each given text in real-time with the most relevant labels from a massive label collection. The development of the HT was motivated by the need to perform classification with a high volume of labels and the lack of existing out-of-box solutions. Another issue in industrial scenarios is the limited availability of high-end computational hardware because clients often require running production-grade applications using their own infrastructure. Hence HT was designed to work with fairly limited computational power regarding today's standards. HT is currently used by two Estonian media corporations to label newspaper articles with topics & keywords and the National Library of Estonia to label books, dissertations, periodicals, etc.

The paper is further structured as follows: Section 2. gives a brief overview of the existing research on this topic, Section 3. provides an overview of the workflow used in Hybrid Tagger, Section 4. describes the results of applying the Hybrid Tagger on a dataset of newspaper articles, and Section 5. concludes the paper.

[1] https://docs.texta.ee/

2. Related Research

Multi-class classification is a type of problem where an input is assigned with a single label from a finite set of labels. However, in real-life scenarios, the content of the text is semantically far more variable, which means it most probably contains many different topics as opposed to a binary topic distribution. This limitation leads to the *multi-labelled classification* problem in which a subset of labels is assigned to the input object from a finite set of labels (Tsoumakas and Katakis, 2009; Godbole and Sarawagi, 2004).

A common approach to solve the multi-label classification problem is the *problem transformation*, specifically the *binary relevance method* (Tsoumakas and Katakis, 2009; Godbole and Sarawagi, 2004; Zhang et al., 2017a) in which the multi-label problem will be split into binary classification subtasks. After splitting the task into binary subtasks, these binary classifiers will be converted into the multi-label representation meaning that n binary classifiers are trained whereas each classifier gives a prediction of 0 or 1, i.e giving a corresponding label from a finite set of labels. Binary relevance method has often been overlooked because of the assumption that it ignores the correlations between labels, meaning that in real life it will likely give too many or too few labels. However, as Read et al. (2009) point out that if the multi-labelled datasets grow in size, methods taking label correlations into account struggle with the exponential growth of the possible correlations.

On the industry level scalability in text processing is a very important factor, although the number of labels to predict is often disregarded. Existing extreme multi-label classification papers use tree-based methods (Agrawal et al., 2013; Weston et al., 2013; Prabhu and Varma, 2014) or reduce the dimensions of the original label matrix, e.g. Bi and Kwok (2013) reduced the number of labels by doing random sampling; Zhou et al. (2012) have proposed a method called

compressed labelling which compresses the original label matrix in order to reduce the number of binary classifiers. Both of these methods can be generalized as two-stage approaches depending on complex matrix computations to find a subset of most probable labels (stage one) and to apply the corresponding binary classifiers (stage two). Zhang et al. (2017b) have applied deep learning by establishing a non-linear embedding for both feature and label spaces and combine it with a label graph, which is built from label space (two nodes share an edge if corresponding labels co-occur frequently enough). However, none of the referenced works is applicable in the industry because of the computational, infrastructural, and (labelled) resource limitations.

3. Workflow of Hybrid Tagger

Hybrid Tagger incorporates a high number of binary classification models combined with unsupervised label pre-filtering in order to achieve real-time predictions for thousands of labels. HT uses traditional classification algorithms because neural network classifiers require more training data per label to provide adequate results.

The supervised part of HT has been developed by using scikit-learn (Pedregosa et al., 2011), primarily logistic regression or SVM algorithms. The unsupervised pre-filtering is achieved by using Elasticsearch[2] engine's document retrieval features where all texts used for training and validating the classification models are indexed. Elasticsearch is used because of its stable and scalable platform for retrieving documents (for training) and performing document similarity queries for the label pre-filtering. In contrast to existing extreme multi-label classification solutions, Elasticsearch allows to disregard complex matrix computations and instead offers a fairly transparent way to filter labels based on the document similarity they have been assigned to. In this section, the workflow pipeline of HT will be described in detail.

3.1. Preprocessing

The preprocessing pipeline consists of tokenization, lemmatization, part of speech (POS) tagging, and named entity recognition (NER). This is done by using TEXTA Multilingual Processor (MLP) which uses NLTK (Bird et al., 2009) with Stanford models and EstNLTK (Orasmaa et al., 2016). Preprocessing pipeline can be omitted, but it depends on the language and domain HT is applied to. For example, morphologically less inflective languages do not necessarily require to be lemmatized to provide adequate features, whilst for other languages (e.g. Estonian) it is crucial. It decreases the size of the vocabulary, thus model size and computation time. NER can be used for speeding up the prediction process.

3.2. Training

Binary classifiers used in HT can be trained on any text segment, e.g. title, content, author, etc. In our experimental setup models are usually trained using lemmatized content and also optional POS tags. For vocabulary reduction stop words are removed from all texts prior to training.

[2] `https://www.elastic.co/elasticsearch`

Training data is selected according to the pre-existing labelling. For each label, all existing *positive* examples (texts annotated with the specific label) and the same amount of randomly selected *negative* examples (texts not annotated with the specific label) are retrieved. Examples for all labels are then randomly split into training and validation sets (default 80-20).

In real-life scenarios, the training process may result in thousands of classification models which have a significant memory imprint when combined. To combat this problem, HashingVectorizer (Pedregosa et al., 2011) is used to vectorize the training data. It has significantly smaller memory imprint than other vectorizers supported by scikit-learn (e.g. commonly used TfIdfVectorizer) as it uses the hashing trick to find the token string name to feature integer index mapping.

For training models 5-fold cross-validation and grid search is used to find the most optimal parameters for the C-value; minimum and maximum length of *n*-grams used in the model; and whether to use words (and word-based *n*-grams) or word-bound character-based *n*-grams as features. The best model from grid search is then validated using the validation set. For each model, a confusion matrix with precision, recall, and F1 score are computed.

3.3. Prediction

The training phase's aftermath is a large volume of binary models making it infeasible to apply them for label prediction in a sequential manner. This limitation is handled by devising an approach in which the number of executable binary models are limited by finding a subset of models which most likely will provide an accurate label prediction. This is done by finding n (default n=10) similar texts from the training data indexed in Elasticsearch and finding m (default m=10) most frequent labels assigned to the texts. Retrieving similar texts is done by using an Elasticsearch *more like this* query[3] which calculates top k words with the highest TF-IDF score per text and afterwards performs a disjunctive query using the pre-existing labels to match similar texts.

In general, the prediction pipeline is as follows:

- preprocess the input text (optional);
- find n similar texts indexed in Elasticsearch;
- find top m labels attached to the texts found in the previous step;
- apply named entity recognition on the input to identify l entity-related labels and remove these binary models from the list of m models which will be used later for label prediction;
- retrieve all models for each of m-l top labels;
- apply models, retrieve and combine the list of predicted labels classified by the binary models and the entity-related labels, and output the results.

[3] `https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-mlt-query.html`

In the presented pipeline both Elasticsearch-based label pre-filtering and NER-based labelling are used to reduce the number of binary classification models applied to each text. We acknowledge that the effectiveness of NER greatly depends on the specific domain and label set, nevertheless, in our use cases, it has proven to be an effective method to reduce the number binary models required.

4. Evaluation

The purpose of the evaluation is to show that Hybrid Tagger performs significantly faster than the baseline model without negative impact on prediction scores.
First, we provide a brief theoretical overview of how the applicability of Hybrid Tagger depends on the available computation power and number of labels present in the dataset. Then we evaluate Hybrid Tagger's performance on a real multi-label dataset.

4.1. Hybrid Tagger's Applicability

Let n_t be the number of classifiers used for prediction, n_c be the number of cores used for computation, μ_t the average prediction time of one binary classifier and t_{mlt_s} the time of Elasticsearch's *more like this* query with number of similar documents set to s. The approximate simplified formula for time consumption of tagging one document is thus:

$$t_{tag} = \left\lceil \frac{n_t}{n_c} \right\rceil \cdot \mu_t + t_{mlt_s} \quad (1)$$

However, it is important to note that *more like this* query does not have a significant impact on time consumption in most practical scenarios as s is usually set in the range between 10 and 100 resulting in query time under 1 second (except for datasets containing very long texts, e.g. books). Let n be the total number of labels in the dataset and m be the number of top labels used for prediction by Hybrid Tagger. The gain in time efficiency (i.e. how many times faster are HT predictions compared to applying all binary classifiers) depending on number of available cores and size of the label set, can be calculated by the following formula:

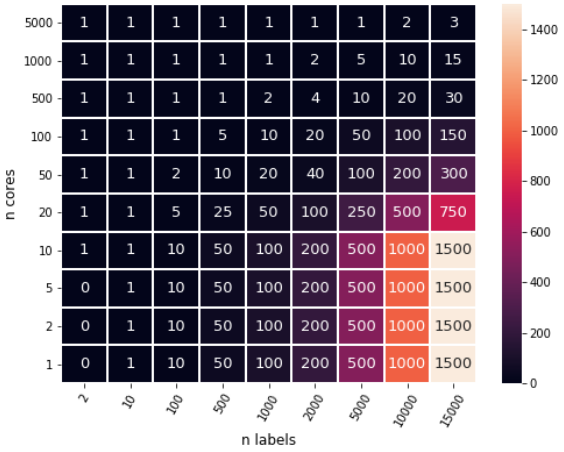
$$HT_{gain} = \left(\left\lceil \frac{n}{n_c} \right\rceil \div \left\lceil \frac{m}{n_c} \right\rceil \right) - t_{mlt_s} \quad (2)$$

Figure 1 illustrates how the gain in time efficiency with HT changes based on the number of available cores and the number of labels with m set to 10 and t_{mlt_s} set to 0 for simplification. We can see that the gain in time with HT grows as the label set increases while the number of available cores stays relatively low. For example, applying HT with original label set size of 10 000 on a machine with 5 available cores is 1000× faster than applying all binary classifiers.

4.2. Case Study: Õhtuleht Newspaper Dataset

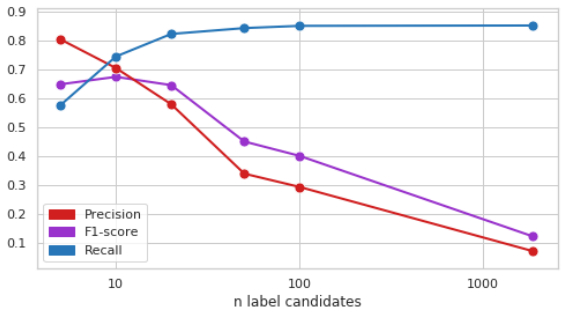
Õhtuleht dataset contains newspaper articles spanning from years 2013 to 2019, covering a wide range of topics (news, sports, entertainment, crime, etc). The dataset is not publicly available because of legal limitations. It contains 102

Figure 1: HT gain in time efficiency, if $m = 10$



450 documents with 1978 unique labels. The average number of labels per document is 4.4.
For evaluating HT, Õhtuleht dataset is split into train and test set (100 000 and 2450, respectively), and trained 1870 binary classifiers on lemmatized articles' content using logistic regression as the predictor function. The minimum number of examples per each label was set to 50, resulting in disqualifying 108 labels with a smaller number of examples. For choosing the best parameter configuration for HT, we measured average precision, recall and f1-score on the test data with (*n label candidates*, *n similar texts*) set to (5,10), (10,10), (20,10), (50,30) and (100,30). Figure 2 illustrates these results by showing how the prediction scores change when the size of the label candidate set increases. We see that precision starts decreasing after 10 candidate labels while recall stabilizes after 100. The best trade-off between precision and recall is obtained at 10 resulting in the highest f1-score.

Figure 2: Label candidates set size effect on prediction scores



To further evaluate the results, we measured average prediction time, precision, recall, f1-score and the number of predicted labels on the test data by applying:

1. Baseline model (BL) consisting of all binary classifiers;
2. Hybrid Tagger with the best detected parameter configuration (*n label candidates* = *10*, *n similar texts* =

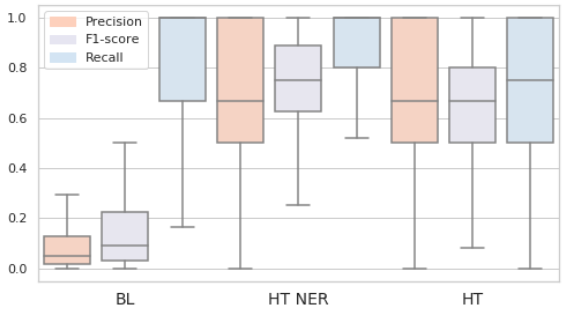
10) with NER enabled (HT NER) and NER disabled (HT).

Table 1: Comparing Baseline with Hybrid Tagger

	BL	HT NER	HT
n taggers	1870	10	10
n similar texts	nan	10	10
n cores	24	24	24
NER enabled	no	yes	no
Time (s)	82.34	1.01	1.01
Precision	0.07	0.70	0.71
Recall	0.85	0.92	0.75
F1-score	0.12	0.76	0.67
n predicted labels	128.87	6.21	4.89

Table 1 gives an overview of Hybrid Tagger's performance compared with the baseline model (BL). Figure 3 visualizes the prediction scores' distribution of the same models, while the times' distribution can be seen on Figure 4 and the distribution of predicted labels per one document on Figure 5.

Figure 3: Prediction precision, f1 and recall scores



To verify that our experimental results are in accordance with theory, we can calculate the theoretical gain in time with our HT configuration by inserting the chosen parameter combination into formula (2) and compare it with the measured result. Thus, for theoretical gain in time we get: $HT_{gain} = \left(\left\lceil \frac{1870}{24} \right\rceil \div \left\lceil \frac{10}{24} \right\rceil \right) = 78\times$.

BL model's actual average prediction time is 82.34 seconds while HT labels one document on average with 1.01 seconds resulting in an actual gain of $81.5\times$. The minor difference in magnitude can be caused by practical reasons, e.g. small variations of prediction times of individual taggers (the slower models are always included in the BL label set, while might not be in HT label candidates). Nonetheless, our experimental results prove that HT performs significantly faster than the BL model.

Furthermore, BL's f1-score is only 0.12 as a result of extreme over labelling causing a very low precision of 0.07. We can see from Figure 5 that the number of labels predicted with BL is close to 100, while the actual number of labels seldom passes 10. As HT limits the number of label candidates, it does not suffer from the same problem and precision remains 0.7 both with and without NER (see Figure 3). HT's recall without NER is 0.75 being slightly lower than BL's average of 0.85. However, HT with NER obtains even better recall than the BL model with an average score of 0.92. It is still important to keep in mind that the applicability of NER is dataset-specific as using entity-related labels is fairly common with newspaper articles and enabling NER helps to combat the problem of low number of training examples of such labels, but might make the model prone to false positives in some other domains.

Figure 4: Prediction times in seconds

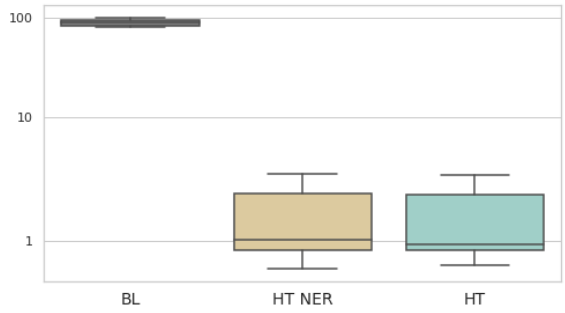
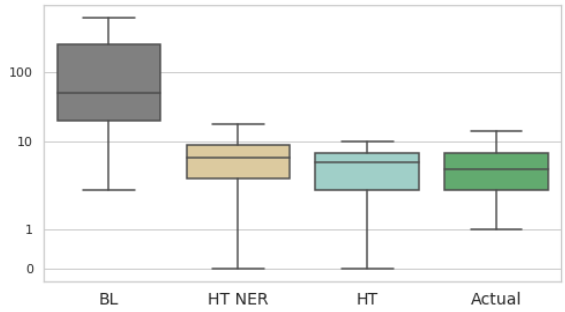


Figure 5: Number of predicted labels per document



5. Conclusion

This paper presented an industry-driven solution, Hybrid Tagger, for extreme multi-label classification with a large volume of unique labels. The proposed system incorporates a high number of binary classification models coupled with unsupervised label pre-filtering and named entity recognition to achieve real-time predictions with thousands of labels. As the development of Hybrid Tagger was industry-driven, it does not employ state-of-art methods but rather relies on methods and technologies proven to work on an industrial scale.

The evaluation of Hybrid Tagger on Õhtuleht newspaper dataset shows that Hybrid Tagger helps to significantly improve both the prediction times and precision scores in comparison to executing all binary classification models of the label set.

Hybrid Tagger is currently used by two Estonian newspaper corporations and Estonian National Library to label their content. Since the workflow of HT is language independent, the next step is to apply the solution for industrial projects in other languages.

6. Acknowledgements

The work described in this paper has been supported by the language technology research and development program "Estonian Language Technology 2018–2027" of the Ministry of Education and Research under grant EKTR3, by European Union's Horizon 2020 research and innovation programme under grant agreement No 825153, project EMBEDDIA (Cross-Lingual Embeddings for Less-Represented Languages in European News Media) and Enterprise Estonia project No. EU48684, Research Project No. 1.11 (Deep neural models and cross-lingual embeddings in TEXTA Toolkit).

7. Bibliographical References

Agrawal, R., Gupta, A., Prabhu, Y., and Varma, M. (2013). Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. pages 13–24, 05.

Bi, W. and Kwok, J. (2013). Efficient multi-label classification with many labels. *ICML*, pages 405–413, 01.

Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition.

Godbole, S. and Sarawagi, S. (2004). Discriminative methods for multi-labeled classification. In Honghua Dai, et al., editors, *Advances in Knowledge Discovery and Data Mining*, pages 22–30, Berlin, Heidelberg. Springer Berlin Heidelberg.

Orasmaa, S., Petmanson, T., Tkachenko, A., Laur, S., and Kaalep, H.-J. (2016). Estnltk - nlp toolkit for estonian. In Nicoletta Calzolari (Conference Chair), et al., editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France, may. European Language Resources Association (ELRA).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Prabhu, Y. and Varma, M. (2014). Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 08.

Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2009). Classifier chains for multi-label classification. In Wray Buntine, et al., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 254–269, Berlin, Heidelberg. Springer Berlin Heidelberg.

Tsoumakas, G. and Katakis, I. (2009). Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 3:1–13, 09.

Weston, J., Makadia, A., and Yee, H. (2013). Label partitioning for sublinear ranking. *30th International Conference on Machine Learning, ICML 2013*, pages 840–848, 01.

Zhang, M.-L., Li, Y.-K., Liu, X.-Y., and Geng, X. (2017a). Binary relevance for multi-label learning: an overview. *Frontiers of Computer Science*, 12, 11.

Zhang, W., Wang, L., Yan, J., Wang, X., and Zha, H. (2017b). Deep extreme multi-label learning. 04.

Zhou, T., Tao, D., and Wu, X. (2012). Compressed labeling on distilled labelsets for multi-label learning. *Machine Learning*, 88, 07.

Proceedings of the LREC2020 Industry Track , pages 31–35
Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020

Industrial Machine Translation System for Automotive Domain

Maria Sukhareva*, Olgierd Grodzki†, Bernhard Pflugfelder*
*BMW Group
Bremer Straße 6, 80807 Munich
{maria.sukhareva, bernhard.pflugfelder}@bmwgroup.com

†Data Reply
Luise-Ullrich-Straße 14, 80636 Munich
o.grodzki@reply.de

Abstract

BMW Group, a large multinational company, inevitably faced the challenge of processing and creating large amounts of multilingual data. As manual translation fails to provide sufficient coverage and cost efficiency on a large scale, an acute need for a reliable machine translation service naturally arose. Translating highly technical automotive texts has proven to not be a trivial task and off-the-shelf commercial cloud solutions fail to deliver satisfactory translation quality. In this paper, we present a customized machine translation system tailored for automotive needs. Our system is well-suited for translating automotive texts and satisfies all data protection requirements. The use cases that we discuss in this paper have a projected business value between one and five million euros.

Keywords: machine translation, industrial system, domain adaptation

1. Introduction

BMW Group is a multinational company: it currently operates 30 production and assembly facilities in 14 countries and has a global sales network in more than 140 countries. As of December 2018, the BMW Group had a workforce of 134,682 employees from 124 nationalities. Having a fast and reliable machine translation infrastructure is vital for the company's value chain.

Assimilation of multilingual information in BMW Group The BMW Group receives daily thousands of customer comments in over 100 languages. These comments further undergo qualitative and quantitative analysis. BMW data scientists use machine learning methods to cluster, classify and extract information from the comments while customer relations specialists process the feedback individually addressing specific customer needs. Another source of multilingual data is dealer feedback. Currently car dealers operate in over 140 countries and BMW support service receives tickets in dozens of languages. Manual translation of such tickets is costly and causes unnecessary delay. Applying machine translation can significantly diminish the processing time.

Dissemination of multilingual information in BMW Group BMW Group creates a multitude of multilingual texts on a daily basis such as car manuals and training materials for dealers, marketers and customers. These kind of texts cannot tolerate any errors. Thus, the current approach is to automatically pre-translate the texts and involve human technical translators as post-editors.

Multilingual communication The company operates several support hotlines e.g. IT support, financial services and dealer support. Support agents are reachable by telephone as well as through an online chat. Machine translation here can be used for real time translation of low-resource languages for which it is not plausible to create a manned support service.

2. Automotive domain

The automotive domain includes a variety of texts, e.g. car manuals, promotional texts, error reports, protocols of production changes. All those texts pose a challenges of specialized terminology. While our machine translation encompasses a multitude of BMW domains and applications, this paper will focus on two use cases: (i) translation of car manuals and training materials and (ii) translation of production protocols. The savings from the usage of machine translation on these use cases are projected to be between one and five million euros annually.

Car manuals are instructions for car mechanics, dealers and customers on how to repair and maintain the vehicles. The original texts are written in German and are to be translated into multiple languages.

BMW Group has *prescriptive* terminological dictionaries for human translators and technical writers. The dictionaries contain lists of concepts and their lemmata in over 30 languages. The concepts are defined in German and, thus, all the dictionary entries have a German lemmata. An entry can have several non-German entries in other languages that should be used for translation. There is a list of synonyms for a given concept which shall not be used in technical documentation and translation (*negative* terms). Languages are not equally represented in the dictionary: English has most of the entries while Ukrainian, Bulgarian etc. have just a few thousands terms. Table 1 shows the statistics over the corporate lexicon. The lexicon has a total of 1241087 entries with 227590 being negative terms.

The dictionary is prescriptive i.e. the resulting translations have zero tolerance for synonyms i.e. using a term that is marked as negative invalidates the whole translation even if the meaning is preserved. This kind of restrictions pose several challenges, first of all, the challenge of integrating the lexica into the translation and the challenge of the automatic evaluation as commonly used measures such as TER or BLEU treat all the n-grams equally.

lang	entries	lang	entries	lang	entries	lang	entries	lang	entries
German	96032	Portuguese	54932	Turkish	48020	Swedish	13526	Bulgarian	6974
English (UK)	74404	Russian	53210	English (US)	47150	Slovene	12550	Portuguese (BR)	6632
French	73626	Greek	52832	Thai	46744	Hungarian	12174	Dutch (BE)	2284
Spanish	69050	Finnish	50932	Chinese	45946	Romanian	12096	French (BE)	2272
Italian	63466	Korean	50172	Czech	34888	Norwegian	9016	Ukrainian	2150
Dutch	60434	Danish	48332	Indonesian	34692	Chinese (TW)	8178	English (AU)	2104
Swedish	60358	Japanese	48090	Polish	28880	Arabic (SA)	7496	English (ZA)	1444

Table 1: The amount of entries per language in the prescriptive corporate lexicon

```
G34 ag upr/lwr seal/grommet opt. E34.234.2 assy line protection LU with ERWU.
If the electronic immobilizer (EWS) functionality is disabled,  the TEE shall
prevent disengagement of the parking lock. G83 = Non-return valve, cylinder head.
Close the trim grille CS/JCW/Cooper/CooperD/One/OneD.
```

Figure 1: An example of a production protocol

Translation of production protocols is another challenging MT use case. Daily, engineers protocol their actions e.g. ordering car parts, changing design decision etc. The protocols are highly technical but also contain a lot of abbreviations, acronyms etc. for brevity. Figure 1 shows a snippet of a production protocol. The texts are not easily decipherable by a person who lacks specialized training. Similarly, machine translation systems trained on out-of-domain data such as news corpora have a sub-par performance on this data. The original protocols are written either in English or in German. The German protocols are translated into English and communicated to the plants in non-German speaking countries. The English protocol translations are sent to the plants in Germany. Apart from the obvious challenges of non-standardized punctuation and domain specific lexica, additional challenges of copying numbers and translating abbreviation is added. Having erroneous translation of numbers has in fact proven to be worse than not having a translation at all as post-editors may fail to notice the error if the number is present in the text.

To sum up, the automotive domain is comprised of multiple genres of varying complexity: from easily readable customer reviews to terminology-rich production protocols. Implementing machine translation of automotive texts is not trivial and involves various degrees of domain adaptation depending on the use case.

3. Customized machine translation

Industrial MT also have to comply with restrictions imposed by the EU's General Data Protection Regulations[1] that states that no personal data can be disclosed to a third party without customer consent. This limits the options for cloud-based machine translation systems as data coming from customers and dealers may contain personal information. Another restriction is confidentiality: Production protocols are confidential and cannot be passed to a third party. Over the years, BMW Group has accumulated a large amount of translation memories for the languages listed in Table 1. The translation memories are parallel text fragments that have been translated to or from German. The fragments can be sentences and clauses, but are on average much shorter phrases.

We have implemented customized machine translation solutions for German and three high-resource languages: English, Italian and Spanish. English, unsurprisingly, has the largest amount of parallel data with over 5.5 million fragments. Spanish has 2.6 million parallel fragments and the Italian data has 2 million fragments. Training a machine translation system solely on this data is not possible as, for example, the average sentence length of a fragment is 9 words in German and 11 words in English for the production protocols and 7 words in German and 9 words in English for the car manuals. Thus, to ensure that the models learn to handle longer sequences, we used out-of-domain open-source data, which eventually doubled the training sets for all the language pairs: over 11 million parallel fragments for English and 5 million and 4 million for Spanish and Italian correspondingly.

3.0.1. Data filtering and preprocessing

We filtered the data with a language detector and eliminated all of the sentence pairs with a length discrepancy (the token ratio 0.6). We also deleted all the tags and non-ASCII characters. To prevent mistranslation of numeric data, we substituted all the tokens that contain digits (e.g. GB18, 10.02.2020, 24-241-123-123-432) with a placeholder and used lexical constraints (Post and Vilar, 2018) to make sure that all the digits are present in the translation. We also use placeholders to integrate a list of BMW abbreviations and untranslatables. Finally, we used byte-pair encoding to tokenize the data.

3.1. Model training

The models are on 8 GPUs with a toolkit for neural machine translation Sockeye (Hieber et al., 2017a). To avoid later bias towards shorter translations, we also learn the brevity penalty parameter during the training (Hieber et al., 2017b). The Spanish and Italian models are only used for car manuals while the English model is applied to production protocol as well. Despite terminological similarity, production

[1] https://gdpr-info.eu/

protocols are hastily written texts with multiple acronyms, abbreviations, digits and orthographic errors. Car manuals are, on the contrary, carefully crafted texts and void of errors. Thus, we have experimented with various set-ups: (i) training separate models for car manuals and protocols, (ii) training a multitask model by adding a tag to differentiate between use cases (Johnson et al., 2017) (iii) and training a joint model. The joint model has reached the highest BLEU score (Papineni et al., 2002), followed by the disjoint training for which the BLEU score dropped on average by 0.06. The multitask approach performed the worst mostly because the model would opt for short translations biased towards the length of the in-domain data.

3.2. Translation

We use the same preprocessing steps for translation as for training. The models are deployed on the BMW AWS cloud described in section 4. To accelerate inference, we also follow Post and Vilar (2018) and integrate the lexical translation probabilities learned from the training data with fast align (Dyer et al., 2013). The lexicon is learnt on the training data for each language pair and we set the top k candidates to 200.

3.3. Evaluation

We have compared our machine translation system to cloud-based commercial systems available on the market. The goal of the evaluation was to show that our system can achieve state-of-the-art performance and is more suitable for translating automotive domain texts than the off-the-shelf tools. While we are aware that some commercial systems allow in-domain data integration, our training data are confidential and cannot be shared with third parties. Therefore, our system has a clear advantage of having seen in-domain data but as the goal of the evaluation is not to compare algorithmic approaches but rather to show that an in-house machine translation system can deliver high quality translation while complying with data protection standards. 1000 reference sentences were translated by our CMT and by two cloud-based commercial translators (CS1 and CS2) Unsurprisingly, the cloud-based systems performed poorly on both datasets (see Table 2) with CS1 reaching the BLEU score of 0.55 and CS2 reaching the BLEU score of only 0.4. The CMT reached the BLEU score of 0.78 on the same test set. We have conducted qualitative evaluation of the results and observed that even for the sentences that have lower n-gram overlap the CMT produces better translations than commercial systems. Table 3 shows examples of sentences that were not translated perfectly by the CMT. The subsequent qualitative analysis by human translators concluded that CMT produces acceptable translations unlike off-the-shelf commercial systems that frequently fail to convey even the general idea of the sentence. As the main practical objective of translating production protocols was to facilitate information exchange between BMW engineers, we have concluded that CMT satisfies the quality requirements and the system has been launched into production.

Translation of car manuals posed an additional challenge as the end users are dealers and customers and, thus, the translation should not only adequately transfer the meaning but also be completely error-free as well as grammatically, orthographically and punctuation-wise correct. This could only be achieved by adding post-editors to the workflow. We have conducted three post-editing experiments for three customized models: German to English, German to Italian and German to Spanish. Human translators were asked to post-edit five documents (10259 sentences) translated from German into a corresponding language. The post-editors were instructed to only apply minimal edits in order to reach publishable quality. As the main pragmatic objective of integrating machine translation into the workflow was to accelerate the translation process, the usefulness of the system corresponds to the speed gains by post-editors as compared to translators. The best results were achieved for English with post-editing speed of 909 words per hour. Italian post-editor gained the speed of 650 words per hour and the Spanish post-editor was processing 641 words per hour. The better results for English can be easily explained by the fact that the English customized model was trained on three times as much data as compared to the other two languages. As the average translation speed of a BMW technical translator is 330 words per hour, we have shown that using our customized machine translation, BMW translators can process double or triple text volumes as compared to translation from scratch.

Domain	CS1	CS2	CMT
Protocols	0.56	0.4	0.73
Manuals	0.63	0.56	0.76

Table 2: BLEU Score evaluation of the CMT as compared to industrial systems

3.4. System updates and quality control

In order to improve the performance of the machine translation system and to keep the system up-to-date, the system is retrained each time when 100,000 new sentence pairs are available for a language pair. These data are then split into training, development and test sets with both test and development sets having 4,000 sentences. The evaluation is done on the newly acquired test set as well as on the test sets kept from previous (re-)trainings. In this way, we make sure that the new system is not overfitting the new data. If after retraining the model delivers satisfying performance on all the test data, the model is moved into production.

We also monitor the quality of the productive models. The post-editing results for each sentence are directly communicated back to the system and two measures are computed: the post-editing speed and the human-targeted translation edit rate (HTER) (Snover et al., 2006).

4. Scaling CMT to production

The solution is deployed in Amazon Web Services (AWS, 2020b), the world's most widely adopted cloud platform. It offers reliable, scalable, and inexpensive cloud computing services. The translation system was designed using a selection of the services offered by AWS. The architecture diagram is presented in Figure 2. All components were de-

Original	Human	CS1	CS2	BMW CMT
Neusausleitung Sensor Fussgaengerschutz G0x aufgrund der Max Boole	New export of sensor pedestrian protection G0x due to the Max Boole	Reuse sensor foot protector G0x due to the Max Boole	New version sensor football instrument protection G0x due to the Max Boole	New export sensor pedestrian protection G0x due to Max Boole
Rippe an Auflage LT entfernen, um Zugaenglichkeit an Sitzaussenlager zu gewaehrleisten	Remove rib at the support of the side member to ensure accessibility at the outer seat bearing.	Remove the rib on support LT to ensure traction on the outside bearing	Remove rib to edition LT to ensure access to seat outside warehouses	Remove rib on support LT to ensure accessibility to seat outer bearing
Fruehester moeglicher Wareneingangstermin BMW:	earliest possible incoming goods date BMW:	Early arrival date BMW:	Fruitful possible goods receipt date BMW:	Earliest possible receipts of goods BMW:
Freigabe der Bauteile die nicht Inhalt Huelle sind.	Release of components that are not part of the cover.	Release of components that are not content hulle.	Release of the components that are not content Huelle.	Release of the components that are not contained in the case.
Abloesung der BAW BET698	Replacement of deviation permit BET698	The removal of the BAW BET698	Abloation of BAW BET698	Release of the Closure of dev.perm BET698

Table 3: Comparison of selected translations from two commercial systems (CS1 and CS2) and BMW CMT

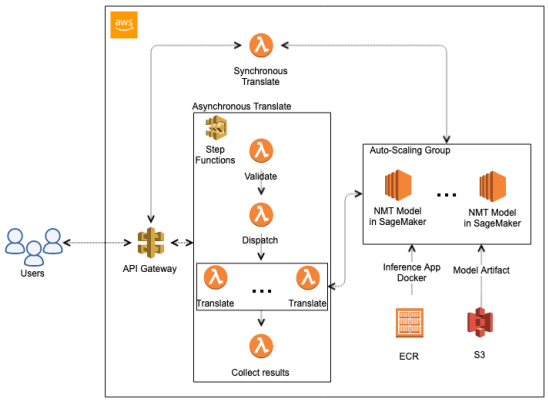


Figure 2: Implementation of the customized machine translation on AWS

ployed within an BMW AWS VPC with only private subnets and without any external connectivity.

The inference code was wrapped in a Flask application in order to expose a REST API to perform translations. It was then packaged in a Docker image and pushed to AWS Elastic Container Registry (AWS, 2020a). The model artifact is not a part of the image, instead it is packaged in an archive and uploaded to AWS Simple Storage Service (S3). This object storage service offers industry-leading scalability, data availability, security, and performance (AWS, 2016). This separation was done following engineering best practices - the model lifecycle should be managed separately from the inference code.

The application is deployed in AWS SageMaker, which is a fully managed service that provides every developer and data scientist with the ability to build, train, and deploy machine learning models quickly (AWS, 2019a). It provides many built in features needed for production, e.g. versioning, autoscaling, zero downtime redeployments, and canary testing. The SageMaker deployment makes uses of the inference application Docker image and the model artifact.

The translation system is exposed via an API deployed using AWS API Gateway (AWS, 2015). The API can be called in one of the following request-reply patterns:

- synchronous - the client submits a translation request and blocks execution until it receives a translation response,
- asynchronous - the client submits a translation start request, immediately receives a job ID as a response and proceeds to use it to poll for status and retrieve the results once the job is finished.

The asynchronous pattern is preferred because it leverages the potential parallelism of the translation service, reduces the risk of timing out, and accelerates the translation of large text packages. The system also includes a web frontend which allows users to engage with it from their web browsers.

Robustness of the service is ensured through the auxiliary code e.g. validation of the requests, choosing and invoking the correct SageMaker endpoint, etc. This code was deployed using AWS Lambda, which is a platform that enables code execution without provisioning or managing servers. The orchestration of the Lambda functions was done by AWS Step Functions (AWS, 2019b), which allow for the coordination of multiple AWS services within serverless workflows. In doing so it abstracts away the state and transformation management in order to focus on the business logic.

The production-readiness of the system is further enhanced via the application of a number of engineering best practices. The infrastructure is managed by an infrastructure as code (IaC) solution Terraform. Building and deploying the code for both the frontend and backend is managed by CI/CD pipelines (Chapman, 2014). Every component of the system provides logs and metrics to AWS CloudWatch. Alarms, based on the CloudWatch metrics, are sent to an SNS topic which delivers emails to subscribers in both the alarm and OK states. The observability of the service, meaning the distributed tracing of end-to-end requests through all layers of the solution, is performed via the use of AWS X-Ray to generate service maps and traces.

5. Conclusion

Machine translation has a wide spectrum of applications in BMW Group. This paper has focused on two challenging use cases: the translation of production protocols and the translation of car manuals. These use cases have a projected business value of several million euros. Our CMT systems have proven to deliver high quality translations of automotive texts and to accelerate human translation. Our future work includes introducing additional languages into the CMT system, integrating lexica and extending our evaluation methodology to assess terminological consistency according to corporate standards.

6. Bibliographical References

Dyer, C., Chahuneau, V., and Smith, N. A. (2013). A simple, fast, and effective reparameterization of IBM model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, Atlanta, Georgia, June. Association for Computational Linguistics.

Hieber, F., Domhan, T., Denkowski, M., Vilar, D., Sokolov, A., Clifton, A., and Post, M. (2017a). Sockeye: A Toolkit for Neural Machine Translation. *arXiv preprint arXiv:1712.05690*, December.

Hieber, F., Domhan, T., Denkowski, M., Vilar, D., Sokolov, A., Clifton, A., and Post, M. (2017b). Sockeye: A toolkit for neural machine translation. *CoRR*, abs/1712.05690.

Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., Hughes, M., and Dean, J. (2017). Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July. Association for Computational Linguistics.

Post, M. and Vilar, D. (2018). Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1314–1324, New Orleans, Louisiana, June. Association for Computational Linguistics.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *In Proceedings of Association for Machine Translation in the Americas*, pages 223–231.

AWS, (2015). *AWS Serverless Multi-Tier Architectures*.

AWS, (2016). *AWS Storage Services Overview*.

AWS, (2019a). *Deep Learning on AWS*.

AWS, (2019b). *Implementing Microservices on AWS*.

AWS, (2020a). *Amazon ECR User Guide*.

AWS, (2020b). *Overview of Amazon Web Services*.

Chapman, D., (2014). *Introduction to DevOps on AWS*.

Proceedings of the LREC2020 Industry Track , pages 36–39
Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020

NERPy: A Framework for Named Entity Recognition Experiments

Constantine Lignos
Brandeis University
415 South St., Waltham, MA 02453, USA
lignos@brandeis.edu

Abstract

Creating a high-performing sequence named entity recognition (NER) system requires a series of interconnected design decisions, including the choice of entity encoding, and for some models, the design and selection of features. In this paper, we introduce NERPy, an MIT-licensed NER framework designed to support flexible experiments in named entity recognition. We demonstrate NERPy by performing a sample experiment using the CoNLL 2003 English NER data that explores the performance of different entity encoding schemes across a wide range of training data sizes.

Keywords: Named entity recognition, Sequence models, NLP frameworks

1. Introduction

Careful experimentation is a key part of producing high-performing named entity recognition systems. However, researchers, software developers, and students interested in performing experiments in NER system design have limited tools available to them, and constructing an experimental framework for NER is a daunting task for non-experts. We address this gap with the creation of NERPy, a Python-based framework for NER experiments.

In addition to the substantial amount of work on developing state of the art systems, previous work has addressed design questions for NER systems and released software that allows for experimentation. Ratinov and Roth (2009) provide one of the most comprehensive explorations of design considerations for NER systems, finding that BILOU entity encoding (see Section 2.1) outperforms BIO in their system and that standard Viterbi decoding approaches may be slower and worse-performing in practice compared to greedy decoding with non-local features. Dernoncourt et al. (2017) introduce NeuroNER, a toolkit for neural NER designed for non-expert use, which support tight integration with the annotation process. Yang and Zhang (2018) introduce NCRF++ for performing experiments with neural NER models, and Yang et al. (2018) carefully explore the performance characteristics of models in that framework.

Many existing toolkits are primary targeted at NER researchers looking to experiment with state of the art systems using standard datasets, and are built around specific sequence model backends which provide for training and prediction. Like NeuroNER, NERPy primarily targets non-experts in the NER domain, especially software developers in industry, and is constructed to allow them to build and experiment with NER systems using standard datasets or their own data. NERPy allows users without expertise in NER to build NER systems and optimize their design by selecting the features, entity encoding, sequence model backend, and hyperparameters and then exploring the results.

NERPy's primary focus is building *non-neural* models, primarily using CRFsuite (Okazaki, 2007) for inference. Integration with neural backends is currently ongoing; see Section 4 for further discussion. Non-neural backends were prioritized first due to the existence of NCRF++ and NeuroNER, which reduced need in this area, and because one of NERPy's strengths is the ease with which users can select and customize features, which is of much lower importance for neural systems. Many of NERPy's users may lack the expertise and/or computational resources to effectively train neural NER models for custom tasks, and thus it is important to provide a framework that meets their needs. Even if a neural model is desired for the final system, a non-neural model can provide a strong baseline and may aid in the "booststrapping" process where models are trained with small amounts of data until more is available.

While NERPy is not designed to be used in production, it can be used for prototyping, selecting the design parameters of an NER system, performing research in NER system design, and as a strong but fast baseline for other systems to beat. NERPy is designed to support comprehensive configuration of every part of the system design, including selection of hand-tuned features, use of unsupervised word representations such as word embeddings or Brown clusters, entity encoding, and the sequence model backend. This framework allows non-expert users to train NER systems from scratch using large or small data sets and compare system designs without having to implement any of the NER system themselves. NERPy enables reproducible, comprehensive experiments regarding the overall design of NER systems in any natural language for which the user can provide data.

2. Design Goals for NERPy

A complete NER system. NERPy provides a complete system for NER, including ingesting annotation, generating features (Section 2.2), encoding entities, training, predicting, scoring, and analyzing errors. It depends only on the attrs, python-crfsuite, frozendict, numpy, and regex Python packages, easing installation. NERPy uses a universal document representation that consists of sentences, tokens (which support storing properties such as part of speech and any user-defined properties), and entity mentions. Unlike spaCy (Honnibal and Montani, 2017), it supports representing nested or overlapping names, but does require that names are token-aligned, as is common for token-based sequence models and is required by the CoNLL annotation format.

Encoding	Labels					
BIO	B-MISC	B-MISC	I-MISC	O	B-PER	I-PER
IOB	I-MISC	B-MISC	I-MISC	O	I-PER	I-PER
IO	I-MISC	I-MISC	I-MISC	O	I-PER	I-PER
BILOU	U-MISC	B-MISC	L-MISC	O	B-PER	L-PER

Table 1: Entity encodings for the tokens of the string *Australian Davis Cup captain John Newcombe*.

NERPy can read CoNLL shared task[1] and OntoNotes formats and provides a CoNLL format writer. NERPy provides evaluation scripts that can score output in the NERPy or CoNLL formats and report overall and per-type entity F1 in addition to producing delimited files to support error analysis (most frequent errors, etc.). Users can call a single provided script to train and test a model, and can run experiments without any code changes simply by specifying different JSON files to configure features and select the sequence model backend and its hyperparameters.
In addition to supporting running experiments from the command line, NERPy's API can be used to perform in-memory training and prediction without needing to write any data to disk. This allows it to support rapid prototyping of applications that use named entity recognition. NERPy accepts input consisting of tokenized, sentence-segmented data organized into documents of any length, and adds named entity mentions to the document structure.

Interchangeable sequence model backends. NERPy primarily uses CRFsuite (Okazaki, 2007) to support training and decoding of sequence models. CRFsuite's supported training algorithms include L-BFGS, averaged perceptron, passive aggressive, AROW, and SGD.

Tested and easy to extend. NERPy is licensed using the MIT license. It is written in pure Python and supports Python 3.7 and up. NERPy includes a comprehensive test suite with 100% code coverage, allowing users to easily verify that any modifications do not affect correctness. It is extremely to add features or integrate a new backend other than CRFSuite, and we have tested integration with other backends during development.

2.1. Encoding Entities

To encode named entities in a sequence model, each entity must be converted into a sequence of labels. Consider this example from the CoNLL 2003 English NER annotation: [*Australian*]MISC [*Davis Cup*]MISC *captain* [*John Newcombe*]PER. NERPy supports the most popular entity encoding schemes, which would encode the five tokens of this example sentence as shown in Table 1[2], and provides robust decoding that can handle invalid label sequences produced by the sequence model. With the exception of IO, which is lossy in the case of adjacent entities, any of these encoding strategies are capable of supporting separate, adjacent entities of the same type, and there is no *a priori* reason to select one lossless encoding over another. We return to entity encodings in Section 3.1.

[1]The format varies across the four languages used in the 2002-3 CoNLL shared tasks, differing in the number of fields and the exact manner in which the *DOCSTART* document separator was used. NERPy maintains the original document boundaries and can read all fields specified in any of the four languages and adds them as attributes on each token.

[2]We can only briefly note that IOB and BIO have often been confused, and that BILOU is isomorphic to BMES/BIOES/IOBES.

```
{
  "word": {
    "window": [-2, -1, 0, 1, 2],
    "token_identity": {
      "lowercase": true
    },
    "word_shape": {},
    "is_capitalized": {},
  },
  "subword": {
    "window": [0],
    "suffix": {
      "min_length": 1,
      "max_length": 4
    }
  },
  "distributional": {
    "window": [-1, 0, 1],
    "word_vectors": {
      "scale": 2.0,
      "path": "<path to embeddings>"
    }
  }
}
```

Figure 1: Sample JSON feature configuration

2.2. Features

The currently-supported features include: the token itself (lowercased if desired), the word shape of the token (capital letters mapped to *A*, lowercase letters to *a*, digits to *0*, all other characters unchanged), whether the first character of the token is capitalized, whether all characters in the token are capitalized, whether the token is all digits, whether the token contains a digit, whether the token is all punctuation (using Unicode character categories for maximum generalization across languages), the length of the token (either as a binary feature for each length or a single continuous feature), token prefixes and suffixes, Brown cluster paths (and their prefixes), and word embeddings.
Figure 1 gives a sample feature configuration .json file that shows how features can be defined. Multiple user-named feature sets (*word*, *subword*, and *distributional* in this example) can be simultaneously used, each with differently-sized windows of application. For example, a window of `[-1, 0, 1]` will generate features for the current, previous, and next word for that feature set. Some features have mandatory or optional arguments, such as specifying whether to lowercase tokens, or the path to the file to be used for embeddings. Any arguments provided are passed as keyword arguments to the constructor of a class that generates the feature. For example, the string `token_identity` in the configuration is mapped to the `TokenIdentity` class which has a constructor with the signature `def __init__(self, *, lowercase: bool = False)`. Thus, adding a new

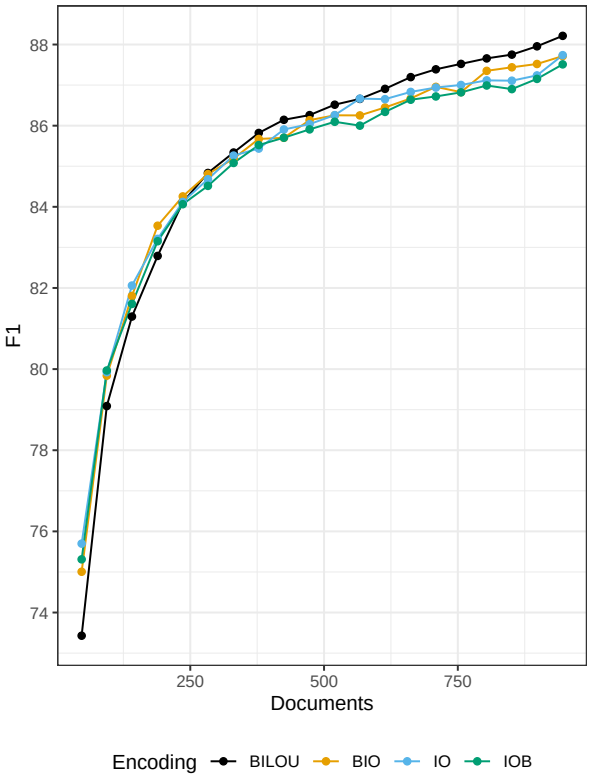


Figure 2: Entity F1 on CoNLL 2003 English test data for all encoding schemes across training data sizes.

feature is as simple as defining a new class that implements it and adding the class to the map of feature names to implementing classes. Any added feature can be configured in the same way as the built-in features by using JSON.

3. Discussion

3.1. Sample Experiment

To demonstrate NERPy's capabilities, we report results for an experiment that explores the impact of entity encoding choice at varying sizes of training data. The features used were: the token string, word shape, whether a token is capitalized, all caps, all punctuation, numeric, and/or contains a number, the length of the token (as a binary feature for each length), suffixes of length one to four, and word embeddings. All features were computed on the focus token and in a window of the two previous and two next tokens. We did not use part of speech features because we are particularly interested in performance on small amounts of training data; as part of speech taggers can produce tags that indicate proper nouns, they can effectively pass information from the part of speech training data into the NER training data, making the NER training data in effect larger than it actually is. Word embedding features were generated using fastText 300-dimensional word embeddings with subword information (Mikolov et al., 2018, wiki-news-300d-1M-subword). We selected these embeddings because they retain the casing of the original data and give some of the benefit of character-based models by using subword information.

For the purpose of easy reproducibility and comparison against other work, we use the CoNLL 2003 NER shared task English data (Tjong Kim Sang and De Meulder, 2003), reporting performance as entity F1 on the test set (the standard CoNLL NER shared task metric). To explore the effect of varying the amount of training data, we evaluated at 20 points, using 5%–100% of the training documents in 5% increments. For all training sizes except 100% (where doing so is impossible), we report the mean F1 for five random selections (without replacement) of training documents.

The training algorithm and hyperparameters were selected based on validation set performance when training on the full training set. Models were trained using L-BFGS for 100 iterations with an L1 regularization coefficient ($c1$) of 0.01, and an L2 regularization coefficient ($c2$) of 0.001. As training is implemented deterministically and initializes with zero weights, experiments from multiple random initializations are not required. We trained the model at each data size point using each label scheme. In total, this resulted in 80 experimental configurations, the cross product of four encoding schemes and 20 data sizes. As a result of the large number of configurations, the difference between encoding schemes across data sizes and features configurations is displayed in Figure 2 rather than presented in table form.

At the lowest data point, IO performs best, consistent with the notion that when there is minimal training data, a minimal encoding scheme performs best. At the highest data point, BILOU performs best, matching the findings of Ratinov and Roth (2009). The system performs similarly to many of the configurations evaluated in by Turian et al. (2010) which use Brown clusters or word embeddings. These findings are not shocking, but serve to demonstrate the capabilities of NERPy as an experimental framework.

3.2. Implementation Challenges

Working with word embeddings. NERPy originally used gensim (Řehůřek and Sojka, 2010) for loading word embeddings to generate features. However, this required loading vectors for all words in the embeddings (not just the training vocabulary), which is slow and memory-intensive. To address this, we developed the QuickVec (`https://github.com/ConstantineLignos/quickvec`) package as part of NERPy, which can instantaneously load word embeddings after a one-time conversion process, similar to Magnitude (Patel et al., 2018). It can be installed with no dependencies except numpy and can convert embeddings into its database format much faster than Magnitude, over three times faster for the 1 million-word vocabulary 300-dimensional embeddings used in the experiment described above.

Performance optimization. Due to CRFsuite's extremely fast C implementation of first-order linear chain CRF models, NER models can be trained rapidly. For example, using a standard untuned feature set (including 300-dimensional word embeddings) computed over a five-word window, training a model on the CoNLL 2003 English NER data using L-BFGS takes approximately 13 minutes, and the resulting model attains an entity of F1 of 88.21 on the test data. Substantial time was invested in optimizing NERPy's code to ensure that the code interfacing with the backend is as fast as possible, including doing line-level code profiling

to optimize frequently-called functions, such as those used in feature generation. However, especially when word embeddings are used, feature generation can take a significant amount of time (2.5 minutes for the training data in this example), as features for each position in each sentence are represented as individual dictionaries before being provided to the backend (which will typically use a more efficient representation). While it is possible to slightly improve performance while maintaining a pure Python implementation, major changes such as implementing parts of NERPy in C/C++ would come at the risk of making the code harder to extend, interact with, and install. Thus we plan to keep the current distinction of NERPy being pure Python but interfacing with backends that may be written in other languages.

4. Conclusion and Future Work

NERPy provides a flexible and accessible framework for named entity recognition that any user capable of using a command line could use to perform experiments in NER system design, and any user capable of using Python could use to create a prototype system. We have publicly released the code (`https://github.com/ConstantineLignos/nerpy`), and are in the process of completing the documentation of NERPy and QuickVec and releasing them on PyPI so that they are pip-installable.

There are many ways in which we believe that NERPy could be extended to further enable experimentation with NER system design and rapid prototyping. First, integration with a neural NER backend, such as NCRF++ and NeuroNER would enable a much broader set of experiments to be run. Integration is currently underway, and we look forward to releasing this soon. The challenge of integrating goes far beyond the software engineering required to merely "pipe" together systems; the configurations must be connected, and errors have to be handled robustly. While using another sequence model backend is relatively simple, connecting NERPy to another feature-rich NER toolkit is complex.

Second, in addition to industrial and novice-user applications, we believe NERPy could provide a reliable baseline for experimenting with NER in lower-resourced languages. Using NERPy, users can experiment with building systems before word embeddings are available, and later identifying the best ways to train their embeddings in the context of a simple, non-neural system before experimenting with more complex neural models, which are often more difficult to train due to the challenges of selecting hyperparameters using small amounts of data.

NERPy could be extended so that users could import pretrained NER models for various ontologies and languages to be used for rapid experimentation. While spaCy provides a similar function, its models are only available for a small set of languages and are chosen to reflect stable, common, ontologies as opposed to recording the research community's progress. We believe that NERPy can provide a framework for researchers to produce models for less commonly studied languages and NER tasks.

References

Dernoncourt, F., Lee, J. Y., and Szolovits, P. (2017). NeuroNER: an easy-to-use program for named-entity recognition based on neural networks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 97–102, Copenhagen, Denmark, September. Association for Computational Linguistics.

Honnibal, M. and Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. *To appear*.

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. (2018). Advances in pre-training distributed word representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May. European Language Resources Association (ELRA).

Okazaki, N. (2007). Crfsuite: a fast implementation of conditional random fields (CRFs).

Patel, A., Sands, A., Callison-Burch, C., and Apidianaki, M. (2018). Magnitude: A fast, efficient universal vector embedding utility package. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 120–126, Brussels, Belgium, November. Association for Computational Linguistics.

Ratinov, L. and Roth, D. (2009). Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, pages 147–155, Boulder, Colorado, June. Association for Computational Linguistics.

Řehůřek, R. and Sojka, P. (2010). Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May. ELRA. `http://is.muni.cz/publication/884893/en`.

Tjong Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Turian, J., Ratinov, L., and Bengio, Y. (2010). Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics.

Yang, J. and Zhang, Y. (2018). NCRF++: An open-source neural sequence labeling toolkit. In *Proceedings of ACL 2018, System Demonstrations*, pages 74–79, Melbourne, Australia, July. Association for Computational Linguistics.

Yang, J., Liang, S., and Zhang, Y. (2018). Design challenges and misconceptions in neural sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3879–3889, Santa Fe, New Mexico, USA, August. Association for Computational Linguistics.

Author Index