Daniel Filipe Rodrigues Henriques

Bachelor of Science

Automatic Completion of Text-based Tasks

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Engineering

Advisers: João Freitas, CTO, DefinedCrowd
João Moura Pires, Assistant Professor,
Faculdade de Ciências e Tecnologia
da Universidade Nova de Lisboa
Co-advisers: Rui Correia, Machine Learning Engineer,
DefinedCrowd
Filipe Marques, Associate Professor,
Faculdade de Ciências e Tecnologia
da Universidade Nova de Lisboa

Examination Committee

Chairperson: Carlos Damásio
Raporteur: João Magalhães

September, 2019

Automatic Completion of Text-based Tasks

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

to my beloved parents who gave me everything they had
without ever holding back.

Acknowledgements

I would first like to thank my thesis advisers João Freitas and Rui Correia from DefinedCrowd and professors João Moura Pires and Filipe Marques. The development of this thesis took place at DefinedCrowd. As a consequence, both João and Rui had to put a great deal of effort to coordinate their work with my thesis's orientation. I have learned plenty from the world of ML and NLP due to these two amazing mentors. I would like to give a special thanks to Rui, whose patience is infinite. Thank you for all the reviews, the tips, the corrections, and the healthy discussions. If today I can write a scientific paper is because of you.

However, both my professors have helped me whenever they could. Professor Filipe Marques was vital during the mathematical component of this thesis as his suggestions steer the investigation to a whole different level. Professor João Moura Pires helped me a lot with the visualizations and thanks to its recommendations, the plots in this thesis are not only informative but also beautiful.

I would like to thank the Computer Science Department and FCT for being my second house during this past 5 years. It was an honor to be able to conclude my studies at the best Portuguese engineering university.

I would also like to thank my coworkers at DefinedCrowd, especially, Jorge Ribeiro and Francisco Ganhão. Both of them helped me in this journey, and their support was core to keep pushing forward and conclude this achievement. I would like to give a special thanks to Jorge, who helped me to grow as a data scientist, an (almost) engineer, and a person.

I would also like to thank the friends that have made during this master's degree. Thanks to André Neves, Filipe Amador, Ivo Rocha, Pedro Almeida, Didier Dias, and Dinis Cabanas. Thank you all for your companionship and your ability to make me laugh.

Last but not least, I would like to profoundly thank my loving family. Thanks, mom and dad for all the sacrifices and for all the love. This degree is my most notorious and best achievement, and I could not have done it without you. I love you now and forever.

Abstract

Crowdsourcing is a widespread problem-solving model which consists in assigning tasks to an existing pool of workers in order to solve a problem, being a scalable alternative to hiring a group of experts for labeling high volumes of data. It can provide results that are similar in quality, with the advantage of achieving such standards in a faster and more efficient manner. Modern approaches to crowdsourcing use Machine Learning models to do the labeling of the data and request the crowd to validate the results.

Such approaches can only be applied if the data in which the model was trained (source data), and the data that needs labeling (target data) share some relation. Furthermore, since the model is not adapted to the target data, its predictions may produce a substantial amount of errors. Consequently, the validation of these predictions can be very time-consuming. In this thesis, we propose an approach that leverages in-domain data, which is a labeled portion of the target data, to adapt the model. The remainder of the data is labeled based on these model's predictions. The crowd is tasked with the generation of the in-domain data and the validation of the model's predictions. Under this approach, train the model with only in-domain data and with both in-domain data and data from an outer domain.

We apply these learning settings with the intent of optimizing a crowdsourcing pipeline for the area of Natural Language Processing, more concretely for the task of Named Entity Recognition (NER). This optimization relates to the effort required by the crowd to performed the NER task. The results of the experiments show that the usage of in-domain data achieves effort savings ranging from 6% to 53%. Furthermore, we such savings in nine distinct datasets, which demonstrates the robustness and application depth of this approach.

In conclusion, the in-domain data approach is capable of optimizing a crowdsourcing pipeline of NER. Furthermore, it has a broader range of use cases when compared to reusing a model to generate predictions in the target data.

Keywords: Crowdsourcing, Transfer Learning, Machine Learning, Named Entity Recognition, Natural Language Processing

Resumo

O *crowdsourcing* é um modelo de resolução de problemas que consiste na atribuição de tarefas a um grupo de trabalhadores para resolver um problema, sendo uma alternativa escalável à contratação de especialistas para classificar grandes volumes de dados. Este modelo é capaz de gerar resultados com qualidade semelhante e de uma forma mais rápida e eficiente. Abordagens recentes ao *crowdsourcing* utilizam modelos de Aprendizagem Automática para fazer a classificação dos dados que é depois validada pela *crowd*.

Tais abordagens só podem ser utilizadas se os dados de treino do modelo (dados fonte) e os dados não classificados (dados objetivo) partilharem alguma relação. Além disso, como o modelo não está adaptado aos dados objetivo, as suas previsões podem conter um elevado número de erros. Por consequência, a validação destas previsões é bastante demorada. Nesta tese, propomos uma abordagem que utiliza dados de domínio, que é uma parte dos dados objetivo que está classificada, para adaptar o modelo que classificará o resto dos dados objectivo. A *crowd* tem de gerar os dados de domínio e fazer a validação das previsões. Nesta abordagem, o modelo é treinado com dados de domínio e com dados de domínio em conjunto com dados de um domínio externo.

Esta abordagem é aplicada com o intuito de otimizar uma *pipeline* de *crowdsourcing* para Processamento de Linguagem Natural, mais concretamente, para o Reconhecimento de Entidades Mencionadas (REM). Esta otimização está relacionada com o esforço da *crowd* na realização da tarefa de REM. Os resultados das nossas experiências mostram que o uso de dados de domínio obtém diminuições de esforço que variam de 6% a 53%. Adicionalmente, estas diminuições em nove conjuntos de dados, o que demonstra a robustez e leque de aplicações desta abordagem.

Em suma, a abordagem proposta é capaz de otimizar uma *pipeline* de *crowdsourcing* de REM. Além disso, esta abordagem tem um espectro de casos de usos mais abrangente quando comparado às abordagens previamente definidas.

Palavras-chave: Crowdsourcing, Transferência de Conhecimento, Aprendizagem Automática, Reconhecimento de Entidades Mencionadas, Processamento de Linguagem Natural

Contents

List of Figures

List of Tables

Listings

Glossary

corpus	Collection of written texts
vocabulary	Body of words used in a particular language

Acronyms

AI	Artificial Intelligence
BiLSTM	bi-directional Long Short Term Memory
CBOW	continuous Bag-of-Words
CNN	Convolutional Neural Network
CoNLL	Computational Natural Language Learning
CRF	Conditional Random Fields
CRL	Communication Research Laboratory
CV	Computer Vision
DL	Deep Learning
FN	False Negative
FP	False Positive
GloVe	Global Vectors
GRU	Gated Recurrent Unit
HIT	Human Intelligence Task
HITEX	Human Intelligence Task execution
HMM	Hidden Markov Model
IAA	Inter-annotator agreement
IDD	in-domain data
IOB	Inside-Outside-Beginning
IOBES	Inside-Outside-Beginning-Ending-Singleton
IOE	Inside-Outside-Ending
IREX	International Retrieval and Extraction Exercise

LSTM	Long Short Term Memory
LSTM-LM	Long Short Term Memory-Language Model
MEM	Maximum Entropy Model
MET	Multilingual Entity Task
ML	Machine Learning
MLE	maximum likelihood estimation
MUC	Message Understanding Conference
MUC-7	Message Understanding Conference - 7
MUC-6	Message Understanding Conference - 6
NER	Named Entity Recognition
NICT	National Institute of Information and Communications Technology
NLP	Natural Language Processing
ODD	out-domain data
POS	part-of-speech
PTB	Penn Treebank
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SGD	stochastic gradient descent
SHL-MDNN	shared-hidden-layer multilingual Deep Neural Network
SVM	Support Vector Machine
tpm	tokens per minute
UI	user interface
WER	Word Error Rate
wpm	words per minute

Chapter 1

Introduction

Understanding diversity is imperative to understanding collective intelligence, and collective intelligence is an essential ingredient in one of the primary categories of crowdsourcing: the attempt to harness many people's knowledge in order to solve problems or predict future outcomes or help direct corporate strategy.

– Jeff Howe

1.1 Motivation

According to Forbes[1], 2.5 quintillion bytes of data are generated every day, 95% of which comes in unstructured form [22]. Companies, and organizations in general, have long since recognized the importance of automatically processing this information, to extract meaning from it. In recent years, this information processing has been by resorting to Machine Learning (ML) algorithms, in particular, supervised learning algorithms. These algorithms learn from previously labeled input-output pairs, which generates a constant necessity for human labeling in the process of training and refining those models.

Up until the mid-2000s, labeling was mostly done by experts. However, this approach does not scale with the amount of data. Having a large number of experts labeling data is costly; on the other hand, having too few renders the process inefficient. The notion of crowdsourcing emerged as a response to these issues. Howe [31], coined the term as "the act of taking a task traditionally performed by a designated agent (such as an employee or a contractor) and outsourcing it by making an open call to an undefined but large group of people."

[1] https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#301956d760ba (consulted on 1 February 2019).

Studies show that, for a group of well-defined tasks, the crowd is capable of providing similar quality results when compared to those produced by experts, with the advantage of achieving such standards faster and in a more cost-efficient manner [47, 59].

Recent approaches to crowdsourcing introduced a step of pre-labeling which is automatically performed by ML models [33, 60]. The pre-labeling process is used to improve the delivery time and, consequently, the costs associated with the crowd's effort. These approaches use models to pre-label the data, which is then validated by the crowd. The models are trained on datasets which are already labeled and reused to generate predictions on new non-labeled datasets – henceforth, we refer to this approach as the pre-trained model approach.

However, the pre-trained model approach has constraints when it comes to its application. The labeled data in which the model is trained needs to be similar or share a relationship with the data to be labeled. Otherwise, the pre-labeling process does not contribute to reducing delivery time, but rather to increase it. Such detail indicates that this process can be optimized.

Companies that rely upon the usage of crowdsourcing to serve its client's needs have an increased interested in optimizing this process. Often the characteristic of the data and the task that needs to be performed by the crowd depend upon the client's requirements. Therefore, these companies need to define strategies that allow applying ML techniques despite the properties of the data and the underlying task.

1.2 Problem Statement

DefinedCrowd is a startup company that leverages crowdsourcing to fulfill the needs of its clients. DefinedCrowd has a crowdsourcing platform, Neevo, where the crowd can perform data labeling tasks. At DefinedCrowd, the tasks are very heterogeneous as they depend upon the requirements of the clients.

For instance, tasks performed in written text can vary in terms of the label set, the type of text, and the language of the text. Finding a model that is trained on a labeled dataset that corresponds precisely to a client's requirements is unfeasible and invalidates the possibility of using this approach. A better approach would be to adapt the model to each different task. The following steps indicate how a model can be adapted to a specific task:

1. require the crowd to label only a portion of the data;
2. train a model using only that portion of labeled data;
3. use the model to generate predictions for the remainder of the data;
4. and, use the crowd to validate such predictions.

In this thesis, we investigate the feasibility of this strategy in the area of Natural Language Processing (NLP), more concretely the task of Named Entity Recognition (NER). NER is the task of extracting and annotating words or phrases, in a text, into categories that map to real-world concepts, such as persons, locations, and quantities. At DefinedCrowd, the NER pipeline is one of the most request tasks, which, as a consequence, generates a large amount of data that can be leveraged for our investigation.

In this thesis, we consider a crowdsourcing pipeline that is divided into generation and validation. In the generation step, the crowd is tasked with the annotation of entities in non-annotated unstructured text. In the validation step, the crowd is tasked with the validation of the annotations performed in the generation step.

Therefore, the process of adapting a model to each task has two associated costs: generation and validation. The generation cost relates to the amount of training data, and the validation cost relates to the time the crowd spends correcting the model's predictions. However, we do not know what which point the model should be trained, and when the generation step should be stopped. Furthermore, we need a way of estimating how much time it takes to generate the training data and how much time it takes to validate the model's predictions. Having a solution to estimate these costs enables the assessment of the range of optimization that is obtained by leveraging the in-domain approach.

To better understand these issues, one needs to study the relation between the amount of training data and the model's performance. Herein, we need to understand how does the amount of training data translates into crowd effort during the generation step, and how does the model's performance translate into crowd effort during the validation.

Ideally, the crowd should generate the least possible amount of training data that allows for training the best possible model to generate predictions for the remainder of the data. Therefore, testing the usage of Transfer Learning falls as a logical path of investigation.

The application of Transfer Learning strategies to the model's training enables the achievement of better performances while requiring fewer amounts of training data [63, 64]. One such strategy is training the model on two datasets simultaneously, meaning that the model trains with domain data and out of domain data. The domain data is a portion of the data that has been annotated by the crowd, while the out of domain data is a fully annotated dataset.

1.3 Research Questions

The goal of this thesis is the study the relationship between the amount of training, the model's performance, and the generation and validation costs associated with both these variables. With this in mind, the following research questions were formulated:

- **Considering the state of the art regarding NER, how does the amount of training data influence the model's performance?**

A model trained with a portion of the data can produce better predictions than a pre-trained model. However, it is unclear what is the amount of domain data that should be generated to start the model's training. The optimum amount of training data is reached when the addition of more data does not contribute significantly to improve the model's performance.

- **How can the in-domain data approach reduce time costs when integrated into a crowdsourcing pipeline?**

  Answering the first research question provide us with a measure of performance associated with the amount of training data. Herein, we want to estimate the cost of generating that amount of data and estimate the cost of correcting the model's predictions. Having these costs into consideration enables to determine the point from which the model's training should be started and measure the extent of the achieve optimizations.

1.4 Document Structure

The remainder of this document is structured as follows:

- **Chapter 2** presents the background and related work for the three main areas in this thesis. First, we present a historical overview and the types of crowdsourcing. We also explain DefinedCrowd's crowdsourcing platform. The concept of NER is explained in detail (including formal problem definition, format, and evaluation metrics), and we also the present state of the art solutions regarding this task. The concept of Transfer Learning is also described, as well as its various types and a more thorough description of the application of Transfer Learning to NER is provided;

- **Chapter 3** describes the experiment that answers the first research question. In this chapter, we implement a state of the art NER that is used to show the relationship between the amount of training data and the model's performance. We compare these results against the pre-trained model approach;

- **Chapter 4** describes a second experiment, which answers the second research question. In this experiment, we conduct a statistical analysis to estimate the necessary effort to perform a NER generation task. We proceed to assess the efforts associated with the model's predictions and present the gains of using our approaches when compared to the traditional crowdsourcing approach;

- **Chapter 5** presents the conclusions and the future work based on the answers of the research questions.

Chapter 2

Background & Related Work

Study the past if you would define the future.
– Confucius

Before proceeding with the investigation, we present the related and work and state of the art regarding the concepts of crowdsourcing, NER, and Transfer Learning. Additionally, we look into related work that has connected these areas. This chapter is organized as follows:

- **Section 2.1** presents a definition of crowdsourcing, the various types of crowdsourcing, the products, and platforms that incorporate this concept and the description of DefinedCrowd's platform, Neevo[1];
- **Section 2.2** presents a definition of the NER task, how it is performed and evaluated. Following this, we describe a range of datasets related to NER. Then we present automated solutions for NER, starting with ML models that depend upon hand-engineered features and finishing with Deep Learning (DL)-based systems. In the end, state of the art solutions are presented;
- **Section 2.3** presents a definition and taxonomy of Transfer Learning, as well as applications of this concept to the areas of Computer Vision (CV), speech, and NLP that show the advantages of using Transfer Learning.

[1] https://neevo.definedcrowd.com/en-us/community/ (consulted on 21 February 2019).

2.1 Crowdsourcing

Crowdsourcing is a widespread problem-solving model which consists of assigning tasks to an existing pool of contributors to solve a problem. Crowdsourcing combines various opinions and answers to reach the problem's solution. It is possible to trace the informal use of crowdsourcing back to 1714. At that time, the British government created the Longitude Prize, awarding a monetary reward to whoever was able to find the best solution to measure a ship's longitudinal position at sea [15].

The formal definition of crowdsourcing, however, was first introduced by Howe in 2006 [30]. In his article, Howe presents the iStockphoto[2] use case and how crowdsourcing changed the stock photography marketplace, by allowing stock photos, whose market value could reach 600 dollars, to be marketed at the one dollar mark. The rationale behind this shift was that professional-grade cameras were starting to cost way less than what they used to and that it was possible to create a crowd of amateurs that could rival professionals. Furthermore, iStockers (contributors of iStockphoto) did not need to profit as much, because this was not their full-time occupation; instead, it was a hobby.

Since 2006, a wide range of companies, universities, and institutions has used crowdsourcing in a variety of ways, and thus a typology was defined to categorize the different uses of crowdsourcing in today's applications [8]:

- Knowledge Discovery and Management - the task of finding, collecting, and organizing information into a common location and format. Examples include SeeClickFix[3] (a reporting system for citizens to report maintenance needs in their neighborhood) or Wikipedia (a non-profitable organization that centralizes an encyclopedia where the crowd generates and reviews content);

- Broadcast Search - the task of solving empirical problems. Examples include InnoCentive[4] (a platform that breaks down problems into challenges that are solved by the crowd to find solutions for these problems);

- Peer-vetted Creative Production - the task of creating and selecting creative ideas. Examples include iStockphoto (a platform that collects stock photos) or LEGO[5] (the company allowed users to submit new products, via the website, which other users could vote for);

- Distributed Human Intelligence Tasking - the task of analyzing large amounts of information. Examples include Amazon Mechanical Turk[6], or Neevo[7].

[2] https://www.istockphoto.com/pt (consulted on 11 January 2019).
[3] https://seeclickfix.com/ (consulted on 18 February 2019).
[4] https://www.innocentive.com/ (consulted on 18 February 2019).
[5] https://shop.lego.com/en-GB/ (consulted on 22 February 2019).
[6] https://www.mturk.com/ (consulted on 22 February 2019).
[7] https://neevo.definedcrowd.com/en-us/community/ (consulted on 22 February 2019).

According to the typology above, this thesis is centered around the notion of distributed human intelligence tasking. The remainder of this section details how this crowdsourcing type works.

2.1.1 Distributed Human Intelligence Tasking

The distributed human intelligence tasking is a type of crowdsourcing that aims to process high volumes of data by resorting to the crowd. This processing task is, typically, decomposed in micro-tasks, which is the atomic unit of work in this context. These micro-tasks are assigned to a pool of workers that are tasked to complete them. In this section, we describe Neevo, which is a platform that monetarily rewards workers for the completion of these micro-tasks. However, we need to define the terminology and the typical setting of a crowdsourcing pipeline.

The following terminology is based on Neevo. However, it is general enough to describe any crowdsourcing pipeline.

- Contributor - the worker that performs the micro-task;
- Human Intelligence Task (HIT) - the task to be performed by the contributors, each HIT can be assigned to one or more contributor. Often, we refer to the execution of one HIT by one contributor as Human Intelligence Task execution (HITEX).
- Job - a collection of HITs. Each job has an associated type, which indicates the type of tasks to be executed in that job. Types can be related to NLP, such as NER and semantic annotation; CV, such as image collection and image tagging; and speech technologies, such as transcription and speech data collection;
- Project - a collection of jobs. Clients, often ask for complex and specific pipelines that require jobs to be executed one after another. For instance, clients may require the task of NER to be performed on speech data. However, they do not possess the data. Due to this, the client may require a job of speech data collection followed by a transcription job, and, finally, the NER job.

Having defined the crowdsourcing taxonomy, we proceed to describe a standard crowdsourcing pipeline. Figure 2.1 shows a standard crowdsourcing pipeline consisting of two steps:

- Generation step - the part of the pipeline in which the contributors perform the HITs. This is the step in which the generation of annotations happens;
- Validation step - the part of the pipeline in which the results of the generation step are validated and, possibly, corrected. It is also possible for this validation step to not be present in a pipeline.

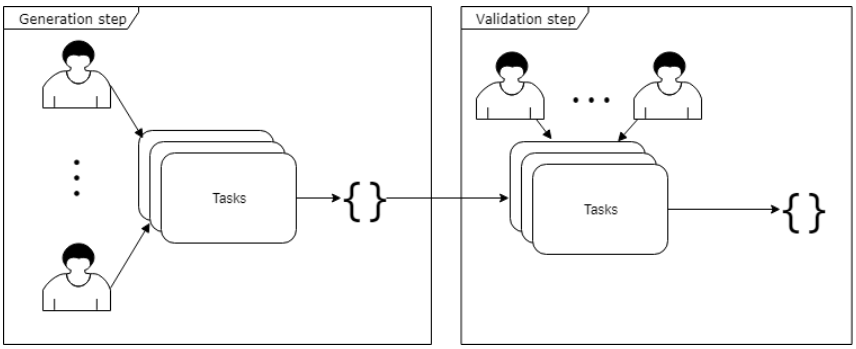


Figure 2.1: A crowdsourcing pipeline consisting of two parts: a generation and a validation step. In the generation step, contributors are tasked with the labeling of data and, in the validation step, users proceed to validate and correct, if necessary, the results of the generation step.

The need for a validation step arises because the crowdsourcing approach allows anyone to solve tasks. Therefore, this validation step is a way of ensuring quality in the results of the generation step [26]. This step is but one approach to ensure that contributors are not malicious or underperforming, and thus ensuring the quality of the results. Two techniques that can be applied to fulfill this goal are: contributor screening and infer contributor's trust [46].

Contributor screening is the application of a test before starting the job. It can be seen as a qualification test, meaning that a contributor answers a set of questions, and if he gets more than a certain percentage (defined upon job creation), then it is assumed that he is going to perform well. This approach can filter malicious or unskilled contributors even before they start the job, which has the advantage of not having to cancel the results perform by this type of contributors.

Inferring contributor's trust is the application of tests, that are based on a gold set, during the execution of a task. A gold set is a set of pairs of questions and answers that are known to be right (a ground truth). When the contributors are performing a particular task, they get prompted with these tests that assess the performance of the contributor. Typically, these tests are injected during the execution of the HIT, and contributors do not know that they are being evaluated. If the contributor's performance starts to decay, he gets excluded from the job, and his results are discarded.

2.2 Named Entity Recognition

The task of NER was first introduced by Grishman and Sundheim [24] during the Message Understanding Conference - 6 (MUC-6). NER is the task of extracting and classifying words or phrases, in a text, into categories that map to real-world concepts, such as persons, locations, and quantities. Table 2.1 depicts the first 7 categories defined for the task of NER and examples of entities for each categories.

Named Entity	Example
ORGANIZATION	United Nations, The International Maritime Organization
PERSON	Barack Obama, Marcelo Rebelo de Sousa
LOCATION	Portugal, London, Kilimanjaro
DATE	March 5, 1996; Today; 10/4/2017
TIME	5 AM, midday, 23:40
MONEY	55 euros, $12, 109 ¥
PERCENT	14%, a quarter, a half

Table 2.1: NER categories and examples. The categories are the ones defined in [24].

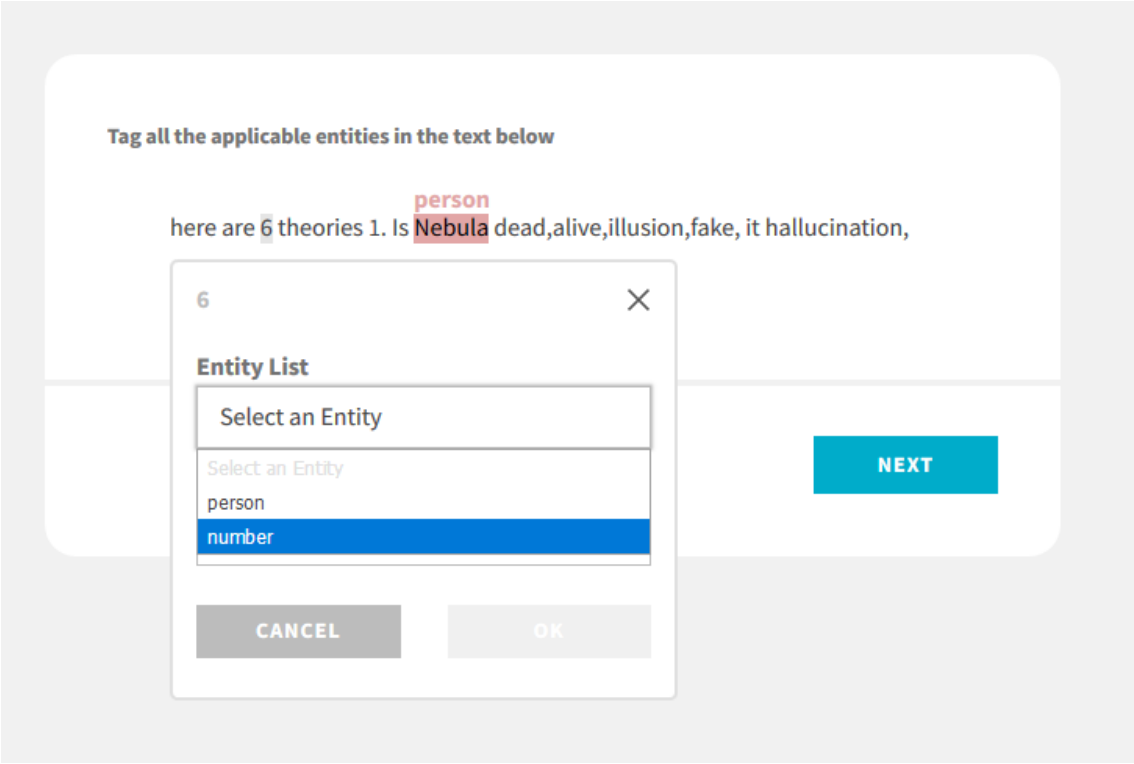


Figure 2.2: user interface (UI) for the task of NER at Neevo.

NER is a common task that is often used as an intermediate step for the completion of more complex tasks, such as question answering and summarization.

Figure 2.2 illustrates an example of a UI for the task of NER at Neevo. The users can click and drag to select the part of the text that they wish to tag. After selection, the user can choose a category from the drop-down list. This action can be executed as many times as necessary and, when there is nothing else to annotate, the user can click next and annotate the following text.

NER is an area where a wide range of work has been performed and, for that reason, there was the need to define annotation schemes to perform this task and evaluation metrics to assess the quality of the results. In this section, we define the annotation schemes, the evaluation metrics, and automatic approaches to the problem used in the task of NER.

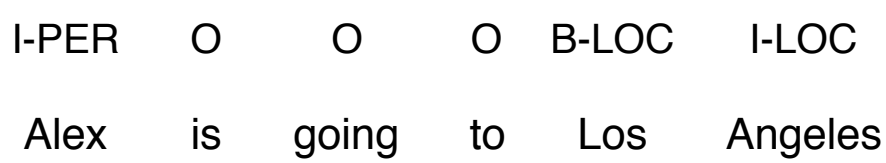


Figure 2.3: The IOB annotation scheme. The prefix I denotes entities composed of only one token and the tokens in the middle of entities which are composed of more than two tokens; the prefix O denotes tokens which are not entities; the prefix B denotes the first token of entities which have more than one token.

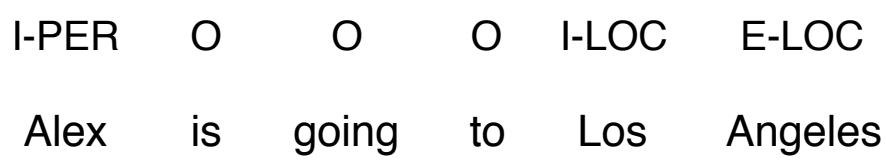


Figure 2.4: The IOE annotation scheme. The prefix I denotes entities composed of only one token and the tokens in the middle of entities which are composed of more than two tokens; the prefix O denotes tokens which are not entities; the prefix E denotes the last token of entities which have more than one token.

2.2.1 Annotation Schemes

From a practical standpoint, the goal of NER is the classification of each token or phrase in a text. With this in mind, the NER community developed and created consensus around a set of annotation schemes. The annotations generated by these schemes are also tailored to be used as training data to ML algorithms. In each scheme, a token is classified with a label consisting of a prefix and the category they belong. This prefix is used to relate the token with the structure of the entity. Examples of these schemes are:

- Inside-Outside-Beginning (IOB) (figure 2.3) - the prefix I denotes singleton entities, which are entities composed of only one token, and the tokens in the middle of entities which are composed of more than two tokens; the prefix O denotes tokens which are not entities; the prefix B denotes the first token of entities which have more than one token. A variation called IOB2 uses the prefix B to denote singleton entities;

- Inside-Outside-Ending (IOE) (figure 2.4) - as in the IOB format, the I and O prefixes are used in the same context. The prefix E denotes the last token of entities which have more than one token. A variation called IOE2 uses the E prefix to denote singleton entities;

- Inside-Outside-Beginning-Ending-Singleton (IOBES) (figure 2.5) - all the prefixes described previously apply in the same situations in this annotation scheme. When a label is composed of more than two tokens, it starts with the prefix B and ends with the prefix E. Singleton entities are denoted with the prefix S.

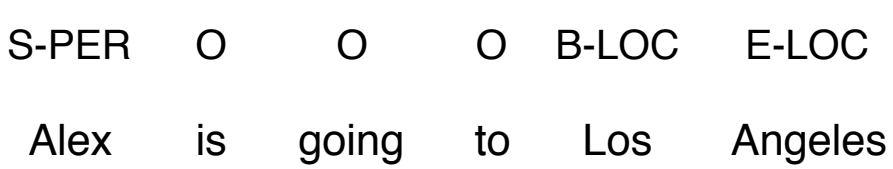


Figure 2.5: The IOBES annotation scheme. The prefix I denotes tokens in the middle of entities which have more than two tokens; the prefix O denotes tokens which are not entities; the prefix B denotes the first token of entities which have more than one token; the prefix E denotes the last token of entities which have more than one token; the prefix S denotes entities composed of only one token.

Error	Explanation	Output	Correct Output
False Negative (FN)	The system misses an entity	Robert (O)	Robert (I-PER)
Category + Boundary	The system predicts an entity with the wrong label and boundaries	in (B-PER) New (I-PER) York (I-PER)	New (B-LOC) York (I-LOC)
Boundary	The system predicts an entity with the wrong boundaries	George (B-PER) R. (I-PER)	George (B-PER) R. (I-PER) R. (I-PER) Martin (I-PER)
Category	The system predicts an entity with the wrong label	John (B-LOC) Briggs (I-LOC)	John (B-PER) Briggs (I-PER)
False Positive (FP)	The system predicts an entity where it should have not done it	Unlike (I-LOC)	Unlike

Table 2.2: Error taxonomy for NER (adapted from [45]). The "O" indicates the absence of category.

2.2.2 Evaluation Metrics

The task of NER can be solved automatically by recurring to ML systems. Therefore, it was necessary to have systematic measures that could represent the quality of these systems and compare performance between them. Before moving to measure performance, it is essential to define which type of errors may arise in what concerns NER.

Table 2.2 shows a taxonomy of NER [45]. This taxonomy has five different types of errors. A FN is an entity that should have been predicted by the system, but it was not. The opposite of this error is the FP. The remainder of the errors relates to the category and boundaries of entities. Given this errors, we present three general metrics that measure model performance in classification problems – precision, recall, and F_1 score – and the score which has been developed to measure performance in NER tasks, the Message

Understanding Conference (MUC) score.

Precision is the percentage of named entities that were classified correctly by the system. Note that, for an entity to be considered correct, it needs to have both category and boundary correct. It can be calculated using equation 2.1.

$$precision = \frac{CNE}{CNE + WNE} \tag{2.1}$$

In equation 2.1, CNE stands for correct named entities, and WNE stands for wrong named entities. Another metric is the recall, which is the percentage of true entities predicted by the system, and it can be calculated using equation 2.2.

$$recall = \frac{CNE}{CNE + UNE} \tag{2.2}$$

In equation 2.2, UNE stands for an unclassified named entity. Having the precision and recall, we can compute the harmonic between the two metrics, which is the F_1 score, by using equation 2.3.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.3}$$

However, these metrics do not account for each one of the errors in table 2.2 because the errors associated with the category and boundaries of the entity assume that entities can be partially correct. For instance, a FN means that both category and boundaries are wrong, which implies that predicting either the category or boundaries correctly should contribute to improving the overall score.

This necessity allowed the creation of the MUC score[8] which is a micro-averaged F_1 score. In this score, named entities are considered correct according to two values: the type (category) and the text (boundaries). Having a two-axis evaluation means that any entity can be considered partially correct if either the type of the text is correct; consequently, this score takes into consideration the errors in table 2.2. The final score is calculated as the F_1 score. E.g., for the phrase depicted in figure 2.3, not classifying the token *Los* as a location would produce a precision of 0.75 and a recall of 0.75, which produces a MUC score of 0.75; computing this by recurring to the traditional definition of precision, recall, and F_1 score would produce the values 0.5, 0.5, and 0.5, respectively.

2.2.3 Annotated Corpora

Throughout the years a wide range of datasets has been proposed to solve specific tasks related to them, or to evaluate the performance of NER systems. In this section, we present a brief description of corpora that are relevant to NER.

The MUC-6 was the first conference ever to define a NER task. The task defined consisted of classifying news reports about labor contract negotiations and corporate

[8] An implementation of this score can be consulted on https://github.com/jantrienes/nereval (consulted on 19 February 2019).

management succession into six categories. These reports were written in English and were retracted from the Wall Street Journal [24]. Later, the Multilingual Entity Task (MET) applied this task to news articles in three distinct languages: Spanish, Japanese, and Chinese [41].

In 1998, during the Message Understanding Conference - 7 (MUC-7) the task aforementioned was applied to a new corpus, whose subject was that was about airplane crashes and rocket missile launches. Similarly, this task was adapted in the MET-2 to the languages of Japanese and Chinese [9].

The International Retrieval and Extraction Exercise (IREX) is an evaluation-based project for information retrieval and extraction in Japanese. This task allowed participants to classify named entities into 8 categories. There were three kinds of training data used in this task: the dry run data, which had 46 articles; the Communication Research Laboratory (CRL)[9] data, which had 1174 articles; and the formal run restricted domain data, which had 23 articles. The dry run dataset did not have a specific domain; instead, it covered a range of topics. The formal run restricted domain data's domain was related to police arrests [57].

In the Computational Natural Language Learning (CoNLL) 2002, it was created a NER task for the languages of Spanish and Dutch and the set of categories consisted of 4 categories. The data provided for this conference consisted of six files, three for each language: the training file, development file, and test file. The data files for both languages came from news articles [53].

The CoNLL 2003 extended the CoNLL 2002 task to the languages of English and German. For each one, there were four different data files: the training file, development file, test file, and a non-annotated data file. The English corpus consisted of 1,393 news articles and contained about 300,000 tokens, while the German corpus consisted of 909 news articles that contained about 310,000 tokens. The non-annotated data contained 17 million tokens, for the English language, and 14 million tokens, for the German language [55].

The OntoNotes v5.0 is a multilingual corpora that has been manually annotated with syntactic, semantic, discourse information, and 18 NER categories. The corpus supports three languages: English, Chinese, and Arabic. The English portion of the corpus has 1.7 million words, the Chinese portion has around 1 million words, and the Arabic portion has 300 thousand words. The English and Chinese corpus were composed of newswire articles, magazine articles, broadcast news, broadcast conversations, web data, and conversational speech data, while the Arabic corpus consisted of newswire articles [51].

[9]Now known as National Institute of Information and Communications Technology (NICT).

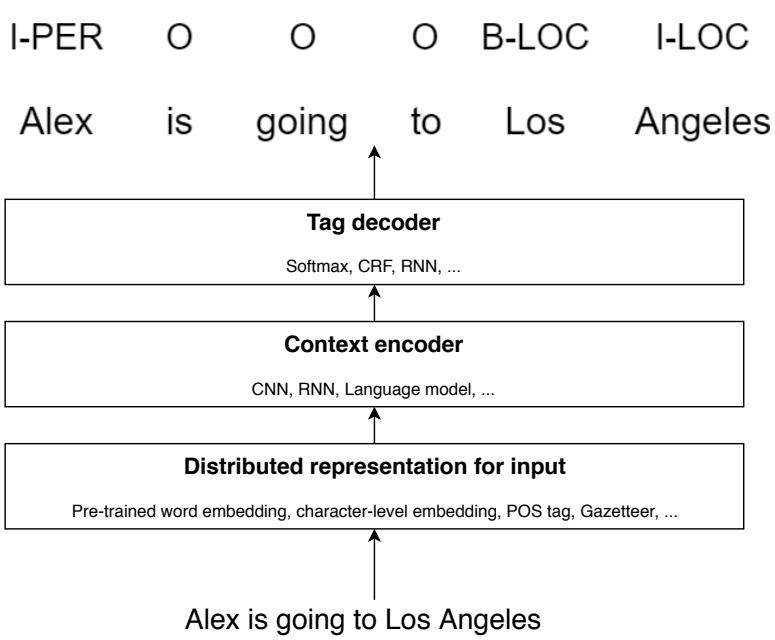


Figure 2.6: High level architecture of DL systems for NER (adapted from [37]).

2.2.4 Machine Learning for NER

ML models have been used to tackle the task of NER. Traditional approaches were based on Hidden Markov Model (HMM) [6], Maximum Entropy Model (MEM) [7], decision trees [58], Support Vector Machine (SVM) [4], and Conditional Random Fields (CRF) [40]. These approaches were able to achieve near-human performance (having an F_1 score of more than 90%) [41]. However, more recent approaches use DL techniques. In this section, we describe these DL systems and their components; this description is inspired in the eloquent work performed by Li et al. [37].

Before proceeding to the explanation of the DL systems, it is worth mentioning the reasons as to why NER benefits from the usage of DL techniques:

- DL systems can learn more complex features from the data due to its non-linearity nature, whereas traditional supervised learning approaches rely upon hand-engineered features which need a significant amount of work as well as expertise to generate;
- The training process of deep neural networks enables the design of complex NER systems.

In figure 2.6, the DL system receives as input a sentence and outputs a classification for each token. However, the context encoder can not process the sentence as a string. It can only process numerical values; thus, the tokens need to be represented as vectors of numerical values.

One way to represent tokens is with the one-hot vector representation, which associates a vector with dimension $\mathcal{V}$, where $\mathcal{V}$ is the size of the vocabulary, to each token in the corpus. Each vector has the value 1 in the token's corresponding position while

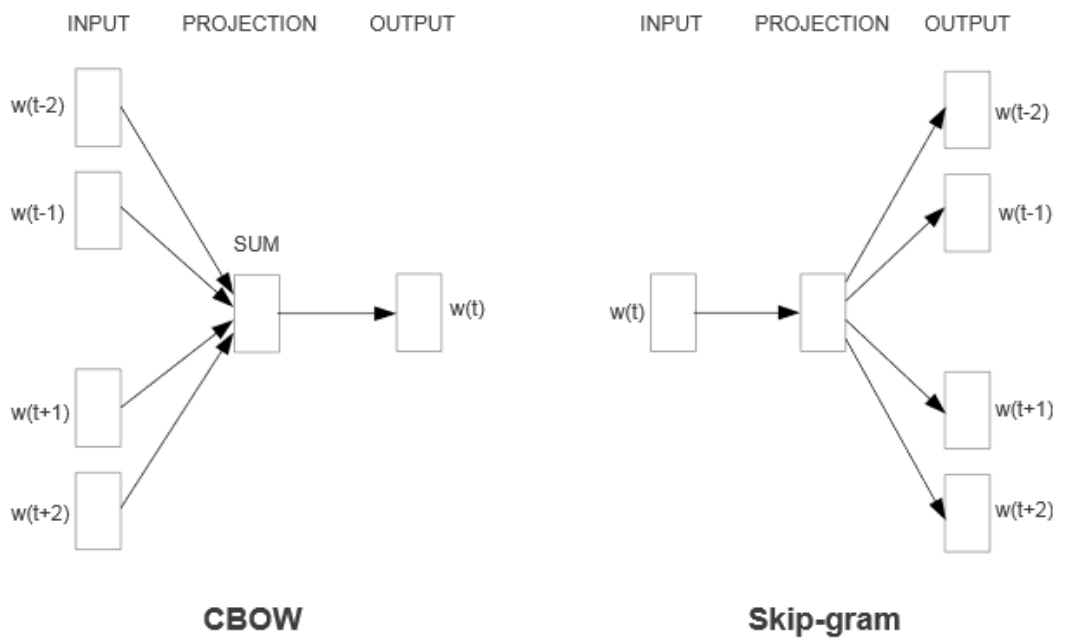


Figure 2.7: Two unsupervised learning algorithms that generate word embeddings: the CBOW and Skip-gram model (retrieved from [43]).

the rest of the vector is filled with 0's. If a corpus has 6 billion tokens and vocabulary with 400 thousand tokens, then the representation of the tokens input to be fed into the context encoder would have 6 billion rows per 400 thousand columns.

A more efficient way of representing tokens is to represent them in a distributed manner, meaning that each token is embedded in a d-dimensional space, where d is defined apriori. These embeddings can learn the syntactic and semantic context of each token and are, often, achieved through unsupervised learning algorithms that are applied to a large corpus. Two examples of models that are used to generate these representations are the continuous Bag-of-Words (CBOW) model and continuous Skip-gram model [43]. Both models' architectures can be seen in figure 2.7.

Intuitively, the CBOW generates a representation of a token given the surrounding tokens, so the syntactic and semantic meaning of one token depends upon its neighbors, and the Skip-gram model does the opposite and predicts the surrounding tokens' meaning based on the current token. A range of software can be used to produce these embeddings, or even reuse existing ones, such as fastText[10] from Facebook, GloVe[11] from Stanford, SENNA[12], and Word2Vec[13] from Google [42].

This distributed representation applies to words; however, character-level distributed

[10]https://fasttext.cc/docs/en/english-vectors.html (consulted on 23 January 2019).
[11]https://nlp.stanford.edu/projects/glove/ (consulted on 23 January 2019).
[12]https://ronan.collobert.com/senna/ (consulted on 23 January 2019).
[13]https://code.google.com/archive/p/word2vec/ (consulted on 23 January 2019).

representations are also possible. These representations are useful because they encode information about prefixes and suffixes, infer representations for unseen words and retrieve information at a sub-word level. Usually, there are two types of neural networks that are used to obtain these representations: Convolutional Neural Network (CNN) [39] and Recurrent Neural Network (RNN) [62].

The context encoder is the step in which a DL system learns from the distributed input representations. In this section, we describe the concept behind CNNs and RNNs, which are the most common choices for this step.

A CNN is an artificial neural network composed of feature learning and classification layers. Typically, the feature learning layers are stacks of three layers: a convolutional, activation, and a pooling layer. The convolutional layer applies filters to the input (this is just the dot product between matrices), enabling the detection of patterns. This application of filters, called convolution, is what enables the neural network to make a feature extraction. The output of this layer is sent into an activation layer, typically a Rectified Linear Unit (ReLU) layer, to introduce non-linearity into the system. Finally, the output of this layer is then sent into the pooling layer, which retracts the essential parts of the input and builds an output with them. When this output reaches the classification group of layers it is flattened (gets a dimensional reduction) so all the features learned can be kept and used for classification. At the end of this group of layers, the classification of the input is produced.

RNNs are a type of artificial neural network that possesses internal memory; such allows RNNs to decide the representation of the current input while considering knowledge acquired from previous inputs. This characteristic enables RNNs to excel at solving sequential data problems, such as NER. However, these neural networks suffer from the vanishing and exploding gradient problems [28, 49]. Due to these problems, two variants of RNNs were proposed: the Long Short Term Memory (LSTM) [29] and the Gated Recurrent Unit (GRU) [11].

The output of these neural networks is sent into a tag decoder component. This component input is the contextual representation of the data, which is decoded and mapped into categories that classify each token. One standard tag decoder is the CRF. A CRF is a type of discriminative model that makes predictions of a label y for a given observation x by computing the conditional probability distribution $P(y|x)$. However, the CRF is capable of predicting a sequence of labels by taking into account a sequence of inputs, allowing the prediction of a specific label based on previous ones.

A broad spectrum of systems that use DL techniques have been proposed throughout the years to solve NER. In the remainder of this section, we describe DL systems that obtained state of the art results, giving an insight into the architectures used and how well they performed.

DL systems for NER started to appear with Collobert et al. [14]. The proposed model obtained an F_1 score of 89.86%, in the CoNLL 2003 English shared task, with almost no feature engineering associated. This model used a CNN as a context encoder and a CRF

as a tag decoder. This model also resorted to word embeddings which were generated by a system called SENNA.

Lample et al. [36] used character embeddings generated by an LSTM unit and word embeddings that were generated by recurring to SENNA. These representations were then sent into another LSTM which outputted the encoded representation into a CRF. This model obtained an F_1 score of 90.94% in the CoNLL 2003 English shared task.

Ma and Hovy [39] used a CNN to obtain character embeddings and used the GloVe word embeddings. These embeddings were processed and sent into a bi-directional Long Short Term Memory (BiLSTM) which would then send its output into a CRF. A BiLSTM is a context encoder that uses two LSTM units that process the input forward and backward; both representations are concatenated meaning that each token's representation encodes information about the past and future tokens. This model obtained an F_1 score of 91.21% in the CoNLL 2003 English shared task.

In 2018, Akbik et al. [3] used character-level word embeddings, called contextual string embeddings, which main advantage is the fact that the same token can have different representations depending on the context in which it is used. This type of embeddings was obtained by using a Long Short Term Memory-Language Model (LSTM-LM). The context encoder was an LSTM, and the tag decoder was a CRF. This model obtained F_1 scores of 93.09% and 88.32%, in the CoNLL 2003 English and German shared tasks, respectively.

2.3 Transfer Learning

In the psychology field, Transfer Learning is the use and application of prior learning when faced with learning something new. This concept is the foundation of learning, thinking, and problem-solving [19]. When people learn how to execute a specific task, they can abstract this knowledge in such a way that it can be reused to complete other tasks in a faster and improved manner.

In the field of Artificial Intelligence (AI), the phenomenon of Transfer Learning relates to reusing previously acquired knowledge, such as data or ML models, to solve new tasks. This learning setting requires fewer data to train a model when compared to the traditional ML learning setting. In this section, we provide a theoretical overview of Transfer Learning as well as its applications to text-based tasks.

2.3.1 Formal Definition and Taxonomy

This section is based on the work performed by Pan and Yang [48].

Given a source domain $\mathcal{D}_S$, a source task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and a target task $\mathcal{T}_T$, Transfer Learning aims to improve the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ by using the knowledge acquired in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$. A domain $\mathcal{D}$ is the conjunction of the feature space $\mathcal{X}$ and the marginal probability distribution $P(X)$,

where $X = \{x_1, ..., x_n\} \in \mathcal{X}$. A task is associated with a domain, $\mathcal{D} = \{\mathcal{X}, P(X)\}$, and is the combination of the label space $\mathcal{Y}$ and the objective predictive function $f(\cdot)$, thus can be represented as $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$. This function is not known but can be approximated by training an ML model with label input-output pairs of the type $\{x_i, y_i\}$ is such that $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$.

With this formal definition, it is possible to identify three Transfer Learning settings:

- Inductive Transfer Learning - applies when the source and target tasks are different, despite the relationship between domains;
- Transductive Transfer Learning - applies when the source and target task are the same, while the domains are different;
- Unsupervised Transfer Learning - applies in the same situations as the inductive transfer learning, with the difference that this setting tries to solve unsupervised learning tasks in the target domain D_T.

The inductive transfer learning setting requires that the data in the target domain D_T is labeled to induce an objective target predictive function $f_T(\cdot)$. This setting considers situations regarding the availability of labeled data in the source domain D_S. The transductive transfer learning approach does not require data in the target domain D_T to be labeled; however, the data the source domain D_S should be abundant in labels. This approach can be further refined concerning the difference between the domains. The latter approach is the unsupervised transfer learning setting, whose difference from the remainder approaches relates to the lack of labeled data in both source and target domains.

Given these settings and based on the question "What to transfer?" one can choose from four different approaches:

- Instance-transfer - this approach transfers data points from the source domain D_S to the target domain D_T by re-weighting;
- Feature-representation-transfer - this approach transfers the feature representation, such that it reduces the difference between the source and target domains, or the classification or regression error. The knowledge transferred between domains is encoded in this feature representation;
- Parameter-transfer - this approach transfers a set of shared parameters or prior distributions of the hyper-parameters that can be transferred;
- Relational-knowledge-transfer - this approach transfers a relationship that is common to both source and target domains.

Learning Setting	Domains	Tasks	Source Labels	Target Labels
Traditional Machine Learning	=	=	NA	NA
Inductive Transfer Learning	NA	≠	NA	✓
Transductive Transfer Learning	≠	=	✗	✓
Unsupervised Transfer Learning	NA	=	✗	✗

Table 2.3: A resume of Transfer Learning settings and their relation with the traditional ML approach. The letters "NA" mean indifferent or irrelevant, also the symbol ✓ for source/target labels means that they are available, thus the symbol ✗ should mean the opposite. This table is a combination of table 1 and 2 from [48]

	Inductive Transfer Learning	Transductive Transfer Learning	Unsupervised Transfer Learning
Instance-transfer	✓	✓	
Feature-representation-transfer	✓	✓	✓
Parameter-transfer	✓		
Relational-knowledge-transfer	✓		

Table 2.4: Depending on the settings and the situation, there is a variety of approaches that can be used. This table was retracted from [48].

Table 2.3 shows the relation of the Transfer Learning settings and the traditional ML approach. Table 2.4 shows when it is adequate to use each approach regarding each setting. Having these relations into account is paramount to avoid negative transfer, which is the tendency that a neural network has of forgetting previously learned knowledge upon learning new knowledge.

2.3.2 Applications and Practical Use

The following section is about the applications of Transfer Learning in the areas of CV, speech technologies, and NLP. We present an example for each one of the areas.

In the area of CV, Transfer Learning has been used to facilitate the training of CNNs. Ahmed et al. [2] used pseudo-tasks, which are automatically constructed from data without supervision, to induce prior knowledge into the neural network's training. This model was tested against the Caltech-101 dataset (used for object recognition) which has 102 object categories, with 31 to 800 images per category. The model was trained with 15 and 30 images per category, obtaining scores of 58.1% and 67.2%, respectively. Training without Transfer Learning only wielded scores of 23.9% and 25.1%.

In the field of speech technologies, Huang et al. [32] proposed that the hidden layers in deep neural networks could be shared across languages for the task of speech recognition. The authors trained a shared-hidden-layer multilingual Deep Neural Network (SHL-MDNN) in four European languages (adapting the final layer to each language) and then used this model to recognize English and Chinese speech; this could be achieved

by retracting the shared-hidden-layers from the SHL-MDNN and adding a language-dependent layer on top of them.

The SHL-MDNN reduced Word Error Rate (WER) for the four European languages by 3-5%. Regarding the English data, the WER was reduced by 6-28%, and the Chinese data was reduced by 8-21%. These results are the improvements registered against the results achieved by language-specific deep neural networks.

The remainder of this section is focused on the application of Transfer Learning for NLP; this section is inspired by the work performed by Yang et al. [64]. Yang et al. propose a DL system that resorts to Transfer Learning strategies to solve sequence labeling tasks. In this case, the task are: part-of-speech (POS) tagging, text chunking[14], and NER. The prime goal of this research was to reuse knowledge, obtained in a labeled rich source task, to improve the performance of a target task, which is assumed to have less labeled data. The paper provides transfer strategies to apply when the tasks differ in terms of:

- Domain - the source and target data are written in the same language, the sequence labeling task is the same, and the relationship between the label set is not relevant;
- Application - the source and target data are written in the same language, and the sequence labeling task is different;
- Language - the source and target data are written in different languages that share the same alphabet.

The base model, which can be seen in figure 2.8 is composed of three layers: character, word, and CRF. The character-level layer that receives as input a sequence of characters (represented as embeddings) and outputs a character-level feature representation. The world-level layer takes as input word embeddings and the character-level feature representation and outputs a word-level feature representation. In the final layer, this word-level feature representation is sent into a CRF that outputs the label sequence.

The authors propose three transfer model, depicted in figure 2.9. Model 2.9a is used in cross-domain situations when the label sets are the same or can be mapped to each other, for instance, the CoNLL 2003 set of labels can be mapped to the Twitter set of labels from [52]. In this case, all the layers are shared, and the label mapping is performed during the classification process. When label mapping is not possible, or in cases of cross-application, the CRF layer is no longer the shared, meaning that each task trains its CRF layer, which is depicted in model 2.9b. The latter proposed model is shown in figure 2.9c. In this architecture, only the character-level layer is shared, and it is used in cross-lingual situations.

Regarding NER shared tasks and datasets, the approach aforementioned was evaluated against the CoNLL 2002 Dutch and Spanish tasks, the CoNLL 2003 English task and the Twitter corpus [52]. Also, the knowledge of non-NER tasks was reused to enhance

[14]https://www.nltk.org/book/ch07.html (consulted on 31 January 2019).

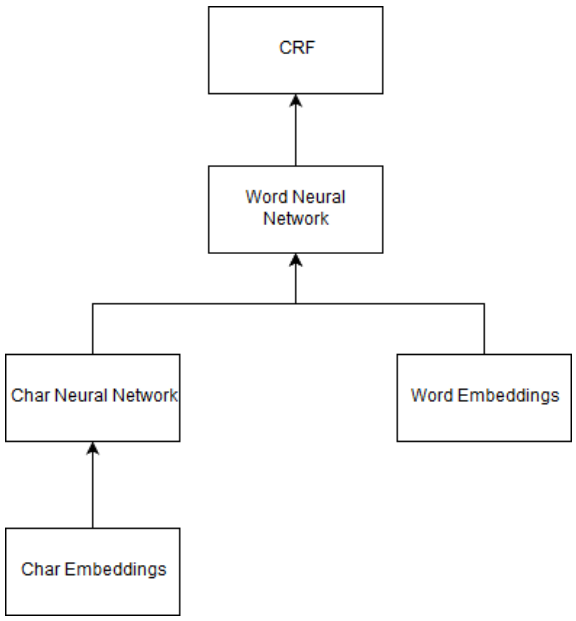


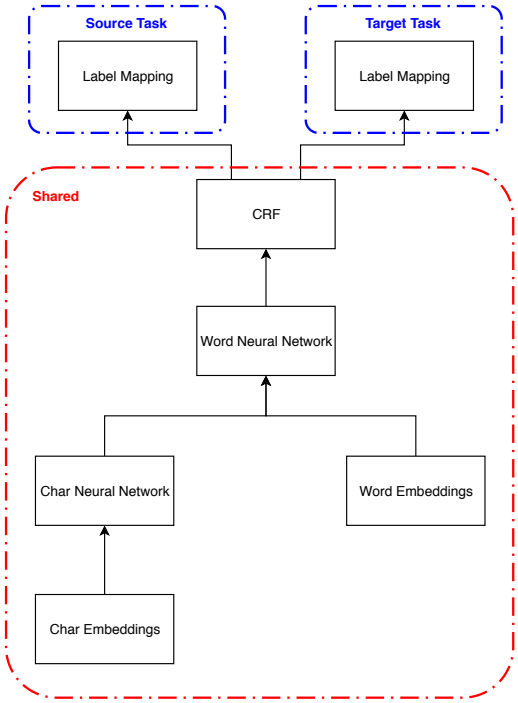
Figure 2.8: The base model, depicted in a three-layered architecture, of a DL systems that uses Transfer Learning strategies for performance boosting (adapted from [64]).

Transfer Type	Average Improvement
Domain	6.94%
Application	4.76%
Dom & App	4.21%
Language	2.28%
Dom & App & Ling	1.13%

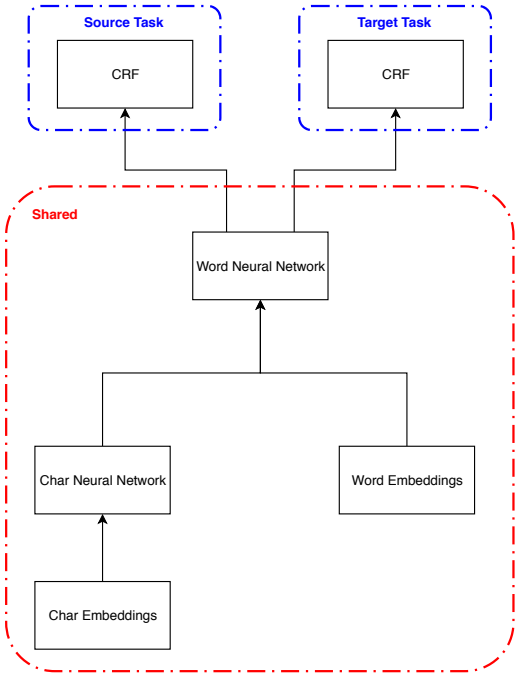
Table 2.5: A table showing for each type of transfer the improvement obtained by using the Transfer Learning approach. These approaches were proposed in [64].

the performance of NER tasks, and vice-versa. Table 2.5 shows for each type of transfer the average improvement that was achieved when using Transfer Learning vs. not using Transfer Learning at all. This table shows that cross-domain situations achieve better performance when using Transfer Learning techniques.

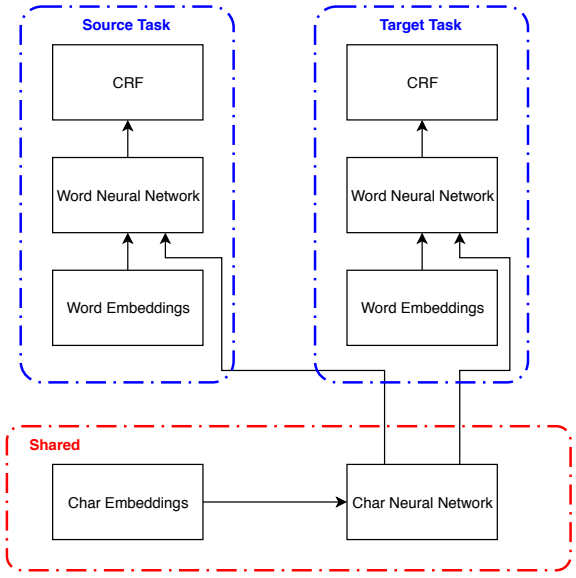
This approach obtained state of the art results in the CoNLL 2003 task by resort to knowledge acquired in the CoNLL 2000 chunking task [54] and the Penn Treebank (PTB) POS tagging task [56], the F_1 score was 91.26%. The result achieved for the CoNLL 2002 Spanish and Dutch tasks were also state of the art, and it was achieved by leveraging knowledge acquired in three languages (English, Spanish and Dutch), for instance, for the Spanish task, the knowledge from the English and Dutch task was reused. The F_1 score for the Spanish task was 85.77% and 85.19% for Dutch.



(a) Transfer model used for cross-domain transfer where label mapping is possible.



(b) Transfer model used for cross-domain transfer where label mapping is not possible and cross-application transfer.



(c) Transfer model user for cross-lingual transfer, if the languages share the same alphabet.

Figure 2.9: Transfer models for situations where the task differ in terms of application, domain, and language (adapted from [64]).

Dataset	# of Categories	# of Tokens	Language
MUC-6	7		English
MET	7		Spanish Chinese Japanese
MUC-7	7		English
MET-2	7		Japanese
IREX	8		Japanese
CoNLL 2002	4	380K	Spanish
CoNLL 2002	4	330K	Dutch
CoNLL 2003	4	310K	German
CoNLL 2003	4	300K	English
OntoNotes v5.0	18	300K	Arabic
OntoNotes v5.0	18	1M	Chinese
OntoNotes v5.0	18	1.7M	English

Table 2.6: Summary of NER datasets. The empty cells are for information that could not be determined or confirmed.

2.4 Discussion

During this chapter, we have described the concepts of crowdsourcing, NER, and Transfer Learning. Given our current knowledge, this thesis is the first to combine these three areas.

Regarding crowdsourcing, we have presented examples that date back to 1714, we have also shown how iStockPhoto has changed the stock photography marketplace by using peer-vetted crowdsourcing. However, we give more relevance to the distributed intelligence tasking throughout this thesis. Under this crowdsourcing type, we presented a typical crowdsourcing pipeline consisting of a generation and validation step, which is the type of pipeline that we aim to optimize. We finished the crowdsourcing section with a description of two methods used to ensure the quality of the results produced by the contributors; as quality control is one of the most challenging areas in crowdsourcing.

We presented the task of NER and provided a historical overview of the creation and first steps of this task. We proceed to describe the annotation schemes, used by both humans and machines, to execute the NER task; the data used in this thesis is annotated according to the IOB2 scheme. We have also defined metrics to measure the performance of ML systems. These metrics can be calculated in a strict or relaxed manner, for instance, a named entity can only be considered correct if the boundaries and the label are correct, or named entities can be considered correct in two-axis evaluation, meaning that an entity can still be considered (partially) correct if either the category or the boundaries are correct; this evaluation method penalizes more predictions which fail both the category and the boundary, and this is the kind of errors that we intend to minimize.

Relevant datasets for NER have also been briefly described, since MUC-6 a wide range

of conferences has been defining new tasks and datasets. Table 2.6 shows a resume of the datasets that have been mentioned. It is possible to see the number of categories, the number of tokens, and the language of the corpus. The corpus from the MUC-6, the MUC-7, and the IREX have well-defined domains, while the remainder has a range of domains.

Approaches that tackle NER have changed throughout time. The first developed models made extensive use of hand-engineered features, which requires a lot of skill and time to generate. More recent approaches use DL techniques to generate distributed representations for inputs, which are then sent into a context encoder. This context encoder is responsible for analyzing these representations with the objective of understanding and encoding, into a new representation, the underlying syntactic and semantic context. This new context-dependent representation is sent into a tag decoder layer, which produces a classification for each token.

The distributed representations for inputs are obtained via the usage of neural networks through unsupervised learning algorithms; however, there are representations publicly available that can be used directly to train models. The context encoder is, normally a CNN, a RNN, which is a LSTM or a GRU. Finally, the tag decoder, as it has been shown, is typically a CRF, although other models can be used, like a RNN or a softmax layer. We aim to implement these systems as they obtained state of the art results in sequential labeling tasks.

Regarding Transfer Learning, we have shown how this concept relates to human psychology and how important it is in the process of learning. The relation between domains and tasks enabled breaking down Transfer Learning into three different settings: inductive, transductive, and unsupervised. Furthermore, it is possible to transfer four types of knowledge: examples of the source domain's data, the representation of the features, shared parameters or prior distributions and relationships between domains.

Transfer Learning has been used in CV, speech, and NLP to surpass previous approaches that did not use Transfer Learning while requiring fewer data. Based on the results achieved in the previous section, we focus on domain adaptation.

Chapter 3

Model Adaptation

During all those years of experimentation and research, I never once made a discovery. All my work was deductive, and the results I achieved were those of invention, pure and simple.

– Thomas Edison

Crowdsourcing appeared as a scalable approach to annotate high volumes of data. The recognition of entities in a text can be decomposed into micro-tasks, which can be solved manually by the crowd or automatically by ML algorithms, or by combining both. A straightforward approach would be to train a model in a dataset annotated for NER and then use it to predict the entities in non-annotated data – the pre-trained model approach. The crowd then validates the generated predictions. However, these predictions may have a substantial error rate because the model is not adapted to this domain. Furthermore, this approach has limited usability as it requires a relationship between the datasets. In this thesis, we propose an approach that leverages in-domain data (IDD) – the designation given to the non-annotated dataset – to adapt a model to the dataset that needs to be labeled.

In this approach, we train a model with part of the dataset. While the model generates predictions for the remainder of the data, these predictions have a lower error rate when compared to the pre-trained model approach because the training data is related to the data in which we want to generate the predictions. Another advantage of this approach is the use case scenarios in which it can be applied, which are, theoretically, unbounded. We also test the usage of out-domain data (ODD), which is data that has already been annotated and can be leveraged when the amount of IDD are limited. The usage of ODD allows for achieving better results while requiring fewer amounts of IDD.

We investigate the answer the first research question and study the relation between the amount of training data and the model's performance, with and without the addition

Data	# Sentences	# Tokens	# Entities
Training	14,041	203,621	23,499
Development	3,250	51,362	5,942
Test	3,453	46,435	5,648

Table 3.1: Table with the number of sentences, tokens, and entities for the three data files that compose the CoNLL 2003 English corpus. The category set consisted of four categories: organization, person, location, and miscellaneous.

of ODD. However, we first need to implement the state of the art regarding the task of NER, more concretely, a neural network architecture that follows the three-layered architecture defined in section 2.3.2. With this in mind, the remainder of this chapter is organized as follows:

- Section 3.1 describes the neural network architecture and the word embeddings to be used in the remainder of the thesis, as well as a study concerning the dimension of the word embeddings;
- Section 3.2 describes the first experiment. In this section, we define the baseline values of performance and present the results of using IDD and IDD + ODD;
- Section 3.3 discusses the results of the NER model implementation and the domain adaptation experiments.

3.1 Implementation of State of the Art NER Model

In section 2.2, we have described the state of the art in NER. To conduct our experiments, we need to implement a neural network architecture that achieved state of the art results in the task of NER. For the implementation of the model, we selected the Keras [12] library with the Tensorflow [1] backend. This library is particularly interesting because it offers a high-level API to develop neural networks, which facilitates the process of prototyping. We also chose to use the CoNLL 2003 dataset in our experiment, and thus chosen a neural network architecture that obtained state of the art results in this dataset. We proceed to describe the dataset and specify the associated task. Table 3.1 presents the number of sentences, tokens, and entities for the dataset aforementioned.

In table 3.1, we present the number of sentences, tokens, and entities of the CoNLL 2003 dataset, which is divided into three distinct datasets: training, development, and test. The category set of this dataset consists of four categories: organization, person, location, and miscellaneous. The training data is the data used to train the model. The development data is used to fine-tune the hyper-parameters of the model. Lastly, the test data is used to retrieve the model's performance.

Given the dataset choice, we choose a neural network architecture that has been tested in this same data as this allows the realization of a sanity check to ensure the

Probabilities and Ratios	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Table 3.2: The probabilities of the words *ice* and *steam* occurring when the words *solid*, *gas*, *water*, and *fashion* occur. The ratio of probabilities is also presented to show words that represent specific properties of the word *ice* or the word *steam*. This table was retrieved from [50].

correct implementation of the neural network. The selected neural network architecture obtained an F_1 score of 0.91. In the upcoming section, we describe the neural network architecture and present the results of the sanity check.

3.1.1 Neural Network Architecture

This section resumes the architecture of the neural network proposed by Ma and Hovy [39].

The neural network architecture follows the three-layered architecture described in 2.3.2 and is organized as follows:

- the character-level context encoder is a CNN which takes as input the character embeddings and outputs the character representation which is then inputted into the word-level context encoder;
- the word-level context encoder takes as input the concatenation of the character-level representation and the word embeddings and outputs the word-level representation. The word-level context encoder is a BiLSTM;
- Finally, the word-level representations are inputted into a tag decoder, which is a CRF, that is responsible for predicting the category of each token.

This neural network diverges from the one described in the paper aforementioned in two aspects: the optimization algorithm and an extra case layer. In the paper, the optimization algorithm was the stochastic gradient descent (SGD). However, we have observed through experimentation that the Nadam optimization algorithm [18, 61] is way faster at reaching an F_1 score of 0.90 than the SGD is at reaching the 0.91 F_1 score. Due to this observation, we favored speed over 0.01 performance and selected the Nadam optimization algorithm. The case input layer is used to retain and leverage the capitalization features, which are lost during the word embeddings lookup [10]. This layer is concatenated to the character-level representation and the word embeddings, and it is inputted into the word-level context encoder.

For word embeddings we selected the pre-trained word embeddings from Global Vectors (GloVe) [50] model. The GloVe model is a log-bilinear model that retrieves meaning by recurring to the ratios of word-word co-occurrence probabilities.

Dimension	F_1 score
50	0.87
100	**0.90**
200	0.89
300	0.89

Table 3.3: Table indicating the obtained F_1 score for each different dimension of word embeddings.

In table 3.2, it is possible to see the ratios for the words *ice* and *steam*. From the rations, one can perceive that these words co-occur more frequently with the word *water* than they do with the word *fashion* because both ice and steam are water in the different states of matter. However, none of these words are applicable when referring to fashion. It is also possible to see that the word *ice* co-occurs more frequently with the word *solid* than it does with the word *gas*, which is expected since ice is water in a solid state. The word *steam* shows the inverse relation and co-occurs more with the word *gas*.

By considering the ratio between the probabilities of co-occurrence, it is possible to notice which words represent specific properties of *ice* and *steam*. Consider the word *water*, which has high co-occurrence probabilities with both *ice* and *steam*, such indicates that the word *water* is not specific to neither of them. Rather it is common to both of them. This fact explains why the ratio between probabilities is close to one. However, the word *solid* is specific to *ice* but not to *steam*, that is why the ratio is much larger than one. The inverse applies to *gas* in relation to *steam*.

3.1.2 Sanity Check and Word Embeddings Dimension Comparison

In this thesis, we utilize the pre-trained word vectors from GloVe that were trained on 6 billion tokens from Wikipedia and Gigaword 5[1]. These word embeddings are made accessible in four dimensions: 50, 100, 200, and 300.

The introduction of word embeddings is core to NER as it allows to discard hand-engineered features, which requires a significant human effort to generate [37]. Therefore, to select the dimension of the word embeddings, we analyze the model's performance.

The F_1 scores achieved by the model while trained and tested in the CoNLL 2003, for the four different dimensions are shown in table 3.3. The best F_1 score is obtained when the dimension of the word embeddings is equal to 100. These results match the ones reported by Ma and Hovy [39]. These scores were obtained when training the model over 50 epochs, which are the epochs that we consider for the remainder of this thesis.

While our model did not achieve 0.91 F_1 score, we consider this experiment to be a success. Our neural network achieved a 0.90 F_1 score when trained over 13 minutes, while the neural network in the paper was trained over 12 hours. In the next section, we show the relationship between the amount of training data and the model's performance.

[1] https://catalog.ldc.upenn.edu/LDC2011T07 (consulted on 29 July 2019).

Metrics	A	B
Categories	4	1
Sentences	20,744	22,500
Tokens	301,418	782,276
Characters	1,314,351	3,478,840
Entities	35,089	14,098
Average sentence length	14.53	34.77
Average token length	4.36	4.45
Entity coverage	11.64%	1.80%

Table 3.4: General metrics regarding the CoNLL 2003 dataset (A) and the financial reports corpus (B).

3.2 In-domain Data and Out-domain Data

In this experiment, the goal is to understand the relation between the amount of IDD and ODD and the model's performance. To perform this experiment, we need to introduce a second dataset, to test the usage of ODD. This dataset is described in the next section.

3.2.1 Data

In addition to the CoNLL 2003 dataset, we consider a dataset of financial reports written in English. This kind of text is typically composed of loose phrases that describe, in a formal way, the financial activities of companies, persons, and other entities. Furthermore, the NER task performed in this financial reports datasets was the identification of companies. For the remainder of this chapter, we refer to the CoNLL 2003 dataset as Dataset A and to the financial reports corpus as Dataset B.

Table 3.4 presents a comparison between the two datasets. In this table, it is possible to see that Dataset A has four categories, which are person, location, organization, and miscellaneous, while Dataset B has one category which is company. Furthermore, Dataset A is smaller than Dataset B, in terms of size, and Dataset B has longer sentences than Dataset A. However, both have the same average token size. Dataset A has a more substantial entity coverage – the ratio of entities per number of tokens – than Dataset B. Furthermore, the division of the two datasets is different. While Dataset A is divided into three sets, Dataset B is divided into two sets (the train and test set).

Figure 3.1 illustrates the two splits. Note that we do not use a development set for Dataset B because we assume that the optimal hyper-parameters are the same for both datasets. It is worth noting that any neural network trained in Dataset A only has access to 70% of the data for training and 85% in the case of Dataset B.

In the next section, we define the baseline values for the model's performance by training a model in Dataset A and use it to annotate Dataset B, and vice-versa.

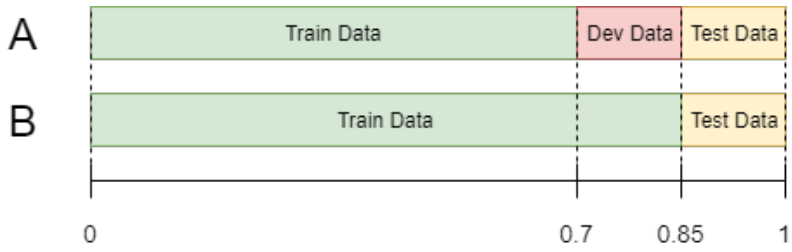


Figure 3.1: The data splits for Dataset A and Dataset B. Dataset A has a development set to enable the fine tune of the hyper-parameters which are then reused for the remainder of the experiment.

Dataset	Precision	Recall	F_1 score	MUC score	Support
A	0.89	0.90	0.90	0.91	5648
B	0.82	0.86	0.84	0.87	2126

Table 3.5: The values of precision, recall, F_1 score, MUC score, and support for both datasets; these values were obtained while training the model with the same hyper-parameters and dimension of word embeddings and over 50 epochs.

3.2.2 Baseline

Before establishing the baseline values, we present the performance results when training and testing the model in the same dataset.

In table 3.5, we present the values of performance for both datasets when using the same hyper-parameters and same dimension of word embeddings. The values of performance of Model A (the model trained on Dataset A) are better than Model B's (the model trained on Dataset B). Also, note that the difference between the MUC scores is not the same as the F_1 scores, which means that Model B is getting entities partially correct. This error sensitivity is essential as it allowed Model B to achieve 0.03 more performance, and thus makes this metric a better indicator of the model's performance in our context.

We proceed to define baseline performance values for Dataset B by recurring to a model that has been trained in Dataset A. Note that, this model can generate predictions in Dataset B because there is a category mapping between both datasets. The category mapping is defined as a function that receives the category of the source dataset and returns a category of the target dataset. This relationship is an example of a constraint when using the pre-trained model approach. Without this relationship, this approach can not be applied as the model would not be solving the intended task.

```
1 def category_mapping(source_category):
2     if source_category == 'ORG':
3         return 'COMPANY'
4
5     return 'O'
```

Model	Dataset	Precision	Recall	F_1 score	MUC score	Support
A	B	0.34	0.78	0.47	0.51	2126
B	A	0.15	0.11	0.13	0.15	1661

Table 3.6: Values of precision, recall, F_1 score, MUC score, and support when using Model A to generate predictions in Dataset B, and vice-versa.

Listing 3.1: Category mapping function that receives a category from Dataset A and returns the correspondent category from Dataset B.

The category mapping in 3.1 shows that the organization category is mappable to the category company, while the remainder – person, location, and miscellaneous – are mapped to the category "O", which is the absence of category. We use the inverse of this function to generate predictions in Dataset A with a model trained on Dataset B.

The results of training a model in one dataset and generate predictions in the other are depicted in table 3.6. These values are henceforth referred to as baseline values. Note that that Model A – the model trained on Dataset A – performs better at generating predictions in Dataset B than Model B – the model trained on Dataset B – performs when generating predictions in Dataset A. The most notorious difference is in terms of recall, while Model A is capable of identifying almost 1,500 companies, Model B is only capable of identifying around 180 organizations.

However, this approach presents two problems: has limited usability, and the model's predictions have a substantial error rate. This approach is limited because it enforces the existence of a category mapping between the two datasets, which is not always possible. Furthermore, the models' performance experienced a significant reduction in this experiment. For instance, Model A achieved a MUC score of 0.91, when training and testing in Dataset A. However, when testing against Dataset B, it achieved 0.47. This decrease in performance shows that the models are not well adapted to the dataset with ODD.

In the next section, we explore a different approach based on IDD and analyze how does the addition of ODD contributes to boosting the model's performance.

3.2.3 In-domain Data Approach

In this section, we describe the IDD approach and present the performance values obtained by the models when training under this approach's learning setting. The IDD approach consists in using only a portion of the dataset to train a model and then use such model to generate predictions for the remainder of the dataset.

Figure 3.2 shows the split of the train data, which is divided into two sets: a train set, and the unseen data. The train data mentioned in the figure is the same in figure 3.1, which means that only a portion of the training data is used for training (green rectangle) while the remainder is used for generation of predictions (gray rectangle). In this approach, we vary the size of the train set to train the neural network, or in other

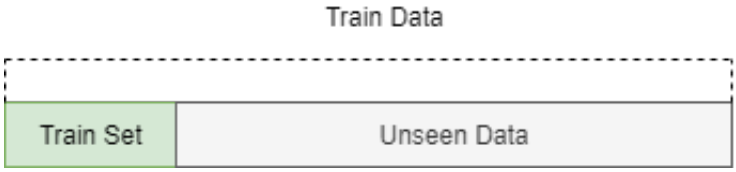


Figure 3.2: The split of the train data in the IDD approach. The train set, in green, is the actual data used for training and the unseen data, in gray, is the portion of the data in which to predictions are generated.

Amount of Training Data (%)	Sentences	Tokens	Entities	Mean Sentence Length	Entity Coverage (%)
1	140	2,444	288	17.46	11.78
3	421	6,393	780	15.19	12.20
5	702	8,987	1,349	12.80	15.01
10	1,404	19,596	2,610	13.96	13.32
20	2,808	39,371	5,134	14.02	13.04
30	4,212	57,435	7,217	13.64	12.56
40	5,616	73,332	9,498	13.06	12.95
50	7,020	91,972	11,668	13.10	12.69
60	8,424	113,302	13,826	13.45	12.20
70	9,828	135,668	16,230	13.80	11.96
80	11,232	156,356	18,891	13.92	12.08
90	12,636	178,816	21,399	14.15	11.97
100	14,041	203,621	23,499	14.50	11.54

Table 3.7: The number of sentences, tokens, entities, the mean sentence length (number of tokens per number of sentences) and the entity coverage (number of entities per number of tokens) for each amount of training data. These amounts of training data are associated with the number of sentences of Dataset A.

words, vary the amount of training data; this allows the understanding of the tip-over point from which adding more data does not contribute significantly to improve the model's performance.

To find this point, we plot the model's learning curve, which is the model's performance as a function of the amount of training data. For this experiment, the amount of training data is partitioned in terms of the percentage of sentences. We study the model's performance of the values of 1%, 3%, 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 100%.

In tables 3.7 and 3.8 are the counts and metrics for each one of the amounts of training data for both datasets. The values these tables were obtained by shuffling the entire dataset and then proceeded to retrieve the amounts of training data aforementioned. These tables serve only to map the relative percentual values to absolute values and retrieve basic metrics to understand if the split maintains the original dataset's characteristics.

Amount of Training Data (%)	Sentences	Tokens	Entities	Mean Sentence Length	Entity Coverage (%)
1	190	6,801	102	35.79	1.50
3	571	20,102	322	35.20	1.60
5	952	33,708	522	35.41	1.55
10	1,905	66,088	1,068	34.69	1.62
20	3,810	134,248	2,226	35.24	1.66
30	5,715	201,626	3,424	35.28	1.70
40	7,620	266,884	4,577	35.02	1.71
50	9,525	334,008	5,867	35.07	1.76
60	11,430	398,912	6,991	34.90	1.75
70	13,335	464,798	8,248	34.86	1.77
80	15,240	530,725	9,491	34.82	1.79
90	17,145	596,399	10,785	34.79	1.81
100	19,051	663,880	11,972	34.85	1.80

Table 3.8: The number of sentences, tokens, entities, the mean sentence length (number of tokens per number of sentences) and the entity coverage (number of entities per number of tokens) for each amount of training data. These amounts of training data are associated with the number of sentences of Dataset B.

We train the neural network on these two datasets for the different amounts of training data. The models' learning curves are presented in figure 3.3. These figure shows the MUC score achieved when training the model in Dataset A (blue line) and in Dataset B (orange line). From the results, we can conclude that even for 1% of training data, the models trained with IDD achieve scores which are better than the baseline.

When comparing the learning curves, it is possible to observe that the slope of Model A's curve is more accentuated than Model B's. Also, both curve slopes are more accentuated up until 5-10%, and then they start to decline until stabilization is reached around 20-30%.

The stabilization should be reached as soon as possible; such means that the model requires less training data to achieve a performance that is not the best, but it is a satisfactory performance nevertheless. With that in mind, we try to boost the model's performance by recurring to ODD. The addition of ODD generates the necessity of redefining the learning setting because, essentially, we need to train the model in two distinct datasets. We accomplish this by adopting the Transfer Learning strategies that have been described in section 2.3.2. In the next section, we describe the new learning setting and present the results of using ODD for performance boosting.

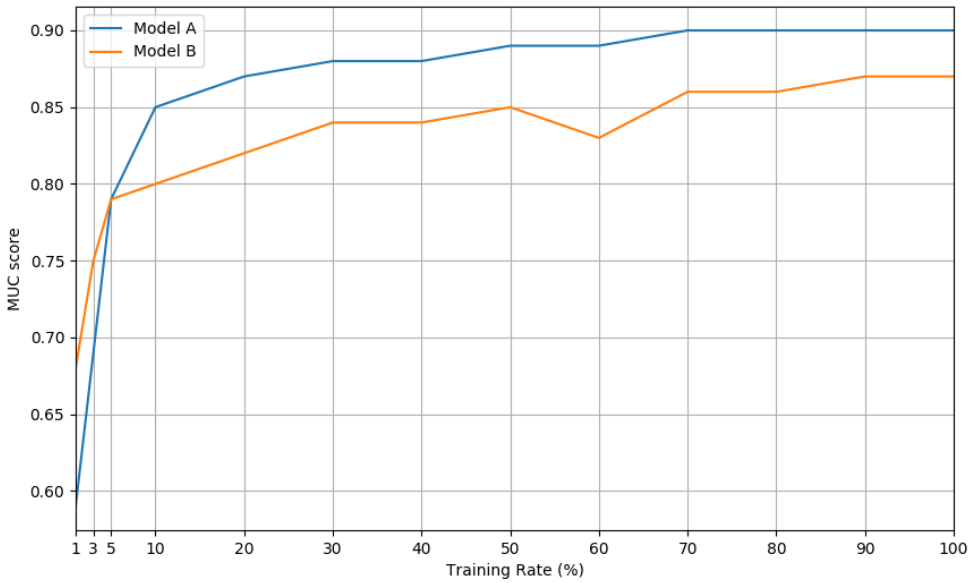


Figure 3.3: A line plot displaying the MUC score for each amounts of training data for both datasets.

3.2.4 ODD for Performance Boosting

The models trained under the Transfer Learning setting require fewer amounts of data to achieve better performances. Given that we want to minimize as much as possible the amount of training data, the experimentation of Transfer Learning strategies appears as a logical investigation path to follow.

In this section, we follow the learning setting described in section 2.3.2. In this scenario, we possess two datasets written in the same language, but with different domains – the type and subject of the two datasets are distinct. We train two models, one for each dataset, that share the weights of the character and word-level neural networks as well as the character and the word embeddings (see figure 2.9b). Therefore, both models are trained in the two datasets, which implies that the models can attain consistent performance in both datasets. However, the goal is not to train two models that perform consistently in two datasets; rather we aim to train the target model, which is the model that solves the task associated with the target dataset. The target dataset is typically the dataset that has less entity coverage – such as Dataset B. This goal is what distinguishes this learning setting from the multi-task learning setting [48].

In figures 3.4 and 3.5, it is possible to compare the values of the MUC score when using only IDD and IDD + ODD for both datasets. As can be seen, there is a boost in terms of performance and that it is more notorious for low amounts of training data. However, when the amount of data from in-domain starts to become relevant, the usage of data from an outer domain starts to be unnecessary as the model's performance does not

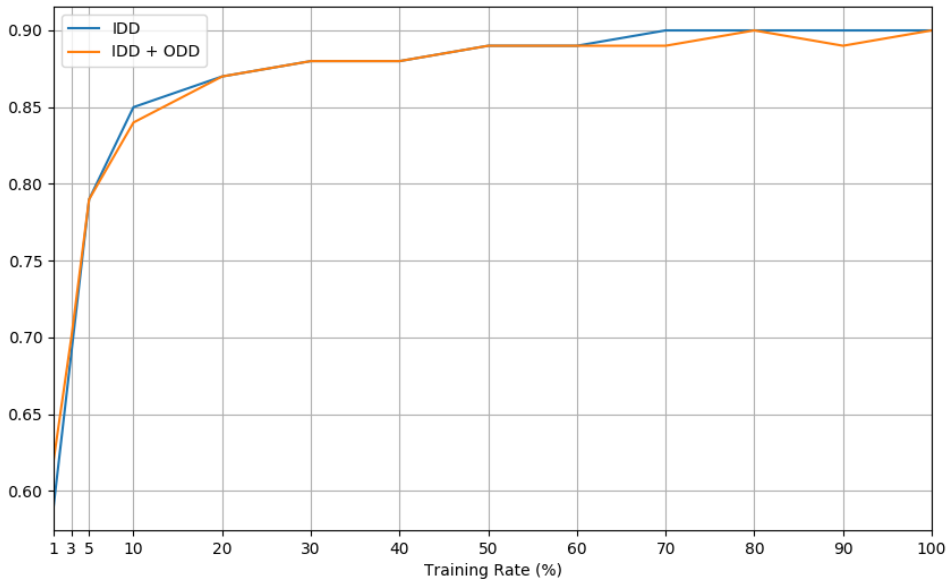


Figure 3.4: The values of the MUC score for Model A when trained with in-domain data (blue line) and with in-domain data with the addition of out-domain data (orange line).

Amount of training data (%)	IDD	IDD + ODD
1	0.14	0.22
3	0.22	0.26
5	0.28	0.29
10	0.29	0.32
20	0.31	0.33
30	0.34	0.34
40	0.34	0.34
50	0.36	0.35
60	0.33	0.34
70	0.37	0.36
80	0.36	0.35
90	0.37	0.35
100	0.37	0.36

Table 3.9: The relative of gains in performance, measure via MUC score, for each amount of training data of using in-domain data and in-domain data + out-domain data when compared the pre-trained model approach.

increase. For amounts of training data, such as 50%, the usage of ODD even deteriorates the performance of the model.

To better understand the boost induced by ODD, we present the relative gains of both approaches against the baseline values in table 3.9. From these results, one can see that the usage of IDD does contribute significantly to improve the model's performance. However, the boost produced by using ODD is much smaller when compared to

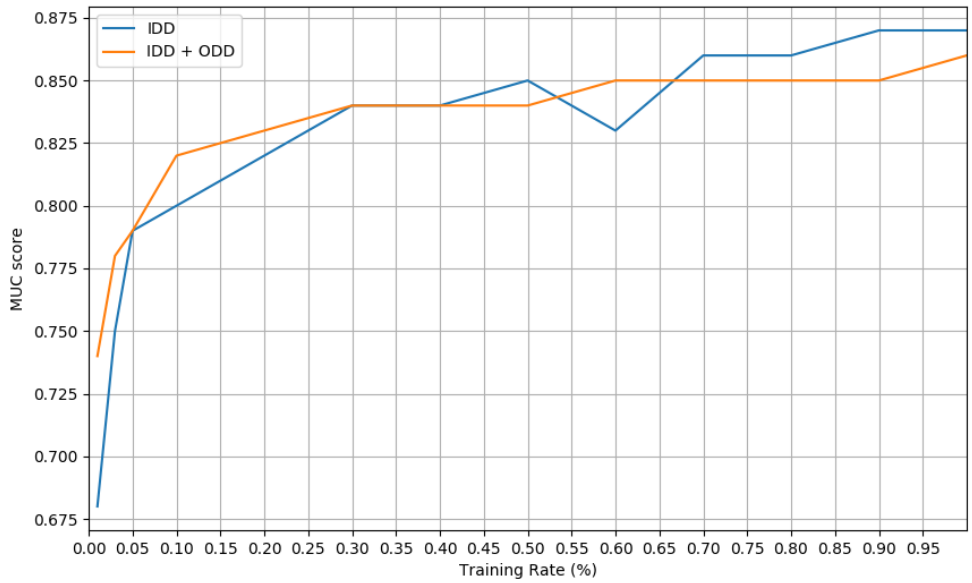


Figure 3.5: The values of the MUC score for Model B when trained with in-domain data (blue line) and with in-domain data with the addition of out-domain data (orange line).

the improvement attained by the IDD approach with respect to the pre-trained model approach.

Therefore, We must assess how the performance gains achieved in this experiment translate into time savings when applied to a crowdsourcing pipeline of NER. This assessment is performed in the second experiment, which is described in the next chapter.

3.3 Discussion

In this chapter, we have implemented the state of the art concerning NER to enable the elaboration of the experiment that answers to how does the amount of training data influence the model's performance, which is the first research question.

Regarding the state of the art, we have conducted an experiment to verify the correct implementation of the neural network architecture. Our model presented an F_1 score of 0.90 which is aligned with our expectations and not showing a significant difference from the performance reported in by Ma and Hovy [39]. Furthermore, such a result was achieved in 13 minutes, which is 55 times less than the training time reported in the paper.

Once the architecture was implemented, we defined the baseline values of the two datasets by training a neural network in one dataset and testing it against the other; such values are presented in table 3.6. However, when comparing these values against the performance values achieved when the neural networks are trained and tested in the same

dataset, one can perceive that the pre-trained models have suboptimal performance. Furthermore, this approach is only applicable because of the existent of a category mapping between the datasets, which is a tremendous limitation.

Given the problems associated with this pre-trained model approach, we presented a new approach that leveraged IDD to make a model adaptation. The results show that using even just 1% of IDD for training wields better results than using the pre-trained model approach. Another interesting fact is that the model's performance tends to stabilize when the amount of training data is between 20%-30%, meaning that from this percentage of training data, the model's performance does not increase significantly.

The results of using ODD to train the model allows to conclude the same that we conclude concerning the IDD approach: the performance is better than the pre-trained model approach, and the model's performance tends to stabilize after a specific amount of training data. Furthermore, this approach achieves the same performances while requiring less training data than the IDD approach. For instance, when training with IDD, Model B achieves 0.82 MUC score only when the amount of training data is equal to 20%. However, the addition of ODD allows Model B to achieve such performance with only 10%.

Also, the dataset with the lowest entity coverage benefits more from jointly learning in both datasets than the dataset with the highest entity coverage. However, the improvements obtained by the usage of ODD seem to be irrelevant when compared to the improvement obtained by using solemnly IDD. Due to this, we must assess how the performance gains achieved in this experiment translate into time savings when applied to a crowdsourcing pipeline of NER. This assessment is our second experiment, and it is described thoroughly in the next chapter.

In sum, two major conclusions can be deduced from this experiment:

- A NER state of the art model can be trained on 20-30% of the dataset and achieve performances close to its best performance. The performance of a model when trained with 20%-30% of training data achieves a MUC score that is 0.02 to 0.05 less than the best-reported score;

- The usage of ODD boosts performance, mainly when low amounts of training data are available. The most notorious performance boost is reported when the model is trained with 1% of training data.

Chapter 4

Assessment of Crowd Effort Savings

Productivity is never an accident. It is always the result of a commitment to excellence, intelligent planning, and focused effort.

– Paul J. Meyer

In the previous chapter, we answered the first research question and understood how does the amount of training data relate to the model's performance. However, having better performance does not mean that this is the recommended setting.

Now, these performances have time costs associated with them. Following the proposed architecture, there are two steps:

- Generation;
- Validation.

In the generation step, the cost relates to the amount of IDD required to train a model, while the validation cost relates to the predictions generated by the model. Therefore, we need to understand how much time does it take to generate a certain amount of training data and how much does it take to validate the predictions generated by a model. The sum of these two times is the expected crowd effort necessary to complete a NER job.

With this in mind, we conduct a study to determine the effort that is required to perform a generation task of NER in a crowdsourcing platform. We assume that to complete a single task a contributor needs to read the entire text and annotate the entities within the text, which implies that we need to determine two values: the reading time and the annotation time. Defining these dimensions allows for the estimation of the generation and validation efforts for any English NER job.

These generation and validation efforts are the measures that allow the assessment of whether or not the approaches proposed by this thesis achieve savings regarding crowd efforts; this is the answer to our second research question.

The remainder of this chapter is organized as follows:

- Section 4.1 describes the statistical analysis performed on a dataset of executions of crowdsourced work. The goal of this analysis is to determine the reading and annotation time;
- Section 4.2 describes the results of this experiment. In this section, we define the total generation effort and determine the efforts associated with the proposed approaches to measure the gains concerning the traditional crowdsourcing approach;
- Section 4.3 has the summary of this chapter and the discussion of the results regarding the usage of IDD and ODD.

4.1 Statistical Analysis of Time on Task

In this section, we describe the statistical analysis over the time that contributors take to perform one job of NER in a crowdsourcing platform. The goal is to estimate the generation and validation costs of our approach. To conduct this experiment, we have retrieved a dataset that each row represents a HITEX. In the next section, we describe our data cleaning process, starting by describing the entire dataset and explaining the reasoning behind each data removal.

4.1.1 Data

To assess the time on task, we retrieved a dataset that is composed of 2,560,520 rows and six columns; this dataset includes all the jobs of NER available in DefinedCrowd's platform. The columns in the dataset are the following:

- *hitExecutionId* - The unique identifier of the HITEX;
- *hitId* - The unique identifier of the HIT;
- *jobId* - The unique identifier of the job;
- *totalTaskTime* - The target variable in this study and the amount of time, in minutes, that the contributors spend on the task (defined at HITEX level);
- *prompt* - The text presented to a contributor to perform the task (defined at HIT level);
- *result* - The answer submitted by the contributor (set of entities, each defined in terms of location concerning the prompt, and category).

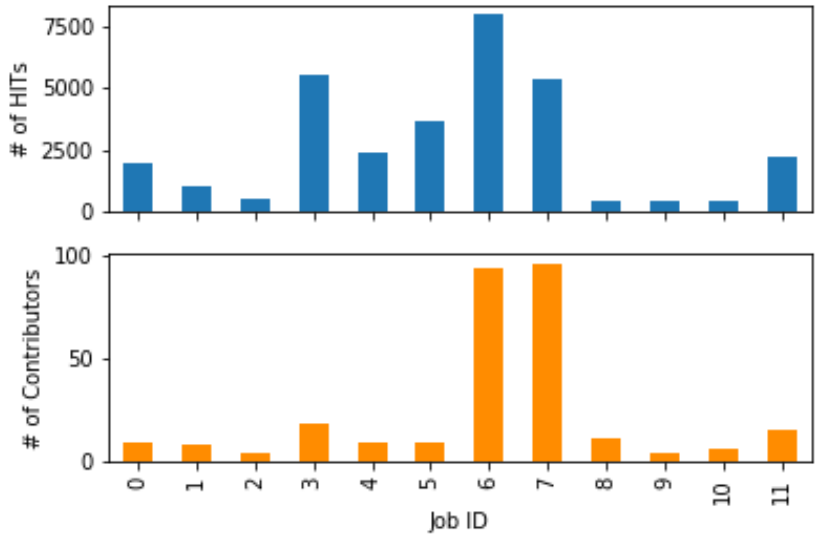


Figure 4.1: The bar plots concerning the number of HITs (top) and the number of contributors (bottom) for each crowdsourcing job.

This dataset is a flat schema [44] version of the actual database. In this scenario, the atomic unit is the HITEX, and the information is normalized up to the job level; this dataset has 2,560,520 HITEXs, 936,063 HITs, and 51 jobs.

We start by removing all the rows in which the *totalTaskTime* is not defined as this means that such metric was not retrieved for that specific execution, and thus it can not be used in our analysis. We removed 1,084,598 HITEXs. After this cleaning, the dataset has 1,475,922 HITEXs, 516,849 HITs, and 32 jobs.

Finally, we remove two outlier jobs from the dataset. The prompts of the HITs associated with these two jobs were documents with an average of 491 tokens, which is 21 times larger than the average size of the prompts of the remaining jobs. Furthermore, these jobs were performed in special conditions (such as being annotated by experts), and thus we considered that they were not an excellent indicator of how much effort the crowd necessitates to execute the task of NER.

The final dataset is composed of 12 jobs, 32,069 HITs, and 99,280 HITEXs. Each job has, on average, 2672.42 ± 2479.27 HITs and got the participation of 23.58 ± 33.61 contributors. Each HIT was executed by 3.10 ± 0.54 contributors. Each contributor executed around 451.27 ± 999.10 HITs. A total of 220 contributors performed these tasks.

The total number of HITs and contributors per job is presented in figure 4.1. As can be seen, job 3, job 6, and job 7 stand out when it comes to the number of HITs. However, the jobs are heterogeneous when it comes to this dimension. Regarding the number of contributors, both job 6 and job 7 have close to 100 contributors, which is comprehensible do to the high number of HIT. However, this rule does not apply for the remainder of the

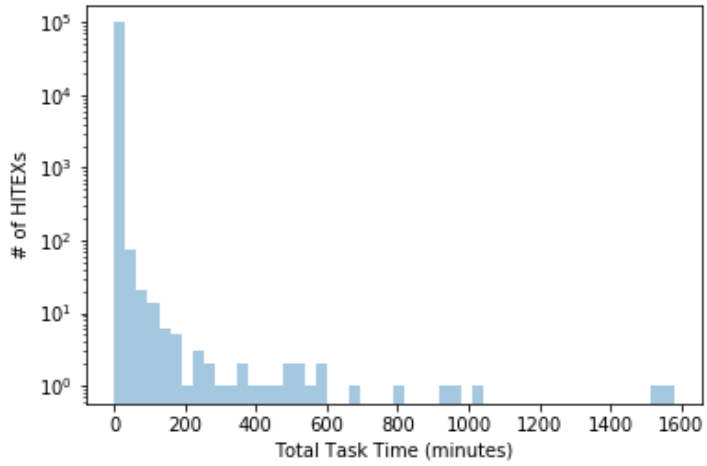


Figure 4.2: The distribution of the target variable, *totalTaskTime*, with the y axis in logarithmic scale.

jobs as the number of contributors is more or less constant despite the number of HIT.

4.1.2 Calculation of the Token Processing Speed

The task of NER can be divided into two parts: the reading of the prompt and the classification of entities. Therefore, to estimate the duration of any task, one requires to understand how much time a contributor takes to read a prompt of size n and how much time it takes to classify m entities. Therefore, we should define a function that receives both n and m and estimates the necessary effort to read and annotate the prompt (the time to complete the task). Therefore, we start by checking the distribution of the *totalTaskTime*, which can be seen in figure 4.2.

To do that, we start by checking the distribution of the *totalTaskTime*, which can be seen in figure 4.2. The y axis is in logarithmic scale to better visualize the distribution plot. Note that there are a substantial number of outliers in the data. The task of NER is straightforward, whose time executions should not take longer than 4 or 5 minutes, considering the size of the jobs. However, the x axis depicts times ranging from 0 to 1600 minutes, which is more than one day. These outliers can be explained by contributors leaving the task window open while away from the computer.

However, the total task time presents a broad range of values, even when not considering extreme values. This fact creates the necessity of normalizing the time values and create a new normalized dimension. Given the dimensions in our dataset, we select the prompt size in tokens to do this normalization as it is sensible to assume that the prompt size is highly correlated with the total task time.

Having this normalized dimension enables the removal of outliers independently of the prompt size and the total task time, which is ideal as removing outliers based on

total task time might remove tasks have large prompts and consequently are more time-consuming tasks. We refer to this dimension as token processing speed, and it can be computed by using equation 4.1, where d is the prompt size in tokens, t is total task time in minutes, and thus the token processing speed is given in tokens per minute (tpm).

$$v = \frac{d}{t} \tag{4.1}$$

Computing the token processing speed, also allow us to apply plausibility thresholds over the actual velocity humans can read. The investigation around this topic started around the 1960s with Fry [21] whose work defined that slow readers read at about 150 words per minute (wpm), while fast readers may achieve reading speeds of around 350 wpm. De Leeuw and De Leeuw [16] showed that the average initial reading speed of the general population is around 250 wpm.

Due to these studies, there was a growth in the development of reading speed courses, whose objective was training individuals to read faster through paper-based techniques and technological aids [5]. One such course enabled students to optimize their reading speed from 200 wpm to 314 wpm. Furthermore, a portion of the students was able to achieve reading speeds of 600 wpm [25].

Based on these studies and considering that the total task time is composed of two distinct times:

- Reading time - the time to read the prompt;
- Annotation time - the time to annotate all the entities within the prompt.

We expect that contributors have lower processing speed because the total task time accounts for the overhead of annotating entities. However, it is worth noting that more experienced contributors may be able to perform tasks at 600 wpm. Therefore, to be conservative, we define the plausibility thresholds of 10 wpm for the lower bound and 600 wpm for the upper bound, removing every HITEX whose processing speed does not fall within this range. Note that equation 4.1 computes the processing speed in tpm; however, we need it to be in wpm. Such conversion is performed by counting the prompt size in words instead of tokens.

As an example, consider a contributor performs the task of NER in the sentence "Alexa! Open Spotify, please, and play Guns and Roses.". This sentence has nine words and thirteen tokens (nine words plus four punctuation marks). Assuming that a contributor takes 1 second to read this sentence and it does not perform any annotation. Under these conditions, the total task time is equal to the reading time, and thus, the values of processing speeds are 540 wpm and 780 tpm. By considering the processing speed in wpm, we do not discard this execution, whereas if we consider in tpm, we do.

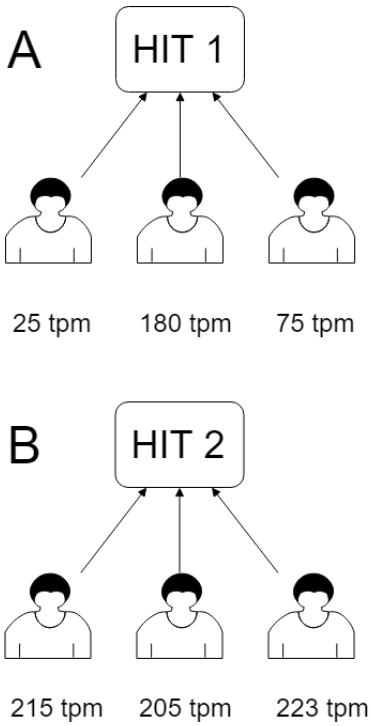


Figure 4.3: An example of two HITs, one that has no agreement in speed at which it should be performed (HIT 1) and another that has (HIT 2).

In previous in-house studies, we removed outliers while considering the distribution of the token processing speed after the removal of values that fall over the defined thresholds [23]. However, we argue that a better approach would be to remove outliers based on the agreement between the token processing speed at the task level, meaning that the HITEXs should be removed based on the standard deviation on HIT level.

To illustrate this removal strategy, consider the example depicted in figure 4.3 which illustrates two distinct scenarios, A and B. In A, one can see that the token processing speed fluctuate considerably, while in B the values are closer to each other, the standard deviation for A and B are 65 and 7, respectively. It is possible to conclude that the task in scenario B is valid and can be solved, on average, at 214 tpm. However, such a conclusion can not be achieved for scenario A due to the sparsity of the values. Having a large sparsity in the processing speed values might be an indicator of a range of issues related to the task itself of the contributors. The task might be poorly formulated (such as the categories being ambiguous, or lack of punctuation) or the contributors may leave the task screen open while performing other activities, while the remainder performed the task at once. In sum, we conclude that one should remove the HITEXs from scenario A, rather than the ones from scenario B.

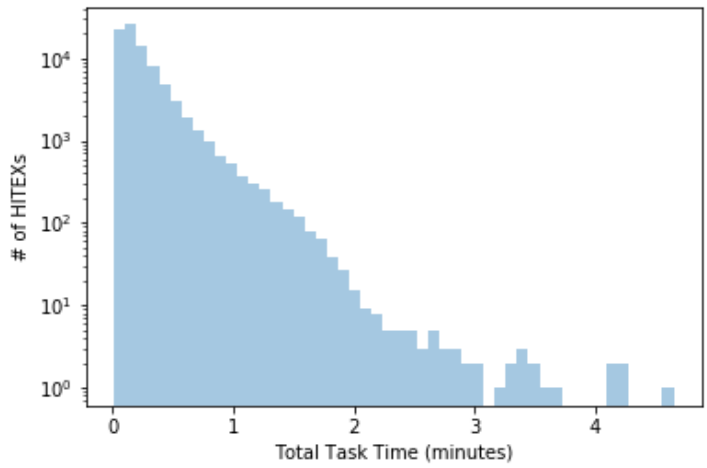


Figure 4.4: The distribution of the target variable, *totalTaskTime*, after the data cleaning process. The y axis is in logarithmic scale.

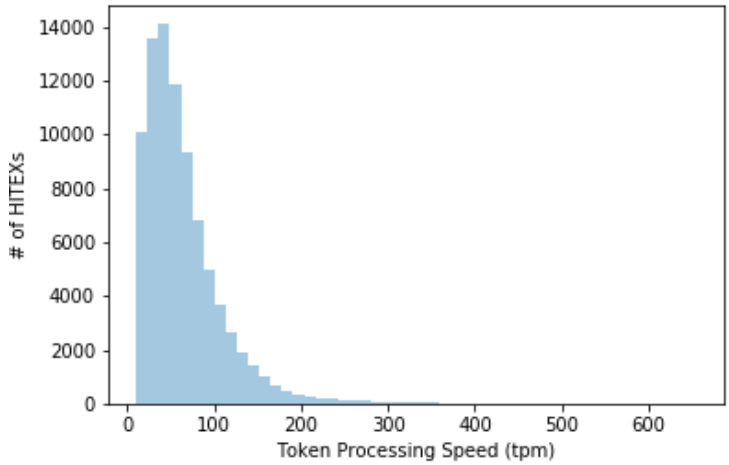


Figure 4.5: The distribution of the token processing speed after the data cleaning process.

This strategy concludes our outlier removal process with the removal of 14,579 HITEXs, which left the dataset with 84,701 HITEXs and 28,708 HITs. Figure 4.4 shows the distribution of the *totalTaskTime* after outliers removal. The tasks' completion times now range from zero up to four minutes.

Since the outliers removal process has been concluded, we proceed to plot the distribution of the token processing speed. The distribution plot can be observed in figure 4.5. From the distribution, we can conclude that the majority of the processing speeds are less than 100 tpm, and values larger than 200 tpm are scarce.

However, the most striking observation is that the distribution of the token processing speed resembles a gamma distribution, more concretely, its probability density function. The gamma distribution is a continuous probability distribution with a shape parameter

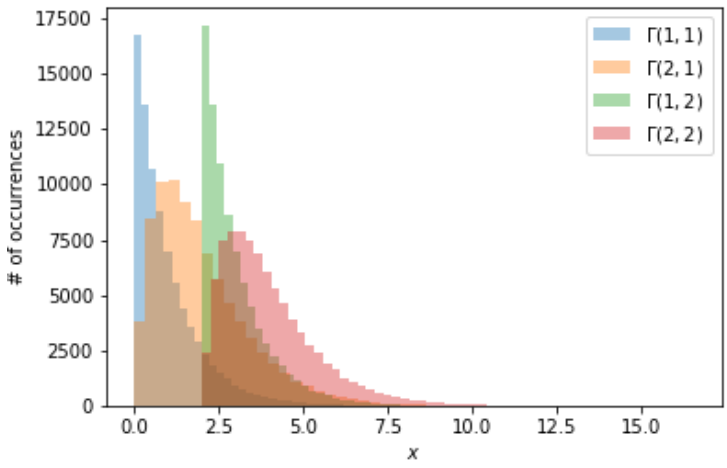


Figure 4.6: Four gamma probability density function distributions where the paremeters, k and θ, vary between 1 and 2.

k and a scale parameter θ and is denoted as $\Gamma(k, \theta)$. Figure 4.6, presents various gamma probability density functions where the value of both parameters varies between 1 and 2. All the distribution have 84,701 samples, the same as our dataset.

We can use this resemblance and know the distribution of our data. Knowing the data's distribution has two main advantages:

- Facilitate future investigation - knowing the distribution of the data allows for future investigation to be conducted even when this dataset is not available as one can generate random values from the gamma's probability function (if the parameters are known). Furthermore, one can quickly reproduce the results of this experiment;

- Estimate efforts more accurately - in a real-world scenario, the processing speed is not the same for every contributor nor every task. Therefore, calculating the generation and validation efforts by using a median or a mean value is not ideal. A more realistic approach is to generate processing speed values and compute the mean processing speed for that task. For instance, consider figure 4.3, in either scenario, we would generate three processing speed values and consider the mean to compute the effort of executing the tasks.

Due to these advantages and the resemblance between the data's distribution and the gamma probability density function, we proceed to estimate the token processing speed's distribution. We do this by leveraging the method `stats.gamma.fit` from the SciPy Python package[1]. This method computes the optimum parameters by maximizing a log-likelihood function, meaning that it performs a maximum likelihood estimation

[1] https://www.scipy.org/ (consulted on 3 September 2019).

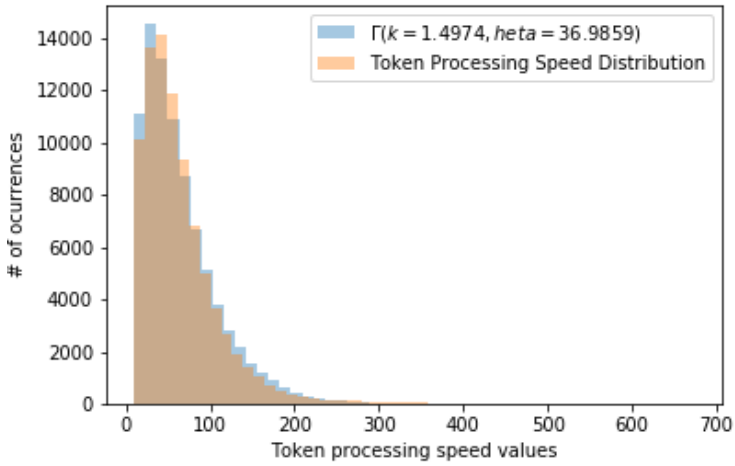


Figure 4.7: The distributions of the token processing speed (orange) and the gamma distribution with the estimated parameters $k = 1.5908$ and $\theta = 33.1918$ (blue).

(MLE)[2]. The estimated parameters, k and θ, are 1.4974 and 36.9859, respectively. The comparison between the token processing speed and the randomly generated values from a distribution $\Gamma(1.4974, 36.9859)$ is presented in figure 4.7. We proceed to use this gamma distribution to estimate our token processing speed for the remainder of this analysis.

4.1.3 Definition of the Reading and Annotation Time

As stated before, the total task time is divided into the time to read the prompt and the time to annotate the entities in the prompt. Having estimated the token processing speed, we can use such estimation to define both the reading and the annotation time. However, the token processing speed accounts for the reading and the annotation time, and thus, we can not directly use this speed value to calculate any of the task's times. Therefore, to calculate these times, we have to use different strategies and calculate new dimensions.

The reading time is the time spent reading the prompt without performing the annotation of any entity. Therefore, the values that better approximate the reading time are the total task time of HITEXs in which no entity was annotated; there are 10,184 HITEXs in this situation. The speeds associated with these HITEXs are the best approximation of the reading speed because these speeds do not account for the annotation of any entity. Furthermore, we hypothesize that the distribution of the reading speed is gamma-distributed as this distribution is a subset of the token processing speed's distribution. The maximum likelihood estimate for k and θ is 1.3652 and 71.1587, respectively. The reading speed distribution and the gamma distribution $\Gamma(1.3652, 71.1587)$ are presented in figure 4.8.

As aforementioned, we can sample speed values to estimate the reading time. To do this, we sample a random number of reading speeds, for instance, and compute the mean.

[2]https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.rv$_c$ontinuous.fit.html(consultedon3September2019).

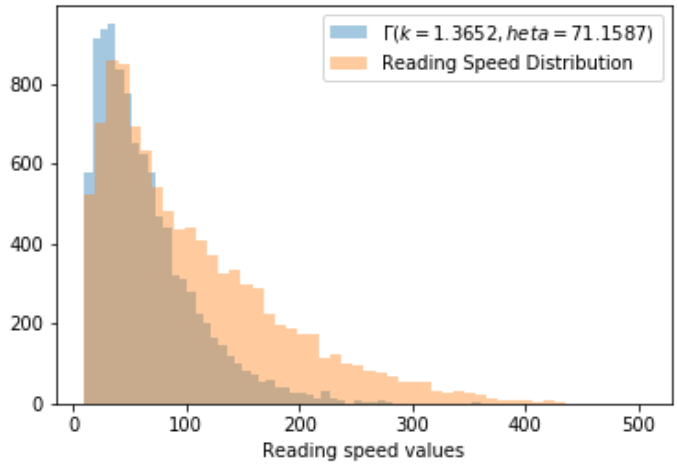


Figure 4.8: The distributions of the reading speed (orange) and the gamma distribution with the estimated parameters $k = 1.3652$ and $\theta = 71.1587$ (blue).

This mean value is used to calculate the reading time (by using equation 4.1). Recall that the total task time is the reading time plus the annotation time, so, to calculate the latter, we can subtract the (predicted) reading time to the total task time. This calculation represents the annotation time per HITEX. Since we aim to define a function that given the number of tokens and entities, each estimates the total task time, we need to calculate the time of annotating one entity and compute the mean across all the executions. This can be done by recurring to equation 4.2.

$$\bar{a} = \frac{\sum_{i=1}^{n} \left(\frac{t_i - r_i}{e_i} \right)}{n} \tag{4.2}$$

In equation 4.2, the notation is as follows:

- n is the number of HITEXs where no entities have been annotated;
- t_i is the total task time of the i-th HITEX;
- r_i is the reading time of the i-th HITEX;
- e_i is the number of entities.

The results regarding the reading and annotation times are presented in table 4.1. It is possible to see in the table that contributors tend to spend more time annotating entities than reading the text. When annotating an entity, the contributor has to perform decisions about the boundaries and the category. These decisions demand a cognitive process that contributes to increasing the annotation time. We proceed to calculate the annotation time per entity and assess whether or not there is a consensus in the time it takes to annotate one entity.

Entities	Total Task Time	Predicted Reading Time	Annotation Time
1	00:10.54	00:06.35	00:04.20
2	00:19.46	00:08.59	00:10.87
3	00:28.45	00:13.80	00:14.65
4	00:38.63	00:20.88	00:17.76
5	00:43.75	00:19.02	00:24.73
6	00:48.93	00:24.51	00:24.42
7	01:00.59	00:21.71	00:38.88
8	01:02.30	00:43.10	00:19.20
9	01:05.10	00:47.16	00:17.93
10	01:13.12	00:36.84	00:36.28
13	03:01.19	00:31.49	02:29:69

Table 4.1: The total task time, predicted reading time, and the annotation time per number of annotated entities in a readable format. The reading time was predicted by recurring to the gamma distribution defined in the previous section. The format of the times is mm:ss.

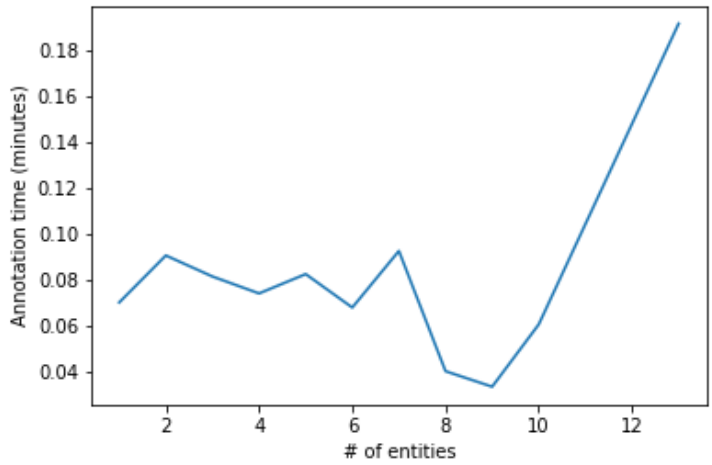


Figure 4.9: The annotation per entity for each number of annotated entities. This value is computed by dividing the annotation time by the number of annotated entities.

Figure 4.9 shows the annotation per entity while varying the number of entities. In the figure, one can visualize that the curve's slope is close to 0 when the number of entities varies between one and eight annotated entities. However, the same conclusion can not be visualized when the number of annotated entities is larger than nine.

This behavior is related to the lack of HITEXs that have more than nine annotated entities. The number of data points for each number of annotated entities is presented in table 4.2. Consequently, we compute a weighted average instead of an arithmetic average, which is equal to 4.05 seconds, or 0.0674 minutes for the values presented in figure 4.9.

We are now able to calculate any HITEXs generation effort given the prompt size in tokens and the number of entities by using the following formula:

# of Entities	# of HITEXs
1	44,690
2	19,000
3	7,374
4	2,450
5	696
6	221
7	63
8	15
9	6
10	1
13	1

Table 4.2: The number of HITEXs per number of annotated entities.

$$h = r + e \times \bar{a},$$

where r is the reading time, e is the number of entities, and $\bar{a}$ is the mean annotation time per entity.

To calculate the generation effort of a job, we need to sum the HITEX generation efforts as in equation 4.3, where h_i is the generation effort of the i-th HITEX.

$$g = \sum_{i=1}^{n} h_i \tag{4.3}$$

In short, we demonstrated a method to calculate any HITEX's generation effort by using the reading speed, and, consequently, we have defined a way of calculating the job generation effort by using the HITEX generation effort. Having the formula to calculate the cost of generating data for a NER job, we can calculate the total job effort and assess the gains of using the two proposed approaches.

4.2 Estimation of Crowd Effort Gains

In this section, we proceed to estimate the total effort of performing a generation task of NER in a crowdsourcing platform. This estimated value is used as a baseline to measure the improvements obtained by the proposed approaches of this thesis. To do this assessment, we use NER datasets that have been annotated in the Neevo platform.

4.2.1 Data

In this experiment, we use nine datasets that have been annotated in a crowdsourcing environment. The datasets had the results of the generation process aggregated at the HIT level. This aggregation has been referred previously as a method of improving the quality of the results produced by the crowd, where, instead of leveraging just the answer of one

Dataset	HITs	Sentences	Tokens	Categories	Entities	Mean sentence length	Entity coverage (%)
CD-1	1,721	22,500	782,276	1	14,098	34.77	1.80
CD-2	2,020	2,037	17,129	10	853	8.41	4.98
CD-3	500	7,143	263,465	1	6,035	36.88	2.29
CD-4	2,413	2,416	12,867	4	2,142	5.33	16.65
CD-5	3,685	3,743	34,439	8	5,583	9.20	16.21
CD-6	5,306	5,307	36,119	14	6,082	6.81	16.84
CD-7	420	1,517	25,648	4	1,779	16.91	6.94
CD-8	421	1,517	25,648	1	425	16.91	1.66
CD-9	2,253	2,254	18,510	12	2,132	8.21	11.52

Table 4.3: Table with the values of number of HITs, sentences, tokens, categories, and entities associated with each of the crowd datasets. It is also presented two additional computed metrics: average sentence length and entity coverage. Note that CD stands for crowd dataset.

contributor, we consider the opinion of a certain number of contributors and aggregate their results. The datasets considered for this experiment were selected based on the Inter-annotator agreement (IAA), which is a measure of between contributors. Typically, the IAA is measure by recurring to the Krippendorff's alpha coefficient [34] and it is recommended to consider only data in which the Krippendorff's alpha is larger than 0.8 [35], which in our scenario only applied to 9 out of 14 datasets. Each row of these datasets is a HIT that has three columns of information:

- HIT ID - the unique identifier of the HIT;
- prompt - the text presented to the contributors;
- result - a list of entities corresponding to the aggregated result of the HITEXs associated with the HIT.

In table 4.3, we present metrics for each one of the crowd datasets; these metrics include the number of HITs, sentences, tokens, categories, and entities, as well as two additional computed metrics: average sentence length and entity coverage.

From table 4.3, one can understand the heterogeneity of the collection of datasets; these vary in terms of size (sentence or token-wise), the number of categories, and entity coverage. The domains of the datasets are also very distinct amongst each other:

- CD-1 and CD-3 are financial reports;
- CD-2, CD-4, CD-5, CD-6, and CD-9 are natural speech sentences, such as "open Spotify and play Gun's and Roses.";
- CD-7 and CD-8 are scripted conversation, meaning that each HIT is a conversation between two individuals.

Dataset	Generation Effort
CD-1	238:57
CD-2	10:21
CD-3	128:00
CD-4	13:43
CD-5	36:09
CD-6	38:57
CD-7	11:34
CD-8	8:07
CD-9	15:51

Table 4.4: The values of total generation effort for each dataset. These are the efforts associated when generating the annotations for the entire dataset. The format of the effort is {hours}:{minutes}.

Having such a wide variety of datasets contributes to assessing the robustness of the proposed approaches.

We have also conducted the first experiment on these datasets. The results can be seen in annex B.

4.2.2 Baseline

Previously, we demonstrated that the reading speed follows a gamma distribution of the form $\Gamma(1.3652, 71.1587)$. We computed both the reading speed and the mean annotation time per entity. These values are used to calculate the generation effort of each dataset. This calculation can be performed by using equation 4.4.

$$g = \frac{\sum_{i=1}^{n} d_i}{v} + \sum_{i=1}^{n} e_i \times \bar{a}, \tag{4.4}$$

In equation 4.4, n is the number of sentences, d_i is the number of tokens of in the i-th sentence, e_i is the number of annotated entities in the i-th sentence, and $\bar{a}$ is the mean annotation time. Note that while equation 4.3 considered , here we consider the number of sentences in the dataset.

Table 4.4 presents the values for the total job effort for each dataset. The values of generation effort are very sparse. Such is due to the heterogeneity in terms of size. These values are used to evaluate the gains of using the proposed approaches as they correspond to the effort of generating annotations for each one of these datasets when requesting the crowd to annotate the entire dataset.

4.2.3 Measurement of Gains

We have defined the baseline value for the total generation effort. This effort corresponds to the necessary effort required by contributors to annotate the entire dataset. However, in the proposed approaches, we only require the datasets to be partially annotated, while

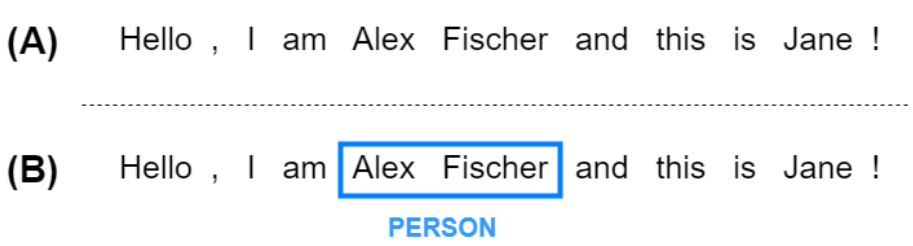


Figure 4.10: The same sentence in two distinct scenarios. In scenario A, the sentence is not pre-annotated depicting a generation task. In scenario B, the sentence is pre-annotated depicting a validation task. Note that the pre-annotation in scenario B has an error: the token "Jane" as not been annotated as an entity.

the remainder of the dataset is annotated based on the model's predictions. Therefore, we need to calculate the generation efforts associated with the different amount of training data and then calculate the validation effort associated with the model's predictions.

In this experiment, we consider the amounts of training data defined in section 3.2.3 (1%, 3%, 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 100%). The generation effort associated with these amounts of training data can be calculated by using the equation 4.4. Note that these amounts of training data are relative to size of dataset in terms of sentences.

Once the model is trained, we use it to generate annotations for the remainder of the data. Such annotations have to be validated by the contributors. In the validation task, the total task time depends on the prompt size and the number of errors that need correction, which means that the calculation of the validation effort can be performed by using equation 4.4. However, instead of considering that e_i is the number of entities in the i-th sentence, we consider that it is the number of errors in the i-th sentence.

Figure 4.10 presents the same sentence in two distinct scenarios. In scenario A, the sentence is not annotated and, in scenario B, the sentence is pre-annotated with one entity (person). Therefore, scenario A depicts a generation task, while scenario B depicts a validation task. Note that, in scenario B, not all the person entities have been annotated (Jane is not annotated). Therefore, this validation task requires the correction of a false negative error.

For simplicity, assume that the reading speed in both scenarios is 84 tpm and that the mean annotation time is the same as the mean validation time (which is the time to correct an error) and is equal to 5 seconds. Therefore, to calculate the effort in scenario A, we do the following:

- Compute the reading time by dividing the number of tokens (11) per the reading speed (84) $\frac{11}{84}$;
- Compute the annotation time by multiplying the number of entities (2) per the mean annotation time in minutes $\left(\frac{5}{60}\right)$ $2 \times \frac{5}{60}$;

- Compute the sum between these two values, which is equal to 0.30 minutes (around 18 seconds).

To calculate the effort in scenario B, we do the following:

- Compute the reading time by dividing the number of tokens (11) per the reading speed (84) $\frac{11}{84}$;
- Compute the validation time by multiplying the number of errors (1) per the mean validation time in minutes $\left(\frac{5}{60}\right) 1 \times \frac{5}{60}$;
- Compute the sum between these two values, which is equal to 0.21 minutes (around 13 seconds).

Note that, we make two assumptions when calculating the validation effort: the reading time is independent of the prompt being pre-annotated and the cost of correcting any NER error is the same as the time to annotate one entity. These assumptions are necessary because the Neevo platform does not support NER validation jobs.

However, calculating the validation effort under these conditions is considered the worst-case scenario. In the literature, it has been shown that contributors experience a decrease in the reading time when tasked to complete the task of NER on pre-annotated text [38]. Furthermore, when the prompt is pre-annotated, the identifying entities are colored. The usage of color has been shown to benefit the search for on the screen [13, 17, 20, 27], meaning that color is a powerful method to draw the contributors' attention to parts of the prompt that need to be validated.

Regarding the second assumption, intuitively, one can understand that different errors of NER have different validation efforts as they require different decisions and actions from the annotators (see the errors in 2.2):

- False negative - the contributor needs to decide about the boundaries and categories. The required actions are the redefinition of both the boundary and the category (reselect the entity and select the category);
- Boundary-category - the contributor needs to decide about the boundaries and categories. The required actions are the redefinition of both the boundary and the category (reselect the entity and select the category);
- Boundary - the contributor needs to decide about the Boundary. The required action is the redefinition of the boundaries (reselected the entity).
- Category - the contributor needs to decide about the category. The required action is the redefinition of the category (click the entity and select the new category).
- False positive - the contributor needs to decide about the boundaries and categories. The required action is to click the entity and click on a button that removes the entity.

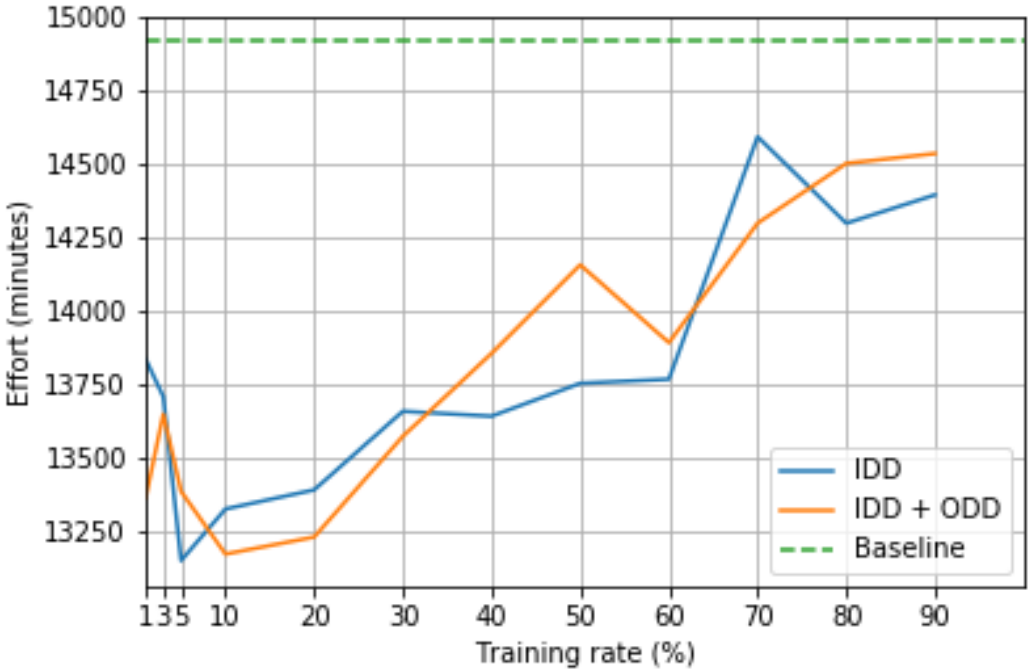


Figure 4.11: A line plot with the total effort curves, in minutes, when training the model with in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line) when compared against the baseline effort (green dashed line) for the different amounts of training data. These effort values are associated with the amounts of training data of CD-1.

Both the false negative and the boundary-category errors required the same decisions and actions as the generation of one entity, while the remainder requires fewer decisions or more straightforward actions. Therefore, only the validation time of two types of errors are the same as the annotation time, while the remainder should be less.

The values of the generation and validation effort for dataset CD-1 and CD-4 when using the two approaches are presented in figures 4.11 and 4.12, respectively. The results for the remainder of the datasets can be found in appendix C. The total effort values present a tendency to decrease when the amount of training data is low. In figure 4.11, the total effort decreases until it reaches its minimum value in the 5% of training data, for IDD only, and 10% of training data, for IDD + ODD. In figure 4.12, the total effort decreases until it reaches its minimum value in the 5% of training data, for both approaches. In an initial phase, the training data that is generated contributes greatly to increasing the model's performance, and, consequently, its predictions. However, from a certain point, adding more data does not contribute greatly to reduce the validation effort as the model's performance does not increase substantially. Therefore, generating data is ineffective as it does not produce any effort gains.

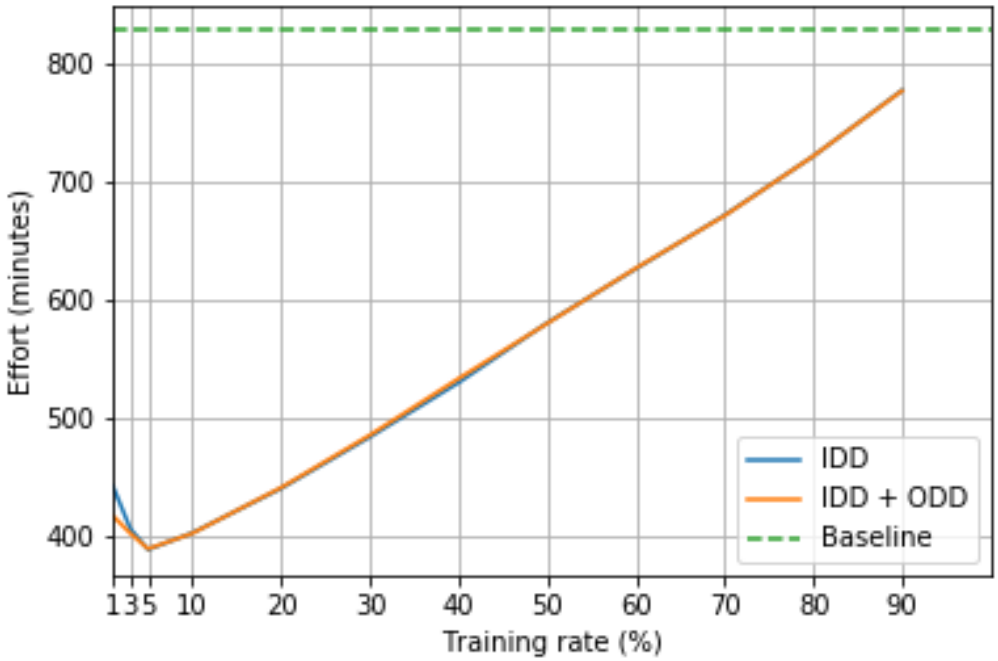


Figure 4.12: A line plot with the total effort curves, in minutes, when training the model with in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line) when compared against the baseline effort (green dashed line) for the different amounts of training data. These effort values are associated with the amounts of training data of CD-4.

The most significant conclusion extracted from these results is that both approaches are most cost-efficient when compared to having the crowd annotate the entire dataset. The best improvement concerning CD-1 is 8.52% when using only in-domain data, and 8.50% when adding out-domain data. Regarding CD-4, the best improvement while using only in-domain data is 53.95% and 52.71% with the addition of out-domain data. The usage of out-domain data does not produce any gains regarding the best effort values.

The plots show that both curves are intertwined, which entails that the relative gain of using ODD is minor. As can be seen, the usage of ODD and Transfer Learning renders better results when the amount of training data less than 5%. The most substantial effort savings are 1.49% for CD-1 and 0.27% for CD-4. Such savings translate into absolute efforts of 214 minutes (which is around 3-4 hours) for CD-1 and 23 minutes for CD-4.

In sum, both approaches can achieve better effort values when compared to the traditional approach to crowdsourcing, even when considering the worst-case scenario. Furthermore, the efforts gains achieved across all datasets (except for CD-8) present the same behavior. The total effort decreases until a certain amount of training data, and then it starts to increase.

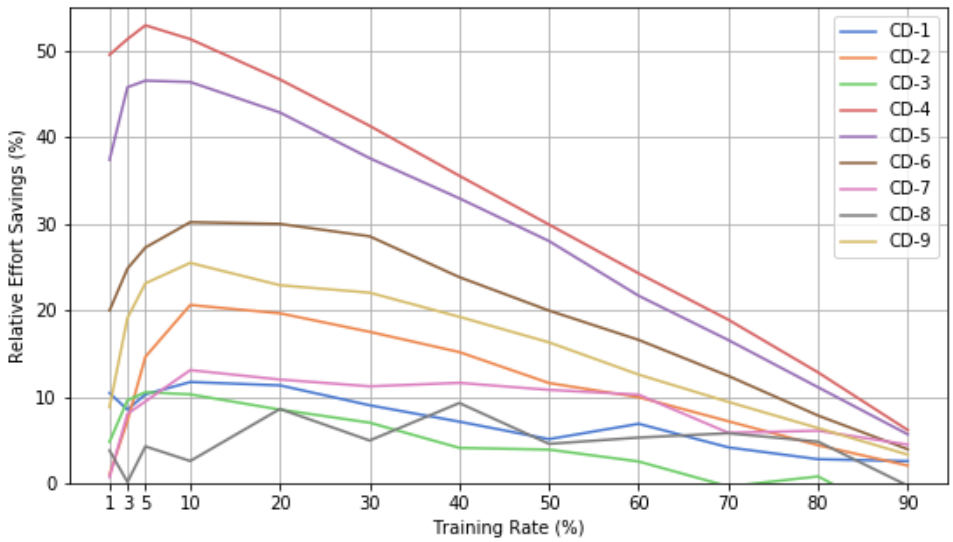


Figure 4.13: The relative effort gains for all crowd datasets for the different amounts of training data.

In figure 4.13, the relative effort gains increase until a certain amount of training data (which for the majority is no larger than 20%), and then the gains start to decrease until they reach (or get close to) 0%.

As stated, we have considered the worst-case scenario to calculate the effort values in the validation step. The assumptions mentioned at the beginning of this section created the necessity of examining the experiment's results from a different point-of-view. To do this, we defined a new metric that does not account for the reading time, as we consider it to be independent of the pre-annotations, and that it considers that any NER error requires the same validation effort. We called such metric the index of saved actions.

4.2.4 Index of Saved Actions

The index of saved actions is a metric that shows the relation between the amount of saved and the amount of induced work by the model and can be computed by using the formula in equation 4.5.

$$\rho = \frac{\sum_{i=1}^{n}(x_i - y_i)}{\sum_{i=1}^{n}(x_i + y_i)} \tag{4.5}$$

In equation 4.5, x_i is the number of true positives in the i-th sentence of the generation set, and y_i is the number of error in the i-th sentence of the generation set. Note that, a true positive predicted by the model is considered to be a positive action as it requires no effort from the crowd to validate nor correct, whereas an erroneous entity is considered a negative action as it requires effort from the crowd. When this metric has the value -1 means that no work has been saved since no true positive was predicted, when this

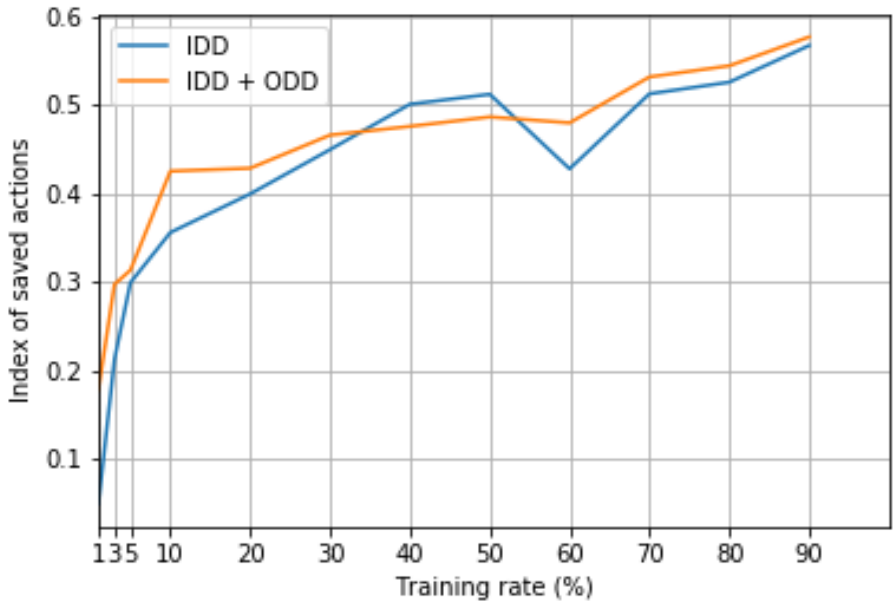


Figure 4.14: The curves of the index of saved actions when training the model with only in-domain data (blue line) in-domain data and out-domain data (orange line). These values are associated with the amounts of training data of CD-1.

metric has the value 1 means that no work needs to be performed as no errors have been induced.

This index is particular because it allows drawing a meaningful conclusion concerning as to when the model's training should be triggered as the model is guaranteed to save work. Consider the following situation: we have x total entities in a given job. We annotate y entities in the generation step, which leaves w entities left to annotate in the remainder of the data, such that $w = x - y$. Assuming that each error has the same correction time, then if the model outputs a true positive entities and b erroneous entities, such that $a \leq w$ and $a > b$. We can infer that $b < w$, which means that the work induced by the model is not superior to the number of entities that the contributors would be required to generate (when under the traditional crowdsourcing setting).

The results of the index of saved actions for CD-1 and CD-4 for the different amounts of training data for both approaches are presented in figures 4.14 and 4.15, respectively. In these figures, one can note the same relation between the usage of ODD: the best gains are attained when the amount of training data is less than 5%. However, the relative gains in this index are substantially larger when comparing to the gains obtained for the effort. In CD-1, the best gain is 6.45% and, in CD-4, the best gain is 4.60%. These improvements indicate that models trained with the aid of ODD produce predictions with better quality than those of models who are trained without it.

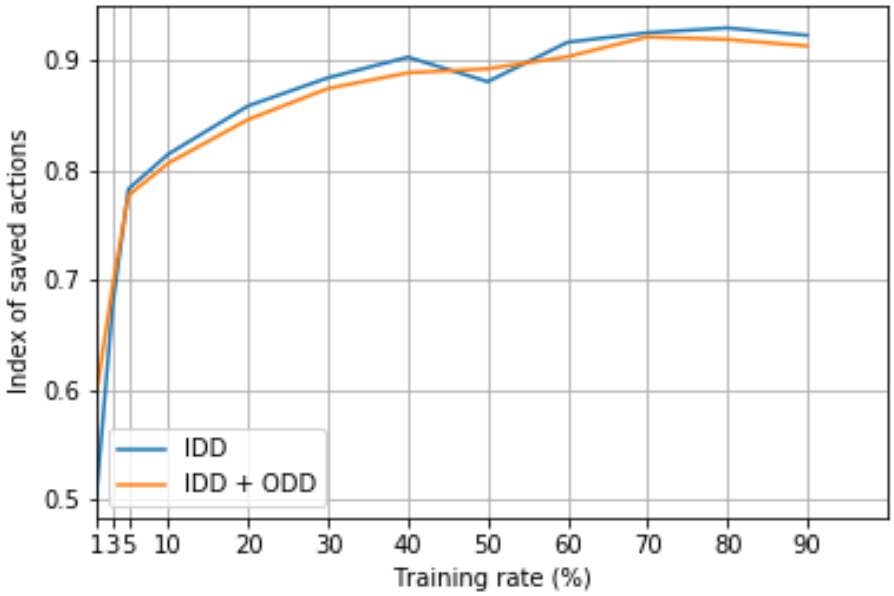


Figure 4.15: The curves of the index of saved actions when training the model with only in-domain data (blue line) in-domain data and out-domain data (orange line). These values are associated with the amounts of training data of CD-4.

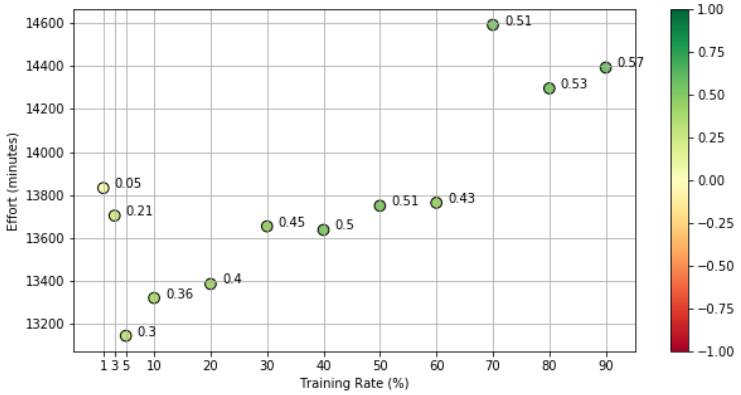


Figure 4.16: A scatter plot corresponding to the relation between the total effort and the index of saved actions (mapped to the color) when the models are trained using only in-domain data. These values are associated with the amounts of training data of CD-1.

In figures 4.16 and 4.17, one can see the relation between the index of saved actions (mapped to the color) and the total effort (circles). Both plots show that even for 1% of training data, there is a saving in terms of actions. Furthermore, the best effort value (associated with 5% of training data in both plots) corresponds to an index of saved actions larger 0. This relation proves that the model's predictions are saving more work than the work they are inducing (such as false positives) or left undone (false negatives).

In appendix C, we present the values of the index of saved actions and this section's

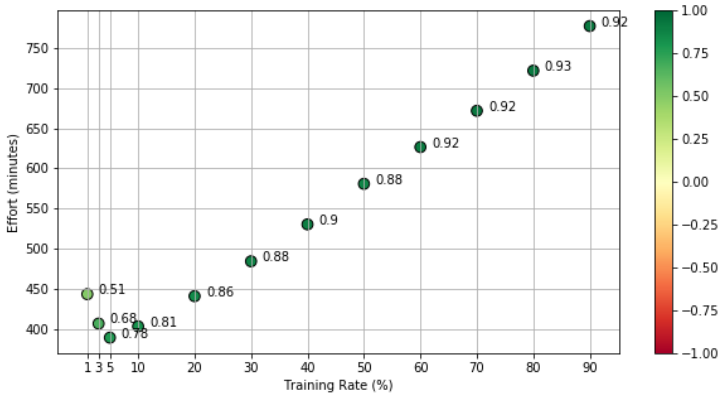


Figure 4.17: A scatter plot corresponding to the relation between the total effort and the index of saved actions (mapped to the color) when the models are trained using only in-domain data. These values are associated with the amounts of training data of CD-4.

Dataset	IDD			IDD + ODD		
	Absolute	Relative (%)	ρ	Absolute	Relative (%)	ρ
CD-1	20:21	8.52	0.56	20:18	8.50	0.58
CD-2	02:13	21.45	0.63	02:16	21.90	0.61
CD-3	15:28	12.09	0.71	18:10	14.19	0.68
CD-4	07:16	52.95	0.93	07:14	52.71	0.92
CD-5	16:41	46.15	0.81	16:38	46.00	0.79
CD-6	12:23	31.78	0.42	11:50	30.37	0.40
CD-7	01:27	12.62	0.16	01:21	11.69	0.18
CD-8	00:42	8.72	0.33	00:30	6.19	0.12
CD-9	04:08	26.13	0.45	03:55	24.76	0.36

Table 4.5: The best absolute and relative gains, and the values index of saved actions ρ achieved for each dataset when using the IDD and IDD + ODD approaches. The relative gains range from 6% to 53% and the ρ indicates that the model's predictions save more work than what they induced.

analysis for the remainder of the data. However, we present the best absolute and relative gains of using IDD and IDD + ODD for each dataset as well as the corresponding index of saved actions. The results are presented in table 4.5.

The results in this table show that the usage of IDD causes effort gains ranging from 42 minutes to 20 hours in absolute terms, and 9% to 53% in relative terms and the index of saved actions is always larger than 0. Furthermore, note that the best effort values are achieved when the amount of training data is smaller than 30%. The two exceptions to the rule are datasets CD-7 and CD-8, which achieve the best effort values at 50% and 40% of training data, respectively.

The addition of ODD does wield similar effort gains. However, there are cases in which the addition of ODD does not improve the results of the IDD and, for the best effort values, the relative gain of using ODD is meager. The most considerable improvement

Dataset	IDD			IDD + ODD		
	Absolute	Relative (%)	ρ	Absolute	Relative (%)	ρ
CD-1	07:41	3.22	0.05	10:14	4.28	0.18
CD-2	00:11	1.88	-0.92	00:15	2.55	-0.88
CD-3	06:21	4.96	-0.43	14:18	11.18	-0.01
CD-4	06:23	46.53	0.51	06:44	49.15	0.60
CD-5	11:30	31.82	0.14	13:18	36.82	0.27
CD-6	06:59	17.95	-0.33	07:56	20.40	-0.29
CD-7	00:00	0.00	-0.79	00:00	0.00	-0.80
CD-8	00:06	1.43	-0.85	00:28	5.82	-0.82
CD-9	01:33	9.84	-0.46	01:17	8.19	-0.40

Table 4.6: The absolute and relative gains for 1% of training data, and the values index of saved actions ρ achieved for each dataset when using the IDD and IDD + ODD approaches.

of using ODD is 2.10% for CD-3. However, when considering 1% of training data, the usage of ODD produces better results; this can be observed in table 4.6.

Note that in table 4.5, the addition of ODD only wielded better results for CD-2 and CD-3, and the largest improve was 2.10%. In table 4.6, there are improvements in every dataset besides CD-7 and CD-9. These improvements range from 0.67% to 6.22%. The index of saved actions registered significant improvements as well. While in table 4.5, the only observable improvement is associated with CD-1, in table 4.6, there are improvements in every dataset. These gains are an indicator that the models are producing fewer errors, which implies that the usage of ODD contributes to further minimizing the amount of work required from the crowd.

4.2.5 Simulated Crowdsourcing Scenario

In a crowdsourcing scenario, dividing the data in terms of the percentage of sentences is not feasible; since the atomic unit of work in crowdsourcing is the HIT. To illustrate this statement, consider that a NER job is composed of two HITs, which have ten sentences each. Note that, when one HIT is completed, we have ten annotated sentences available for training the model. It is not sensible to discard sentences and train the model with only a portion of these ten sentences. However, this situation could happen if we were to consider only 10% of the number of sentences in the dataset.

Furthermore, if we train the model with two sentences only, we expect the crowd to validate the 18 sentences that are left. This situation is not possible in a real-world environment. Therefore, a more sensible approach is to divide the data in terms of the absolute number of HITs.

# of HITs	Generation Effort	Validation Effort	Total Effort
100	16:26	204:56	221:22
200	32:04	194:32	226:37
300	47:57	176:12	224:10
400	68:21	162:53	231:14
500	82:09	144:38	226:47

Table 4.7: The values of effort for the different amount of training data in terms of the absolute number of HITs for CD-1. These values are associated with training the models with in-domain data.

# of HITs	Generation Effort	Validation Effort	Total Effort
100	00:40	05:48	06:28
200	01:21	05:31	06:53
300	01:57	05:00	06:58
400	02:39	04:42	07:21
500	03:19	04:21	07:41

Table 4.8: The values of effort for the differente amount of training data in terms of the absolute number of HITs for CD-4. These values are associated with training the models with in-domain data.

# of HITs	Absolute Effort Savings	Relative Effort Savings (%)	ρ
100	17:34	7.35	0.30
200	12:19	5.16	0.33
300	14:46	6.18	0.41
400	07:42	3.22	0.45
500	12:09	5.09	0.41

Table 4.9: The absolute effort savings, relative effort savings, and index of saved actions for the different amounts of training data. These values are associated with CD-1.

We selected the values of 100, 200, 300, 400 and 500 HITs to conduct this experiment. For simplicity, we present the results of using only in-domain data in tables 4.7 and 4.8 for CD-1 and CD-4, respectively. Note that these effort values are similar to the ones obtained previously in this section, which indicates that the effort values are independent of the data division. Furthermore, this division is much more fitting for automating the process of finding the point to trigger the model's training and stop the generation step in the crowdsourcing pipeline. In DefinedCrowd, project managers can control the amount of HITs that are sent to the crowd for annotation. Having a tool that suggests the amount of HITs that should be sent and sends alerts to stop the generation step is one way to apply this research to a real-world use case.

In tables 4.9 and 4.10, we present the absolute effort savings, the relative effort savings, and the index of saved actions ρ for the different amounts of training data for CD-1 and CD-4, respectively. Note that, the best effort values are obtained when the amount of training data is equal to 100 HITs. However, this observation does not necessarily entail

# of HITs	Absolute Effort Savings	Relative Effort Savings (%)	ρ
100	07:14	52.74	0.78
200	06:49	49.78	0.79
300	06:44	49.17	0.84
400	06:21	46.33	0.85
500	06:01	43.92	0.84

Table 4.10: The absolute effort savings, relative effort savings, and index of saved actions for the different amounts of training data. These values are associated with CD-4.

Dataset	Absolute Effort Savings	Relative Effort Savings	ρ
CD-1	17:34	7.35	0.30
CD-2	02:20	22.7	0.52
CD-3	11:07	8.69	0.53
CD-4	07:14	52.74	0.53
CD-5	16:55	46.81	0.72
CD-6	12:39	32.51	0.24
CD-7	01:21	11.78	0.03
CD-8	00:34	7.01	0.05
CD-9	04:10	26.37	0.30

Table 4.11: The best absolute and relative gains achieved for each one of the datasets when using only IDD for training the models. The corresponding index of saved actions ρ is also presented.

that the best value can not be found when the amount of training data is inferior to 100 HITs.

Recall that the amount of training data, in terms of percentage of sentences, that wielded the best effort savings was 5% for both datasets. This percentage corresponds to roughly 73 HITs for CD-1 and 103 HITs for CD-4. Therefore, regarding CD-1, the best effort savings has already been achieved before the 100 HITs and, regarding CD-4, the best effort savings is not present on the table due to the step between amounts of training data being too large.

In table 4.11, we present a summary of the absolute and relative gains achieved for each dataset, as well as the index of saved actions ρ. These values are associated when training the models with only in-domain data. Note that, in the majority of the datasets, we are not able to achieve better efforts when comparing to dividing the dataset in terms of the percentage of sentences. However, these efforts represent the efforts that would be estimated in a production environment. Note that the range of HITs that we consider was utterly arbitrary. In some datasets, 100 tasks represent 20% of the dataset, while in others 100 tasks are less than 5%. Therefore, in some datasets, the optimum point is reached before the 100 HITs, while in others, it is reached after. The results of this experiment are presented in appendix D. We proceed to present a summary of this chapter and further discuss the results of this experiment.

4.3 Discussion

In this chapter, we carried the experiment that answers the second research question, which relates as to how can IDD and ODD be used to improve the current approach to crowdsourcing. To do so, we divided this experiment into two phases:

- study of the required effort to execute the task of NER in a crowdsourcing platform;
- and calculate the effort gains of using IDD with or without the addition of ODD.

The statistical analysis started with the data cleaning process due to the existence of negative values concerning the total task time. We proceeded to calculate the token processing speed, which is the speed at which a contributor executes a task of NER. We leveraged this speed to remove outliers in the data by computing the standard deviation at HIT level. We removed the tasks in which the standard deviation was high as such is an indicator of disagreement between the time to perform it.

After removing the outliers, we proceeded to define methods to calculate the reading time and the annotation time. The reading time is calculated by using equation 4.1, where v is the reading speed. This reading speed is also used to compute the mean annotation time. The formula is presented in equation 4.2.

Having the reading speed and the mean annotation time enables the calculation of the generation effort and validation effort by using equation 4.3. Recall that the generation effort is dependent on the number of annotated entities, and the validation effort is dependent on the number of erroneous entities.

We consider the worst-case scenario when calculating the validation effort by establishing two assumptions:

- the annotators do not experience any speedup in the reading time when validating pre-annotated text;
- the cost of correcting any NER is the same as generating one entity.

Despite considering the worst-case scenario, we have consistently achieved better effort values in our approaches when comparing to the approach of having the crowd generate annotations for the entire dataset. The effort gains range from 6% to 53%, and thus we can conclude that our proposed approach is reliable and can effectively improve the current approach to crowdsourcing.

Furthermore, we have defined the index of saved actions. This metric is independent of the reading time and assumes that generating and correcting entities requires the same effort. Therefore, this metric accounts for the assumptions that we have defined for this experiment. When this metric assumes values superior to 0, it means that we are saving work regarding the crowd. The results of this experiment do show that this metric is always superior to 0 for the best effort value, which contributes to show the robustness of the attained results.

Finally, we concluded this chapter by presenting a way of dividing the datasets that best simulates a real-world use case. Dividing the datasets in terms of the absolute number of HITs facilitates the automatization of finding the point where the model's training should be triggered, and the generation of data should be stopped.

In sum, we can conclude that the usage of in-domain data benefits the crowdsourcing pipeline, turning it more efficient as the results surpass the traditional approach to crowdsourcing. The usage of out-domain data is only worthwhile when the amount of training data is scarce. For the best effort values attained, the usage of out-domain data proved to be ineffective regarding the total effort savings as the improvements produced are not significant when comparing to not using it. However, we have also demonstrated the robustness and range of applications of the proposed approach by applying it to nine datasets with very diverse characteristics.

Chapter 5

Conclusions and Future Work

It is only through labor and painful effort, by grim energy and resolute courage, that we move on to better things.

– Theodore Roosevelt

In recent years, the wide adoption of data-driven approaches in ML lead the search for scalable annotation solutions. Crowdsourcing can be used to produce annotations for data, which is then used to train ML models. More recent approaches to crowdsourcing have also been leveraging ML algorithms to optimize further the process of annotating data. However, these approaches tend to use pre-trained models which have limited usability and produce predictions with a significant error rate. In this thesis, we proposed two approaches that have a broader spectrum of usability and can generate predictions with better quality. As proof of concept, we applied both approaches to crowdsourcing pipeline of NER.

To prove the feasibility of our procedures, we formulated two research questions:

- **Considering the state of the art regarding NER, how does the amount of training data influence the model's performance?**
- **How can the in-domain data approach reduce time costs when integrated into a crowdsourcing pipeline?**

The experiment that answered the first research question involved the implementation of the state of the art regarding the task of NER and training these models while varying the amount of in-domain data. In this experiment, we define studied the learning curves of the models while being trained with in-domain data and in-domain data with out-domain data. We observed a stabilization point around the 20-30% quantity of training data. This stabilization point is the amount of training data from which adding more

data to the training process does not contribute significantly to improve the model's performance. This finding is particularly interesting as it allows us to conclude that generating annotations past this stabilization point does not contribute to increasing the model's performance significantly. Therefore, we can assume that to ensure quality in the predictions, we require, in the worst-case scenario, the generation of annotations up to 20-30%.

To answer the second research question, we conduct a statistical analysis on the executions of contributors in a crowdsourcing platform to estimate the necessary time to perform a generation and validation task of NER. From the estimation and the assessment of the absolute and relative gains of using IDD and IDD + ODD, one can conclude that using IDD attains significant improvements when comparing to the traditional crowdsourcing strategies. However, the usage of ODD to boost upon the results of IDD does not wield statistically significant results. Note that this, enhancement is when considering the best effort values. The usage of ODD can produce better effort improvements when the amount of training data is close to 1%.

However, under the current setup – which consists of generating annotation in a portion of the dataset, train a model once, and generate the predictions for the remainder of the data – the usage of ODD is irrelevant as the results produced are not statistically significant. We can also conclude that defining a new setup that favors the usage of fewer amounts of data (less than 1%) benefits from the application of ODD.

Furthermore, the proposed approach has its limitations:

- Quality of the annotations - in this thesis, we verified the quality of the annotations by recurring to the Krippendorf's alpha as a measure of the IAA, and discarded the datasets in which the alpha was less than 0.8;

- Existence of a gold set - in this thesis, we used the gold set to assess the model's performance – served the purpose of a test set. In a real-world scenario, the data is partially annotated meaning that we have no way of estimating the number of errors present in the model's predictions. The gold set can be used to assess both performance and estimated effort. The gold set is a portion of the data which is not generated by the contributors. Typically, it is generated apriori by in-house experts; however, the process of generating a gold set is required to ensure quality in the data (through the methods described in section 2.1.1).

However, we recommend the usage of IDD as it optimizes the current approach to crowdsourcing, even in the worst-case scenario, where no contributors experience no speedup in the reading time and assuming that correcting any NER error is the same as generating an entity. Furthermore, the usability of this approach is far broader than the pre-trained model approach as it can be used in, any scenario provided that the limitations are fulfilled.

5.1 Contributions

Throughout our experiments, we have developed software, metrics, and equations, that are contributions to the literature regarding crowdsourcing, NER, and ML in general. Such contributions of our work are:

- A framework to define, train, and test neural network architecture for sequence labeling tasks. It is also possible to save models and use them to generate annotated data on any corpus;
- Observation of a stabilization point during the training process of a neural network for the task of NER when training the model with only in-domain data and with in-domain data and out-domain data;
- The finding that the speed at which NER tasks are completed in a crowdsourcing platform is gamma-distributed;
- The definition of equations that allows the estimation of the required effort to generate and validate annotations in the task of NER;
- The definition of the index of saved actions, which is a metric that measures the trade-off between the induced and saved work by the model's predictions;
- The definition of an approach that improves the current process of crowdsourcing. This approach defines two learning settings to train the models: use only in-domain data and in-domain data with out-domain data.

5.2 Future Work

In this thesis, we rely upon the finding of the model's stabilization point to trigger the model's training. This process can be performed automatically, for instance, using a stochastic gradient descent approach.

This thesis investigation focused on the usage of ODD to perform domain adaptation. However, other types of adaptation are possible, such as application and language adaptation. In the future, a similar investigation should be carried for these scenarios.

Further investigation should focus on assessing the cost of validating pre-labeled tasks of NER in a crowdsourcing platform. The validation is influenced by the different errors produced by a model's predictions and the reading time. Studies show that contributors experience a speedup when performing the task of NER in pre-annotated texts [38]. The study of this speedup factor should focus on assessing under which conditions this phenomenon happens and measure it.

Another course of investigation relates to reducing the number of tasks that should be sent to validation and how they should be sent. To every model prediction, there is an

associated array of probabilities which are used by the tag decoder to select the appropriate category. We refer to the probability of the selected category as confidence. We can compute the trust at the task level and assess the confidence that the model predicted for each category. If this value is larger than a pre-defined threshold, for instance, 0.7, than we do not need to send the task to validation.

Another feature is that even if the average confidence value is lower than the pre-defined threshold, we can still check the individual predictions. Those predictions whose confidence is larger than the threshold, instead of allowing the correction of this entities, we should present them as examples; this may facilitate the task as it allows the contributors to learn by visualizing examples of task's goal.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. "Tensorflow: A system for large-scale machine learning." In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 265–283.

[2] A. Ahmed, K. Yu, W. Xu, Y. Gong, and E. Xing. "Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks." In: *European Conference on Computer Vision*. Springer. 2008, pp. 69–82.

[3] A. Akbik, D. Blythe, and R. Vollgraf. "Contextual string embeddings for sequence labeling." In: *Proceedings of the 27th International Conference on Computational Linguistics*. 2018, pp. 1638–1649.

[4] M. Asahara and Y. Matsumoto. "Japanese named entity extraction with redundant morphological analysis." In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics. 2003, pp. 8–15.

[5] T. Bell. "Extensive reading: Speed and comprehension." In: *The reading matrix* 1.1 (2001).

[6] D. M. Bikel, S. Miller, R. Schwartz, and R. Weischedel. "Nymble: a high-performance learning name-finder." In: *arXiv preprint cmp-lg/9803003* (1998).

[7] A. Borthwick, J. Sterling, E. Agichtein, and R. Grishman. "Exploiting diverse knowledge sources via maximum entropy in named entity recognition." In: *Sixth Workshop on Very Large Corpora*. 1998.

[8] D. Brabham. *Crowdsourcing*. MIT Press Essential Knowledge series. MIT Press, 2013. ISBN: 9780262314251. URL: https://books.google.pt/books?id=ndcnAAAAQBAJ.

[9] N. Chinchor. "Overview of MUC-7." In: *Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29-May 1, 1998*. 1998.

[10] J. P. Chiu and E. Nichols. "Named entity recognition with bidirectional LSTM-CNNs." In: *Transactions of the Association for Computational Linguistics* 4 (2016), pp. 357–370.

[11] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. "On the properties of neural machine translation: Encoder-decoder approaches." In: *arXiv preprint arXiv:1409.1259* (2014).

[12] F. Chollet et al. *Keras*. https://keras.io. 2015.

[13] R. E. Christ. "Review and analysis of color coding research for visual displays." In: *Human factors* 17.6 (1975), pp. 542–570.

[14] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. "Natural language processing (almost) from scratch." In: *Journal of machine learning research* 12.Aug (2011), pp. 2493–2537.

[15] R. Dawson and S. Bynghall. *Getting results from crowds*. Advanced Human Technologies San Francisco, CA, 2012.

[16] M. De Leeuw and E. De Leeuw. *Read better, read faster: a new approach to efficient reading*. Vol. 740. Penguin Books, 1965.

[17] S. A. Demars. "Human factors considerations for the use of color in display systems." In: (1975).

[18] T. Dozat. "Incorporating nesterov momentum into adam." In: (2016).

[19] H. C. Ellis. "The transfer of learning." In: (1965).

[20] C. W. Eriksen. "Location of objects in a visual display as a function of the number of dimensions on which the objects differ." In: *Journal of Experimental Psychology* 44.1 (1952), p. 56.

[21] E. Fry. *Teaching faster reading: a manual*. University Press, 1963.

[22] A. Gandomi and M. Haider. "Beyond the hype: Big data concepts, methods, and analytics." In: *International journal of information management* 35.2 (2015), pp. 137–144.

[23] I. Gomes. "Throughput Forecasting for Crowdsourced Text-Enrichment." Master's thesis. Faculdade de Engenharia da Universidade do Porto, 2019.

[24] R. Grishman and B. Sundheim. "Message understanding conference-6: A brief history." In: *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*. 1996.

[25] J. Hill. "Effective Reading in a Foreign Language: An Experimental Reading Course in English for Overseas Students." In: *English Language Teaching Journal* 35.3 (1981), pp. 270–81.

[26] M. Hirth, T. Hoßfeld, and P. Tran-Gia. "Analyzing costs and accuracy of validation mechanisms for crowdsourcing platforms." In: *Mathematical and Computer Modelling* 57.11-12 (2013), pp. 2918–2932.

[27] W. D. Hitt. "An evaluation of five different abstract coding methods—Experiment IV." In: *Human Factors* 3.2 (1961), pp. 120–130.

[28] S. Hochreiter. "The vanishing gradient problem during learning recurrent neural nets and problem solutions." In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116.

[29] S. Hochreiter and J. Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[30] J. Howe. "The rise of crowdsourcing." In: *Wired magazine* 14.6 (2006), pp. 1–4.

[31] J. Howe. *Crowdsourcing: How the power of the crowd is driving the future of business*. Random House, 2008.

[32] J.-T. Huang, J. Li, D. Yu, L. Deng, and Y. Gong. "Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers." In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 7304–7308.

[33] E. Kamar, S. Hacker, and E. Horvitz. "Combining human and machine intelligence in large-scale crowdsourcing." In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems. 2012, pp. 467–474.

[34] K. Krippendorff. "Computing Krippendorff's alpha-reliability." In: (2011).

[35] K. Krippendorff. *Content analysis: An introduction to its methodology*. Sage publications, 2018.

[36] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer. "Neural architectures for named entity recognition." In: *arXiv preprint arXiv:1603.01360* (2016).

[37] J. Li, A. Sun, J. Han, and C. Li. "A Survey on Deep Learning for Named Entity Recognition." In: *arXiv preprint arXiv:1812.09449* (2018).

[38] T. Lingren, L. Deleger, K. Molnar, H. Zhai, J. Meinzen-Derr, M. Kaiser, L. Stoutenborough, Q. Li, and I. Solti. "Evaluating the impact of pre-annotation on annotation speed and potential bias: natural language processing gold standard development for clinical named entity recognition in clinical trial announcements." In: *Journal of the American Medical Informatics Association* 21.3 (2013), pp. 406–413.

[39] X. Ma and E. Hovy. "End-to-end sequence labeling via bi-directional lstm-cnns-crf." In: *arXiv preprint arXiv:1603.01354* (2016).

[40] A. McCallum and W. Li. "Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons." In: *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*. Association for Computational Linguistics. 2003, pp. 188–191.

[41] R. Merchant, M. E. Okurowski, and N. Chinchor. *The multilingual entity task (MET) overview*. Tech. rep. DEPARTMENT OF DEFENSE FORT GEORGE G MEADE MD, 1996.

[42] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. "Distributed representations of words and phrases and their compositionality." In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[43] T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Efficient estimation of word representations in vector space." In: *arXiv preprint arXiv:1301.3781* (2013).

[44] D. L. Moody and M. A. Kortink. "From enterprise models to dimensional models: a methodology for data warehouse and data mart design." In: *DMDW*. 2000, p. 5.

[45] D. Nadeau and S. Sekine. "A survey of named entity recognition and classification." In: *Lingvisticae Investigationes* 30.1 (2007), pp. 3–26.

[46] D. Oleson, A. Sorokin, G. Laughlin, V. Hester, J. Le, and L. Biewald. "Programmatic gold: Targeted and scalable quality assurance in crowdsourcing." In: *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*. 2011.

[47] J. Oosterman, A. Bozzon, G.-J. Houben, A. Nottamkandath, C. Dijkshoorn, L. Aroyo, M. H. Leyssen, and M. C. Traub. "Crowd vs. experts: nichesourcing for knowledge intensive tasks in cultural heritage." In: *Proceedings of the 23rd International Conference on World Wide Web*. ACM. 2014, pp. 567–568.

[48] S. J. Pan and Q. Yang. "A survey on transfer learning." In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.

[49] R. Pascanu, T. Mikolov, and Y. Bengio. "Understanding the exploding gradient problem." In: *CoRR, abs/1211.5063* 2 (2012).

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

[51] S. Pradhan, A. Moschitti, N. Xue, H. T. Ng, A. Björkelund, O. Uryupina, Y. Zhang, and Z. Zhong. "Towards robust linguistic analysis using OntoNotes." In: *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*. 2013, pp. 143–152.

[52] A. Ritter, S. Clark, O. Etzioni, et al. "Named entity recognition in tweets: an experimental study." In: *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics. 2011, pp. 1524–1534.

[53] E. F. Sang. "Introduction to the CoNLL-2002 Shared Task: Language-Independent Named Entity Recognition." In: *arXiv preprint cs/0209010* (2002).

[54] E. F. Sang and S. Buchholz. "Introduction to the CoNLL-2000 shared task: Chunking." In: *arXiv preprint cs/0009008* (2000).

[55] E. F. Sang and F. De Meulder. "Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition." In: *arXiv preprint cs/0306050* (2003).

[56] B. Santorini. "Part-of-speech tagging guidelines for the Penn Treebank Project (3rd revision)." In: *Technical Reports (CIS)* (1990), p. 570.

[57] S. Sekine and H. Isahara. "IREX: IR & IE Evaluation Project in Japanese." In: *LREC*. Citeseer. 2000, pp. 1977–1980.

[58] S. Sekine, R. Grishman, and H. Shinnou. "A decision tree method for finding and classifying names in Japanese texts." In: *Sixth Workshop on Very Large Corpora*. 1998.

[59] R. Snow, B. O'Connor, D. Jurafsky, and A. Y. Ng. "Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks." In: *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics. 2008, pp. 254–263.

[60] C. Sun, N. Rampalli, F. Yang, and A. Doan. "Chimera: Large-scale classification using machine learning, rules, and crowdsourcing." In: *Proceedings of the VLDB Endowment* 7.13 (2014), pp. 1529–1540.

[61] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. "On the importance of initialization and momentum in deep learning." In: *International conference on machine learning*. 2013, pp. 1139–1147.

[62] Q. Tran, A. MacKinlay, and A. J. Yepes. "Named entity recognition with stack residual lstm and trainable bias decoding." In: *arXiv preprint arXiv:1706.07598* (2017).

[63] Z. Yang, R. Salakhutdinov, and W. Cohen. "Multi-task cross-lingual sequence tagging from scratch." In: *arXiv preprint arXiv:1603.06270* (2016).

[64] Z. Yang, R. Salakhutdinov, and W. W. Cohen. "Transfer learning for sequence tagging with hierarchical recurrent networks." In: *arXiv preprint arXiv:1703.06345* (2017).

Appendix A

Report on Framework Architecture

To execute the proposed work of this thesis, we need a framework that supports the development of neural networks; such includes the selection of dataset, embeddings, and learning setting, as well as the definition of the layers of the neural networks. Furthermore, the framework needs to support different types of metrics that can be computed during the testing phase. Having this in mind, we propose an architecture to implement the mentioned framework. This architecture is divided into six components:

- Dataset;
- Embeddings;
- Dataset processor;
- Deep learning system;
- Learning setting;
- Model Evaluator.

The dataset component is where the data classes are defined. These classe possess metadata, statistics, and the training, validation and test sets. The prime function of these classes is reading the data sources and transform the data into a format that can be used by the dataset processor. By convention, the format is a list of matrices, meaning that for each sentence, there is an array consisting of the features and the respective label.

The embeddings component is where the embeddings are defined. We retrieve the embeddings by using the Flair package [3]. We support all the embeddings available in the package and the different dimensions (50, 100, 200, 300) of the GloVe embeddings.

The dataset processor is responsible for embedding the dataset and pre-process it in such a way that it can be sent into the deep learning system. This system is implement by using the Keras library [12] with the Tensorflow backend.

The deep learning system component is where the neural networks are defined. To add a new architecture, we need to extend the *DeepLearningSystem* class and redefine four methods (char-level builder, word-level builder, classification builder, and optimizer builder).

The learning setting component is where the training method is defined. The component allows to specify the training parameters, such as batch size, epochs, among others. Furthermore, this component is the link between the dataset, deep learning system, and the model evaluator. The learning setting encapsulates the logic of training a specific neural network in a specific dataset under a certain training process. It is also possible to retrieve a specific list of metrics during the testing phase.

Lastly, there is the model evaluator component. This component is where the metrics that evaluate the model's performance are defined.

Appendix B

Model Adaptation

Training Rate (%)	IDD	IDD + ODD
1	0.60	0.74
3	0.74	0.78
5	0.80	0.79
10	0.80	0.82
20	0.83	0.83
30	0.83	0.84
40	0.84	0.84
50	0.85	0.84
60	0.84	0.85
70	0.85	0.85
80	0.86	0.85
90	0.86	0.85
100	0.86	0.86

Table B.1: The values of the MUC score for CD-1 for the different training rates when training the models with in-domain data and in-domain data in conjunction with out-domain data.

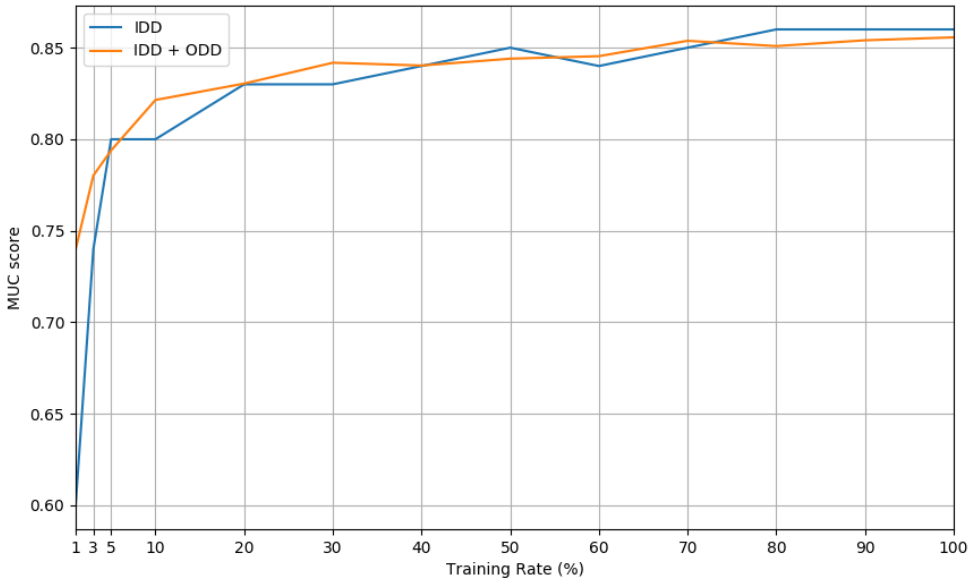


Figure B.1: The values of the MUC score for CD-1 for the different training rates when training the models with in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are presented in table B.1.

Training Rate (%)	IDD	IDD + ODD
1	0.05	0.12
3	0.33	0.38
5	0.64	0.69
10	0.82	0.81
20	0.84	0.85
30	0.85	0.86
40	0.85	0.85
50	0.87	0.86
60	0.86	0.88
70	0.87	0.89
80	0.88	0.88
90	0.87	0.89
100	0.89	0.89

Table B.2: The values of the MUC score for CD-2 for the different training rates when training the models with in-domain data and in-domain data in conjunction with out-domain data.

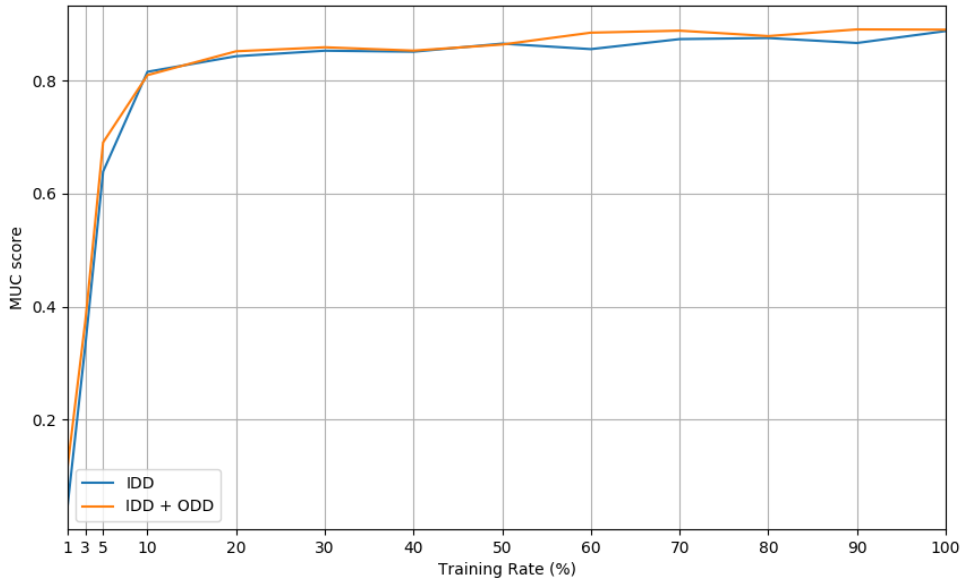


Figure B.2: The values of the MUC score for CD-2 for the different training rates when training the models with in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are presented in table B.2.

Training Rate (%)	IDD	IDD + ODD
1	0.46	0.65
3	0.72	0.79
5	0.77	0.81
10	0.82	0.84
20	0.86	0.84
30	0.85	0.86
40	0.87	0.87
50	0.88	0.88
60	0.88	0.89
70	0.88	0.88
80	0.89	0.88
90	0.89	0.89
100	0.90	0.89

Table B.3: The values of the MUC score for CD-3 for the different training rates when training the models with in-domain data and in-domain data in conjunction with out-domain data.

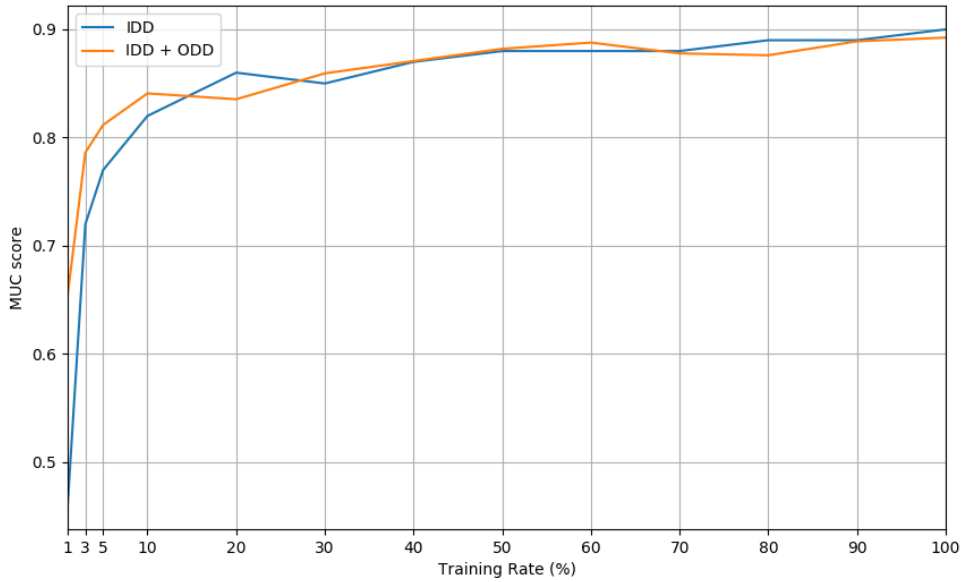


Figure B.3: The values of the MUC score for CD-3 for the different training rates when training the models with in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are presented in table B.3.

Training Rate (%)	IDD	IDD + ODD
1	0.85	0.88
3	0.91	0.93
5	0.94	0.95
10	0.95	0.95
20	0.97	0.97
30	0.98	0.97
40	0.98	0.98
50	0.99	0.98
60	0.98	0.98
70	0.99	0.98
80	1.00	0.98
90	0.99	0.98
100	0.99	0.98

Table B.4: The values of the MUC score for CD-4 for the different training rates when training the models with in-domain data and in-domain data in conjunction with out-domain data.

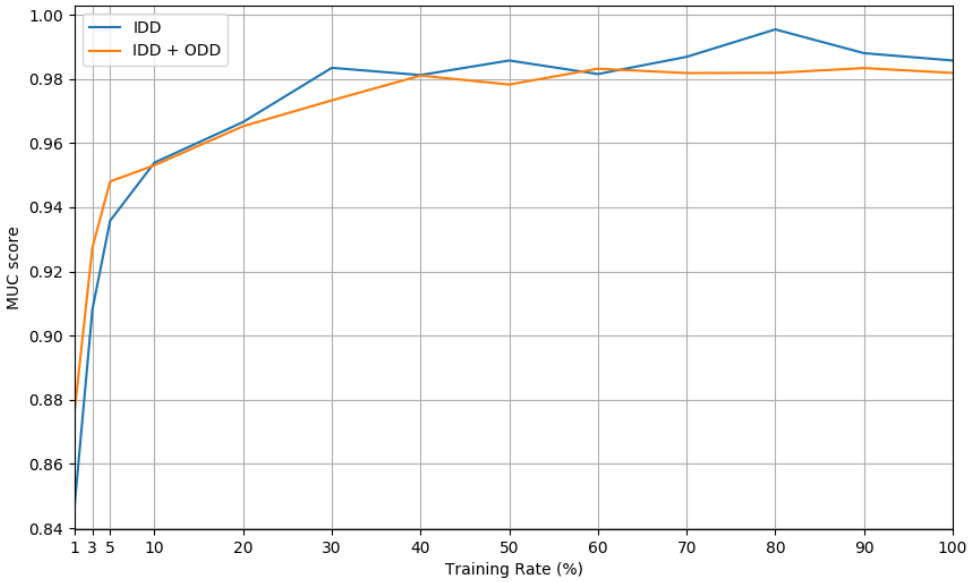


Figure B.4: The values of the MUC score for CD-4 for the different training rates when training the models with in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are presented in table B.4.

Training Rate (%)	IDD	IDD + ODD
1	0.69	0.76
3	0.85	0.85
5	0.88	0.87
10	0.91	0.91
20	0.93	0.93
30	0.94	0.93
40	0.94	0.94
50	0.95	0.95
60	0.95	0.95
70	0.95	0.95
80	0.96	0.95
90	0.95	0.95
100	0.95	0.95

Table B.5: The values of the MUC score for CD-5 for the different training rates when training the models with in-domain data and in-domain data in conjunction with out-domain data.

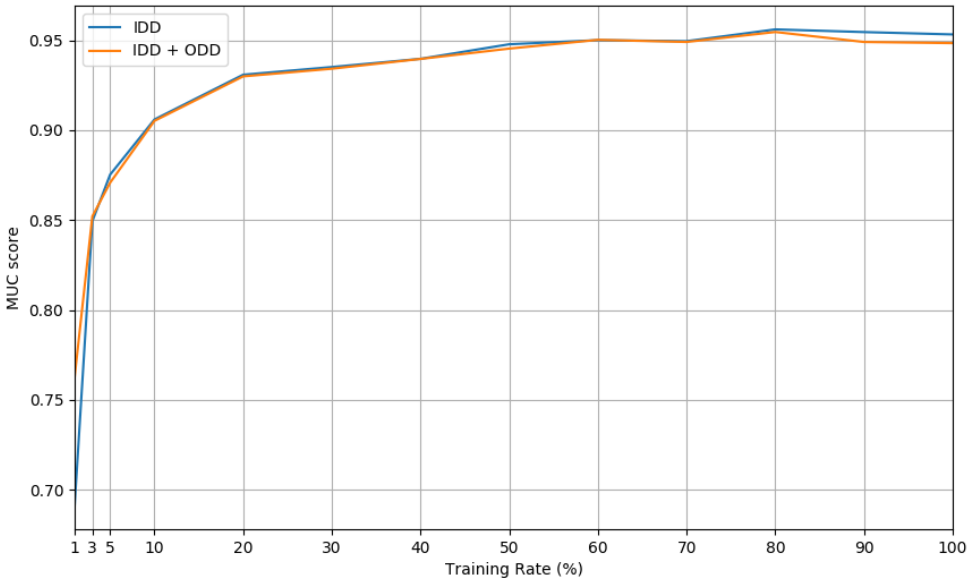


Figure B.5: The values of the MUC score for CD-5 for the different training rates when training the models with in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are presented in table B.5.

Training Rate (%)	IDD	IDD + ODD
1	0.45	0.45
3	0.64	0.63
5	0.67	0.66
10	0.71	0.69
20	0.76	0.73
30	0.75	0.75
40	0.78	0.77
50	0.78	0.77
60	0.79	0.79
70	0.80	0.79
80	0.80	0.78
90	0.81	0.77
100	0.81	0.79

Table B.6: The values of the MUC score for CD-6 for the different training rates when training the models with in-domain data and in-domain data in conjunction with out-domain data.

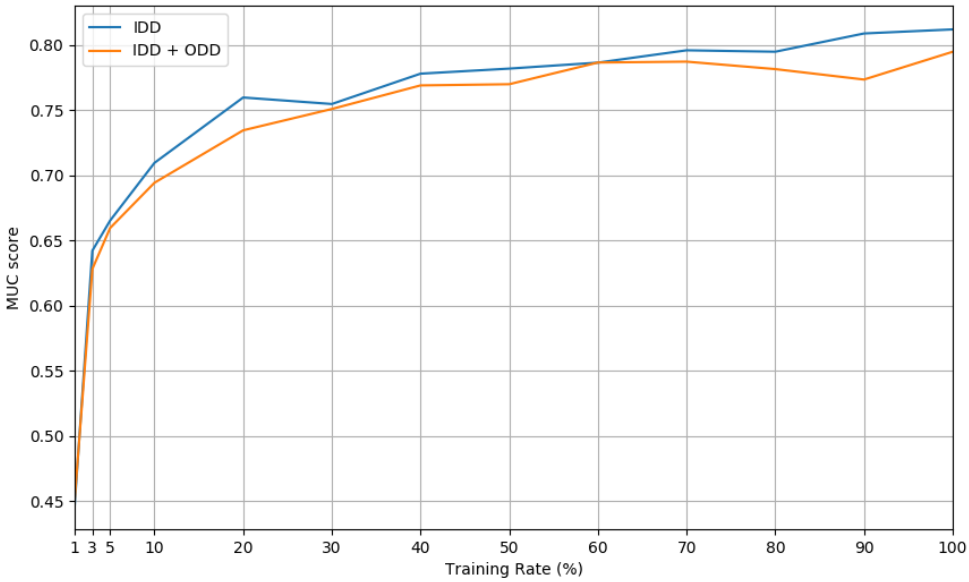


Figure B.6: The values of the MUC score for CD-6 for the different training rates when training the models with in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are presented in table B.6.

Training Rate (%)	IDD	IDD + ODD
1	0.29	0.28
3	0.48	0.54
5	0.65	0.62
10	0.67	0.69
20	0.70	0.70
30	0.71	0.71
40	0.77	0.73
50	0.77	0.74
60	0.74	0.74
70	0.77	0.76
80	0.76	0.75
90	0.75	0.78
100	0.75	0.78

Table B.7: The values of the MUC score for CD-7 for the different training rates when training the models with in-domain data and in-domain data in conjunction with out-domain data.

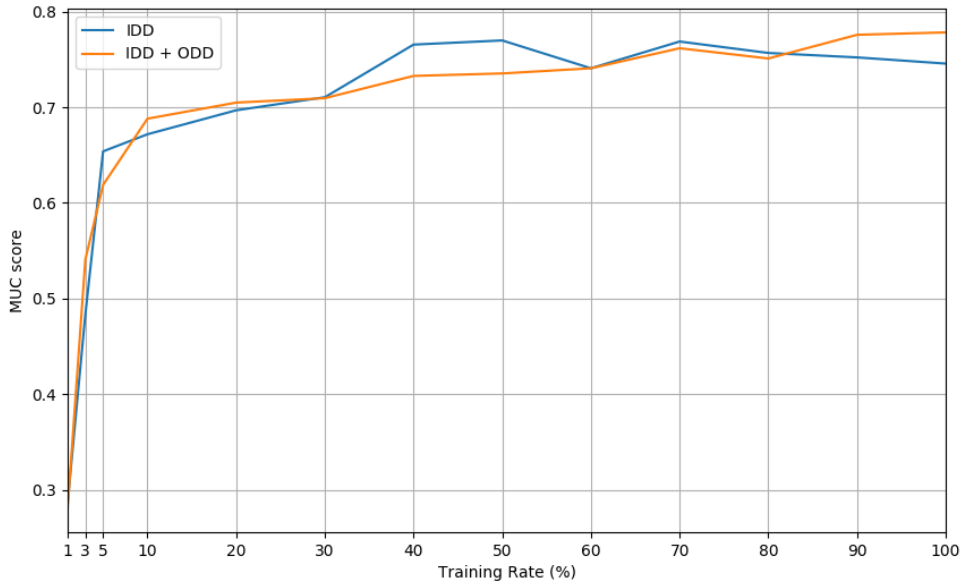


Figure B.7: The values of the MUC score for CD-7 for the different training rates when training the models with in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are presented in table B.7.

Training Rate (%)	IDD	IDD + ODD
1	0.33	0.33
3	0.47	0.48
5	0.57	0.59
10	0.59	0.59
20	0.60	0.63
30	0.66	0.70
40	0.66	0.68
50	0.65	0.64
60	0.66	0.65
70	0.67	0.70
80	0.72	0.69
90	0.72	0.74
100	0.69	0.70

Table B.8: The values of the MUC score for CD-8 for the different training rates when training the models with in-domain data and in-domain data in conjunction with out-domain data.

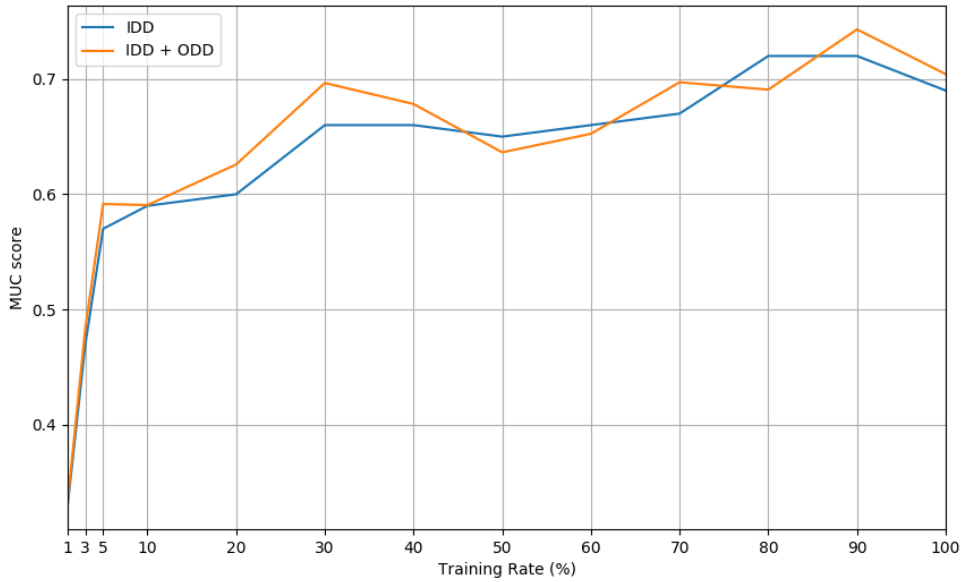


Figure B.8: The values of the MUC score for CD-8 for the different training rates when training the models with in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are presented in table B.8.

Training Rate (%)	IDD	IDD + ODD
1	0.45	0.48
3	0.58	0.61
5	0.65	0.68
10	0.74	0.73
20	0.77	0.76
30	0.77	0.77
40	0.74	0.78
50	0.78	0.77
60	0.78	0.79
70	0.78	0.80
80	0.79	0.78
90	0.78	0.78
100	0.81	0.78

Table B.9: The values of the MUC score for CD-9 for the different training rates when training the models with in-domain data and in-domain data in conjunction with out-domain data.

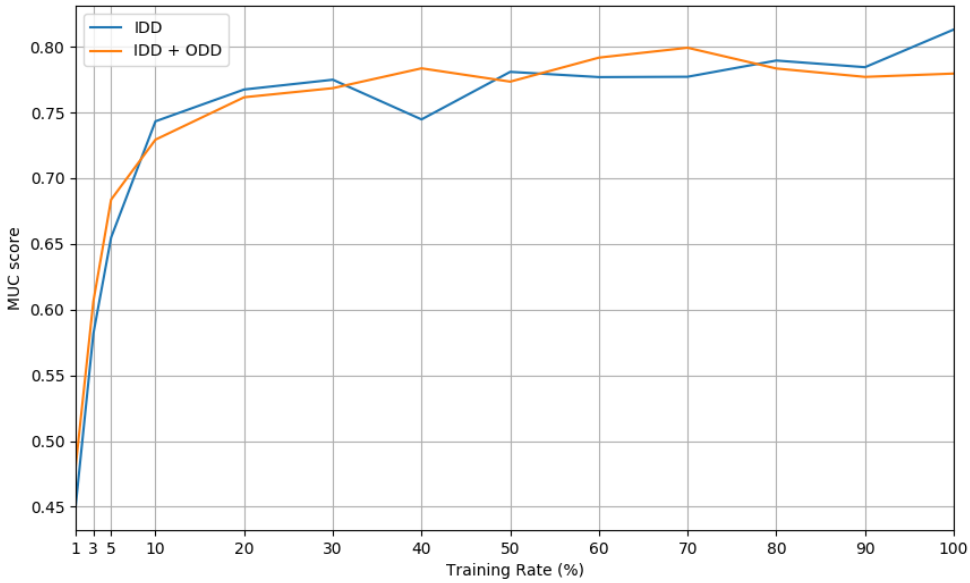


Figure B.9: The values of the MUC score for CD-9 for the different training rates when training the models with in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are presented in table B.9.

Appendix C

Assessment of Crowd Effort Savings

Training Rate (%)	Generation Effort	Validation Effort		Total Effort	
		IDD	IDD + ODD	IDD	IDD + ODD
1	01:57	228:35	220:41	230:32	222:38
3	08:01	220:22	219:23	228:24	227:25
5	11:13	207:52	211:47	219:06	223:01
10	23:54	198:07	195:33	222:01	219:28
20	48:55	174:11	171:31	223:06	220:26
30	74:26	153:08	151:42	227:34	226:09
40	97:49	129:28	133:02	227:17	230:51
50	125:34	103:35	110:18	229:10	235:53
60	144:10	85:13	87:18	229:24	231:28
70	175:33	67:36	62:42	243:10	238:15
80	196:33	41:41	45:05	238:15	241:39
90	219:03	20:48	23:09	239:52	242:12

Table C.1: The values of generation, validation, and total effort (in minutes) for CD-1 for each distinct training rate value. The validation and total effort are associated to models trained with only in-domain data (blue line) and in-domain data in-domain data in conjunction with out-domain data (orange line). Note that the format of the effort is {hours:}:{minutes}.

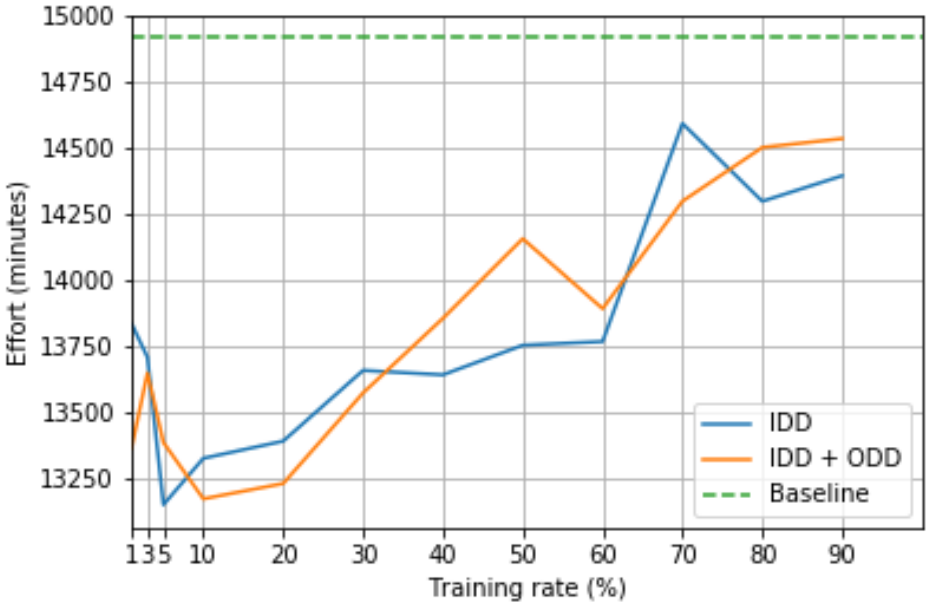


Figure C.1: A line plot with the total effort curves, in minutes, when training the model with in-domain data (blue line) and in conjunction with out-domain data (orange line) when compared against the baseline effort (green dashed line) for the different amount of training rate. These effort values are associated with the training rates of CD-1 (see table C.1).

Training Rate (%)	Generation Effort	Validation Effort		Total Effort	
		IDD	IDD + ODD	IDD	IDD + ODD
1	00:04	10:04	10:01	10:09	10:06
3	00:14	09:20	09:14	09:34	09:28
5	00:27	08:21	08:15	08:48	08:42
10	01:01	07:10	07:05	08:11	08:06
20	02:00	06:15	06:12	08:15	08:12
30	03:02	05:20	05:22	08:22	08:25
40	04:04	04:31	04:35	08:35	08:39
50	05:10	03:45	03:50	08:56	09:01
60	06:08	03:02	03:02	09:11	09:11
70	07:10	02:16	02:18	09:26	09:28
80	08:10	01:33	01:34	09:44	09:45
90	09:12	00:47	00:47	09:59	09:59

Table C.2: The values of generation, validation, and total effort (in minutes) for CD-2 for each distinct training rate value. The validation and total effort are associated to models trained with only in-domain data (blue line) and in-domain data in-domain data in conjunction with out-domain data (orange line). Note that the format of the effort is {hours:}:{minutes}.

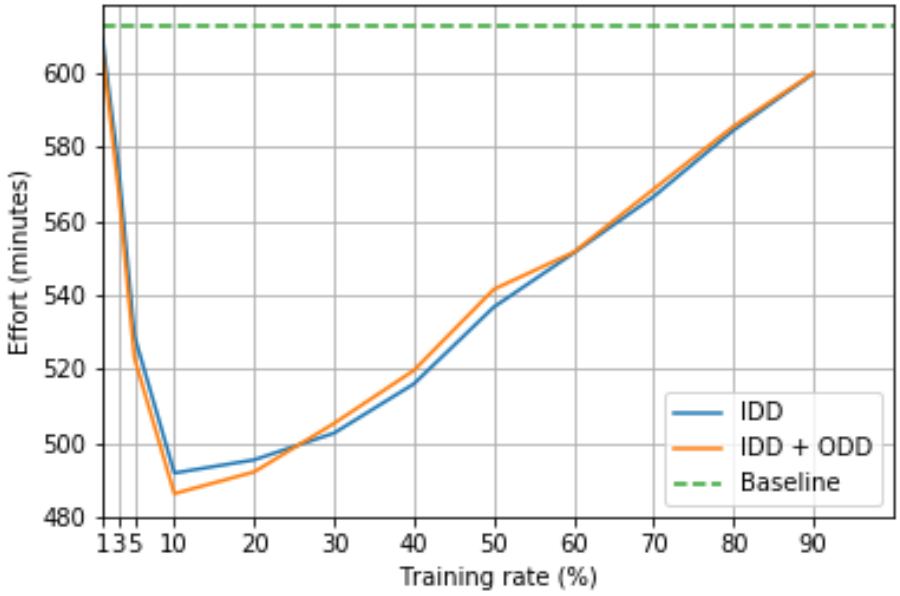


Figure C.2: A line plot with the total effort curves, in minutes, when training the model with in-domain data (blue line) and in conjunction with out-domain data (orange line) when compared against the baseline effort (green dashed line) for the different amount of training rate. These effort values are associated with the training rates of CD-2 (see table C.2).

Training Rate (%)	Generation Effort	Validation Effort		Total Effort	
		IDD	IDD + ODD	IDD	IDD + ODD
1	01:44	121:44	117:22	123:29	119:07
3	03:38	114:19	109:28	117:58	113:06
5	06:50	110:42	105:06	117:32	111:56
10	13:48	97:31	98:27	111:19	112:15
20	27:20	85:43	87:07	113:04	114:27
30	39:17	75:25	77:03	114:43	116:21
40	55:30	62:42	64:29	118:12	119:59
50	68:52	52:48	51:22	121:41	120:14
60	78:20	42:15	43:37	120:35	121:58
70	92:13	31:07	33:20	123:21	125:34
80	102:20	19:44	21:47	122:04	124:07
90	121:03	10:41	09:47	131:45	130:51

Table C.3: The values of generation, validation, and total effort (in minutes) for CD-3 for each distinct training rate value. The validation and total effort are associated to models trained with only in-domain data (blue line) and in-domain data in-domain data in conjunction with out-domain data (orange line). Note that the format of the effort is {hours:}:{minutes}.

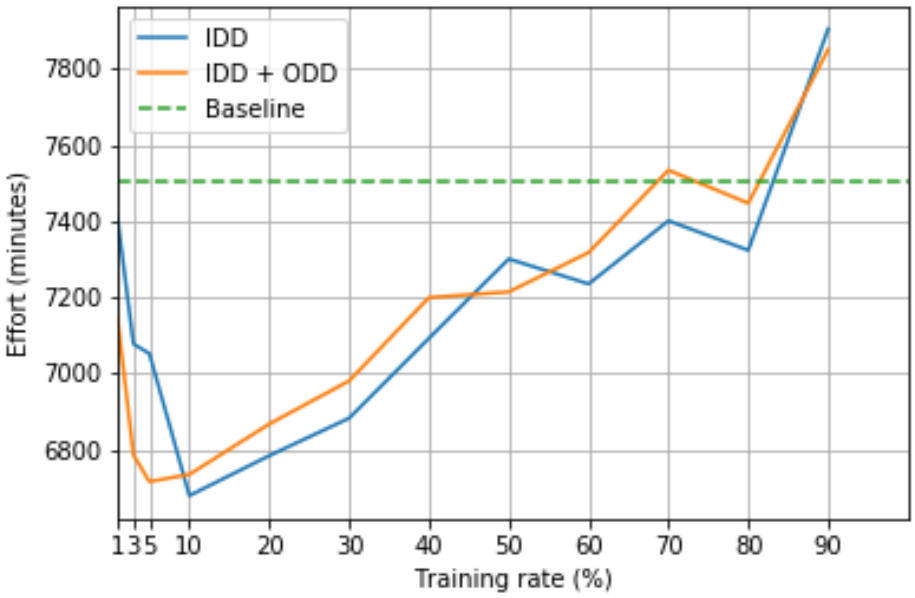


Figure C.3: A line plot with the total effort curves, in minutes, when training the model with in-domain data (blue line) and in conjunction with out-domain data (orange line) when compared against the baseline effort (green dashed line) for the different amount of training rate. These effort values are associated with the training rates of CD-3 (see table C.3).

Training Rate (%)	Generation Effort	Validation Effort		Total Effort	
		IDD	IDD + ODD	IDD	IDD + ODD
1	00:09	07:14	06:48	07:23	06:57
3	00:23	06:23	06:19	06:46	06:42
5	00:41	05:47	05:47	06:29	06:29
10	01:22	05:20	05:20	06:43	06:42
20	02:42	04:37	04:38	07:20	07:21
30	04:06	03:57	03:59	08:04	08:05
40	05:30	03:20	03:23	08:50	08:53
50	06:50	02:50	02:49	09:40	09:40
60	08:13	02:13	02:13	10:26	10:27
70	09:32	01:38	01:38	11:11	11:11
80	10:56	01:05	01:05	12:01	12:01
90	12:24	00:33	00:33	12:57	12:57

Table C.4: The values of generation, validation, and total effort (in minutes) for CD-4 for each distinct training rate value. The validation and total effort are associated to models trained with only in-domain data (blue line) and in-domain data in-domain data in conjunction with out-domain data (orange line). Note that the format of the effort is {hours:}:{minutes}.

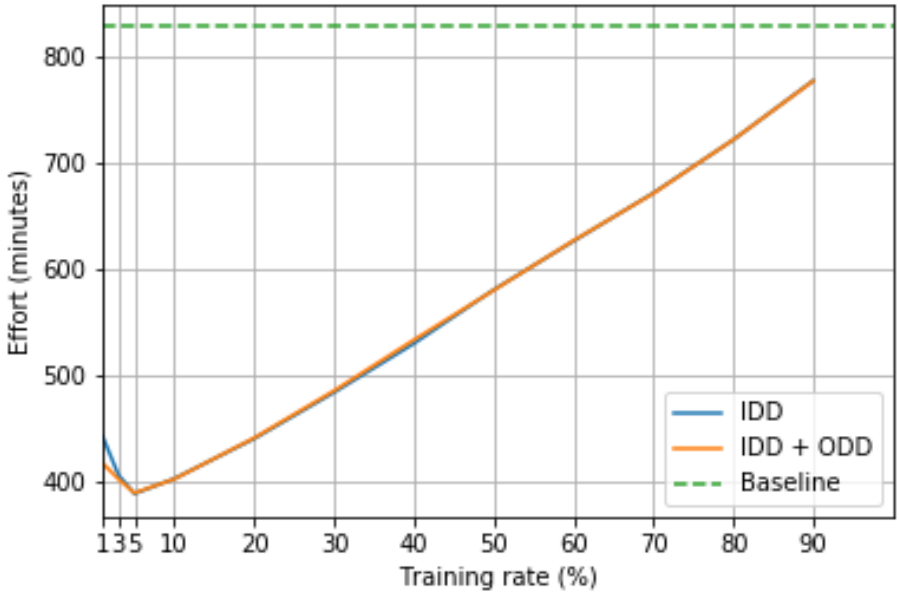


Figure C.4: A line plot with the total effort curves, in minutes, when training the model with in-domain data (blue line) and in conjunction with out-domain data (orange line) when compared against the baseline effort (green dashed line) for the different amount of training rate. These effort values are associated with the training rates of CD-4 (see table C.4).

Training Rate (%)	Generation Effort	Validation Effort		Total Effort	
		IDD	IDD + ODD	IDD	IDD + ODD
1	00:21	24:14	22:23	24:36	22:44
3	01:07	18:51	18:34	19:58	19:41
5	01:51	17:33	17:34	19:24	19:25
10	03:37	15:54	15:51	19:31	19:28
20	07:18	13:10	13:26	20:29	20:45
30	11:03	11:28	11:37	22:31	22:40
40	14:28	09:51	09:53	24:19	24:21
50	17:58	08:10	08:11	26:09	26:09
60	21:50	06:22	06:37	28:13	28:27
70	25:33	04:46	04:46	30:19	30:19
80	29:06	03:06	03:11	32:13	32:18
90	32:41	01:35	01:35	34:16	34:16

Table C.5: The values of generation, validation, and total effort (in minutes) for CD-5 for each distinct training rate value. The validation and total effort are associated to models trained with only in-domain data (blue line) and in-domain data in-domain data in conjunction with out-domain data (orange line). Note that the format of the effort is {hours:}:{minutes}.

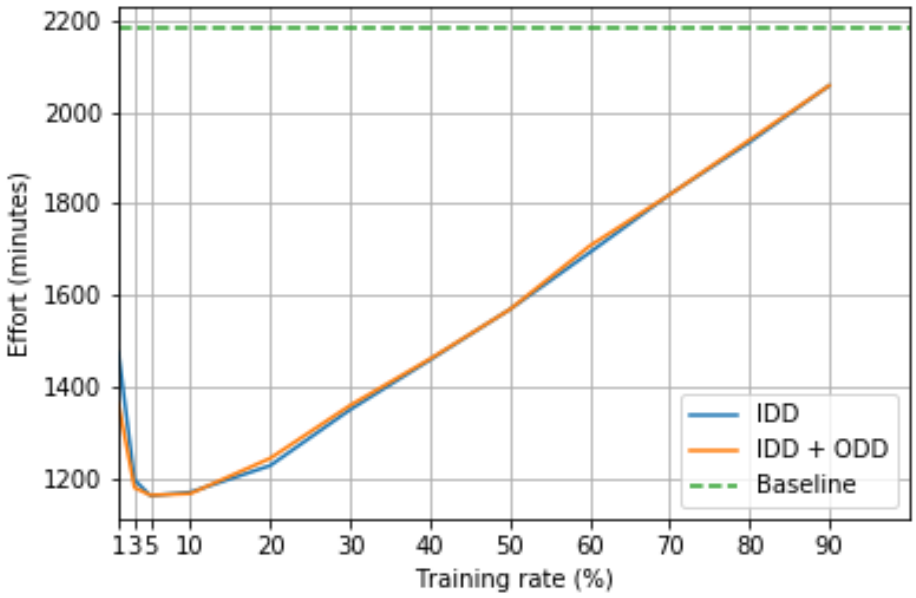


Figure C.5: A line plot with the total effort curves, in minutes, when training the model with in-domain data (blue line) and in conjunction with out-domain data (orange line) when compared against the baseline effort (green dashed line) for the different amount of training rate. These effort values are associated with the training rates of CD-5 (see table C.5).

Training Rate (%)	Generation Effort	Validation Effort		Total Effort	
		IDD	IDD + ODD	IDD	IDD + ODD
1	00:22	31:33	30:39	31:56	31:02
3	01:10	28:13	27:59	29:23	29:09
5	01:55	25:43	26:17	27:38	28:12
10	03:53	22:59	23:10	26:53	27:04
20	07:42	18:48	19:27	26:31	27:09
30	11:23	16:23	16:19	27:46	27:42
40	15:23	13:40	14:09	29:03	29:32
50	19:24	11:26	11:37	30:51	31:02
60	23:15	08:54	09:06	32:09	32:21
70	27:07	06:43	06:51	33:50	33:58
80	31:08	04:27	04:36	35:36	35:45
90	35:00	02:15	02:15	37:16	37:15

Table C.6: The values of generation, validation, and total effort (in minutes) for CD-6 for each distinct training rate value. The validation and total effort are associated to models trained with only in-domain data (blue line) and in-domain data in-domain data in conjunction with out-domain data (orange line). Note that the format of the effort is {hours:}:{minutes}.

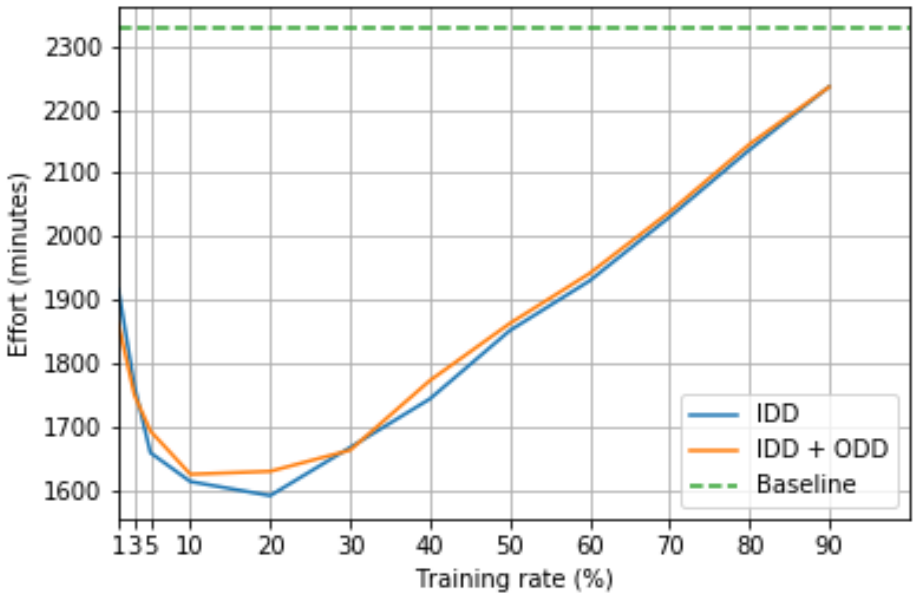


Figure C.6: A line plot with the total effort curves, in minutes, when training the model with in-domain data (blue line) and in conjunction with out-domain data (orange line) when compared against the baseline effort (green dashed line) for the different amount of training rate. These effort values are associated with the training rates of CD-6 (see table C.6).

Training Rate (%)	Generation Effort	Validation Effort		Total Effort	
		IDD	IDD + ODD	IDD	IDD + ODD
1	00:09	11:42	11:27	11:52	11:36
3	00:21	10:41	10:23	11:02	10:45
5	00:40	09:34	09:54	10:14	10:34
10	01:09	09:19	09:00	10:28	10:09
20	02:22	07:50	07:54	10:12	10:17
30	03:32	06:48	06:50	10:20	10:22
40	04:42	05:31	05:37	10:13	10:19
50	05:59	04:24	04:25	10:24	10:25
60	06:59	03:24	03:30	10:24	10:29
70	08:18	02:40	02:41	10:59	11:00
80	09:15	01:50	01:43	11:06	10:58
90	10:17	00:46	00:52	11:04	11:10

Table C.7: The values of generation, validation, and total effort (in minutes) for CD-7 for each distinct training rate value. The validation and total effort are associated to models trained with only in-domain data (blue line) and in-domain data in-domain data in conjunction with out-domain data (orange line). Note that the format of the effort is {hours:}:{minutes}.

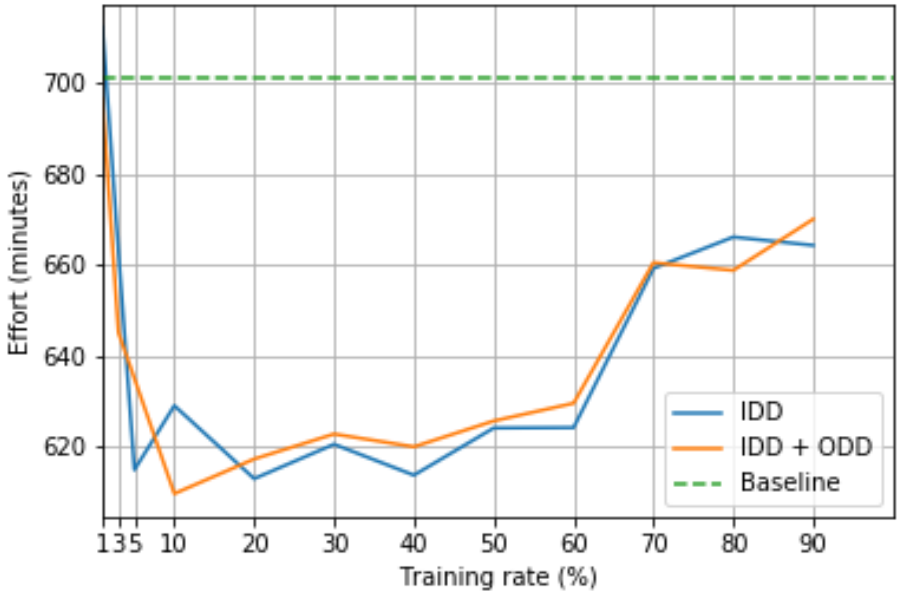


Figure C.7: A line plot with the total effort curves, in minutes, when training the model with in-domain data (blue line) and in conjunction with out-domain data (orange line) when compared against the baseline effort (green dashed line) for the different amount of training rate. These effort values are associated with the training rates of CD-7 (see table C.7).

Training Rate (%)	Generation Effort	Validation Effort		Total Effort	
		IDD	IDD + ODD	IDD	IDD + ODD
1	00:06	07:55	07:48	08:01	07:54
3	00:14	07:23	07:58	07:37	08:12
5	00:26	07:39	07:26	08:05	07:52
10	00:52	07:12	07:08	08:04	08:00
20	01:37	06:03	05:53	07:40	07:31
30	02:28	04:51	05:20	07:20	07:49
40	03:14	04:44	04:13	07:58	07:27
50	03:58	03:55	03:52	07:53	07:50
60	04:42	03:02	03:04	07:44	07:47
70	05:35	02:12	02:09	07:48	07:44
80	06:21	01:18	01:28	07:39	07:49
90	07:35	00:38	00:39	08:13	08:14

Table C.8: The values of generation, validation, and total effort (in minutes) for CD-8 for each distinct training rate value. The validation and total effort are associated to models trained with only in-domain data (blue line) and in-domain data in-domain data in conjunction with out-domain data (orange line). Note that the format of the effort is {hours:}:{minutes}.

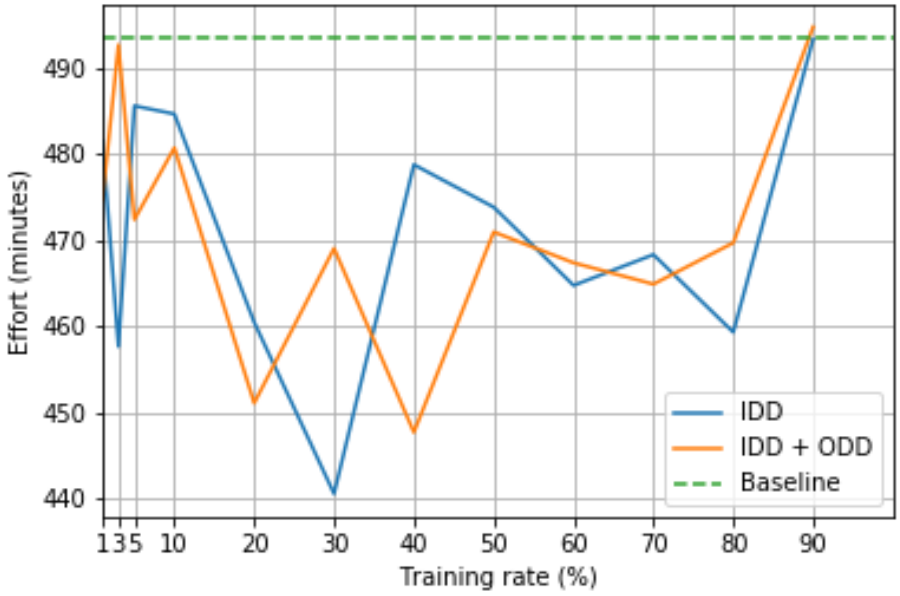


Figure C.8: A line plot with the total effort curves, in minutes, when training the model with in-domain data (blue line) and in conjunction with out-domain data (orange line) when compared against the baseline effort (green dashed line) for the different amount of training rate. These effort values are associated with the training rates of CD-8 (see table C.8).

Training Rate (%)	Generation Effort	Validation Effort		Total Effort	
		IDD	IDD + ODD	IDD	IDD + ODD
1	00:08	14:09	14:22	14:18	14:31
3	00:25	12:09	12:27	12:35	12:52
5	00:44	11:35	11:30	12:20	12:14
10	01:31	10:09	10:20	11:40	11:52
20	03:16	08:40	09:00	11:56	12:16
30	04:49	07:16	07:35	12:06	12:24
40	06:20	07:08	06:31	13:29	12:51
50	07:56	05:22	05:23	13:19	13:19
60	09:43	04:06	04:11	13:50	13:55
70	11:19	03:01	03:05	14:21	14:25
80	12:50	01:57	02:03	14:48	14:54
90	14:19	00:59	01:04	15:18	15:23

Table C.9: The values of generation, validation, and total effort (in minutes) for CD-9 for each distinct training rate value. The validation and total effort are associated to models trained with only in-domain data (blue line) and in-domain data in-domain data in conjunction with out-domain data (orange line). Note that the format of the effort is {hours:}:{minutes}.

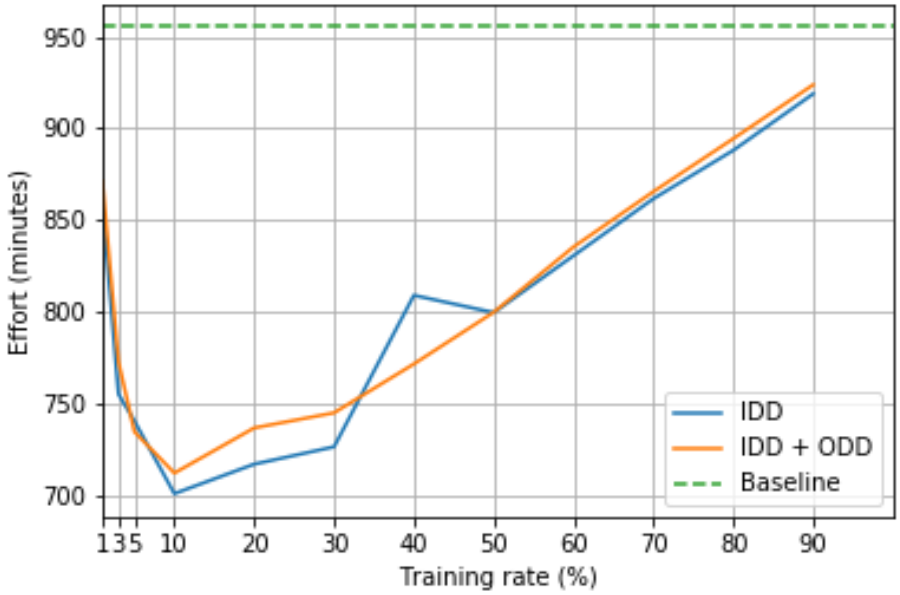


Figure C.9: A line plot with the total effort curves, in minutes, when training the model with in-domain data (blue line) and in conjunction with out-domain data (orange line) when compared against the baseline effort (green dashed line) for the different amount of training rate. These effort values are associated with the training rates of CD-9 (see table C.9).

Training Rate (%)	IDD	IDD + ODD
1	0.05	0.18
3	0.21	0.30
5	0.30	0.31
10	0.36	0.43
20	0.40	0.43
30	0.45	0.47
40	0.50	0.48
50	0.51	0.49
60	0.43	0.48
70	0.51	0.53
80	0.53	0.54
90	0.57	0.58

Table C.10: The values of the index of saved actions when training the model with only in-domain data and in-domain data in conjunction with out-domain data for the different training rates. These values are associated with CD-1.

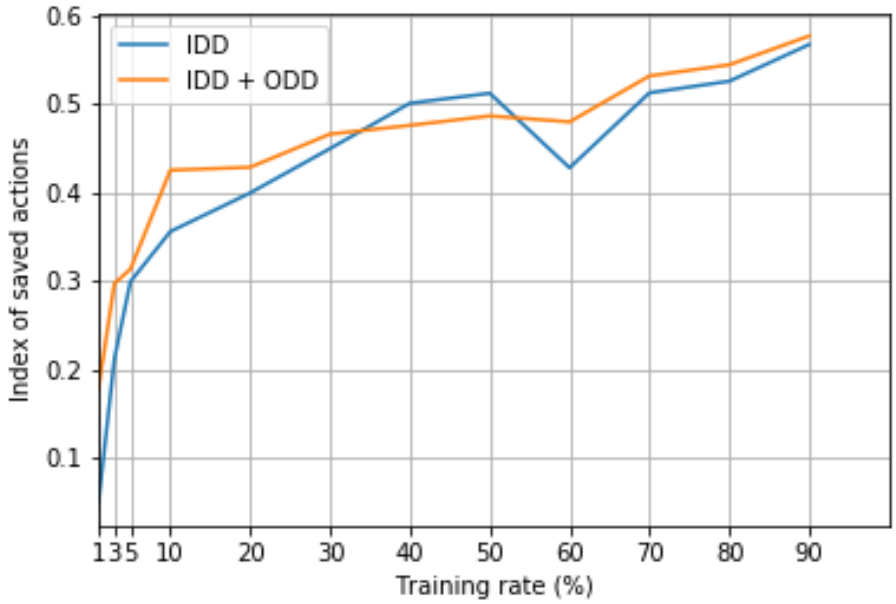


Figure C.10: The curves of the index of saved actions when training the model with only in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are associated with the training rates of CD-1 (see table C.10).

Training Rate (%)	IDD	IDD + ODD
1	-0.92	-0.88
3	-0.57	-0.5
5	-0.02	0.04
10	0.39	0.46
20	0.49	0.54
30	0.57	0.56
40	0.58	0.58
50	0.61	0.54
60	0.60	0.60
70	0.63	0.58
80	0.57	0.51
90	0.51	0.61

Table C.11: The values of the index of saved actions when training the model with only in-domain data and in-domain data in conjunction with out-domain data for the different training rates. These values are associated with CD-2.

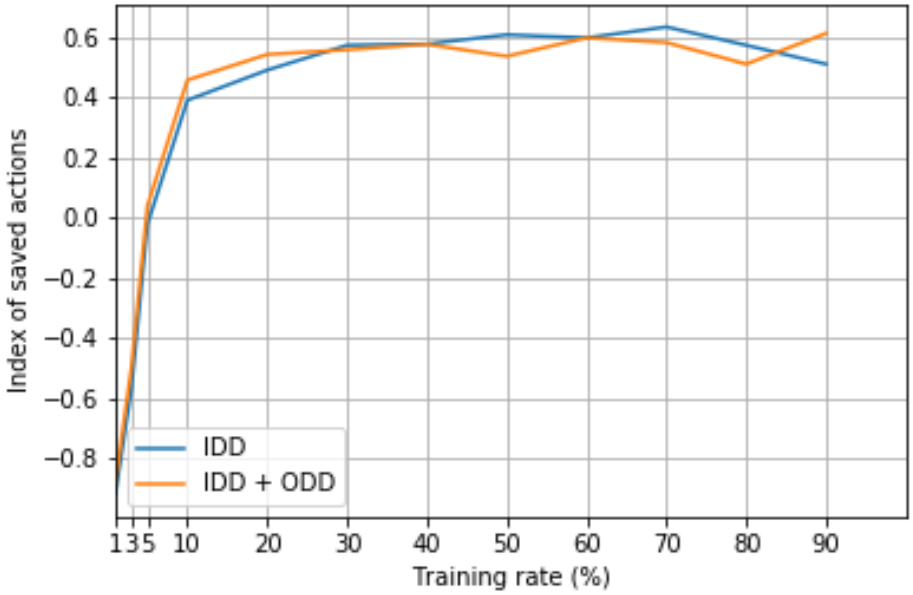


Figure C.11: The curves of the index of saved actions when training the model with only in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are associated with the training rates of CD-2 (see table C.11).

Training Rate (%)	IDD	IDD + ODD
1	-0.43	-0.01
3	0.11	0.33
5	0.29	0.39
10	0.47	0.52
20	0.56	0.56
30	0.61	0.62
40	0.65	0.63
50	0.71	0.68
60	0.67	0.67
70	0.69	0.66
80	0.70	0.64
90	0.62	0.64

Table C.12: The values of the index of saved actions when training the model with only in-domain data and in-domain data in conjunction with out-domain data for the different training rates. These values are associated with CD-3.

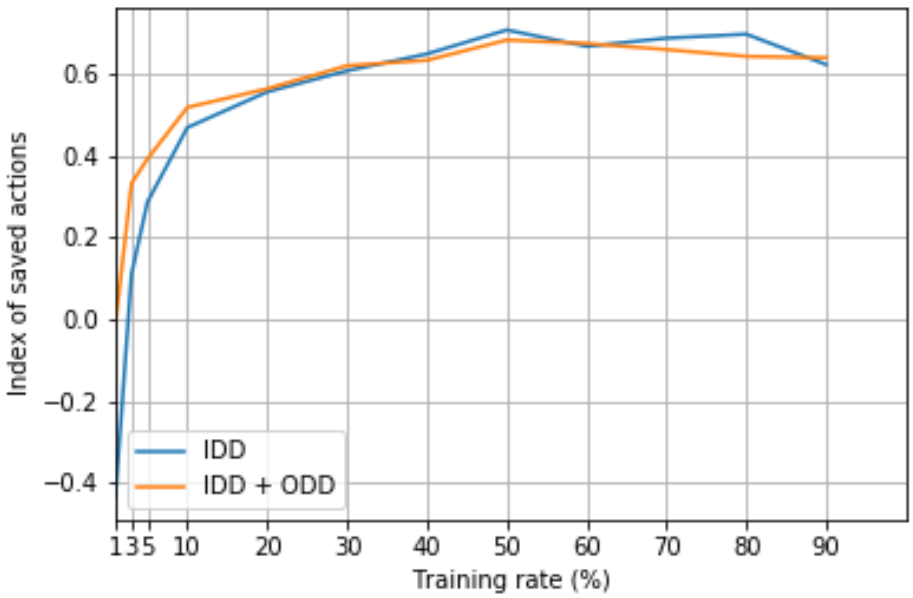


Figure C.12: The curves of the index of saved actions when training the model with only in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are associated with the training rates of CD-3 (see table C.12).

Training Rate (%)	IDD	IDD + ODD
1	0.51	0.60
3	0.68	0.69
5	0.78	0.78
10	0.81	0.81
20	0.86	0.85
30	0.88	0.87
40	0.90	0.89
50	0.88	0.89
60	0.92	0.90
70	0.92	0.92
80	0.93	0.92
90	0.92	0.91

Table C.13: The values of the index of saved actions when training the model with only in-domain data and in-domain data in conjunction with out-domain data for the different training rates. These values are associated with CD-4.

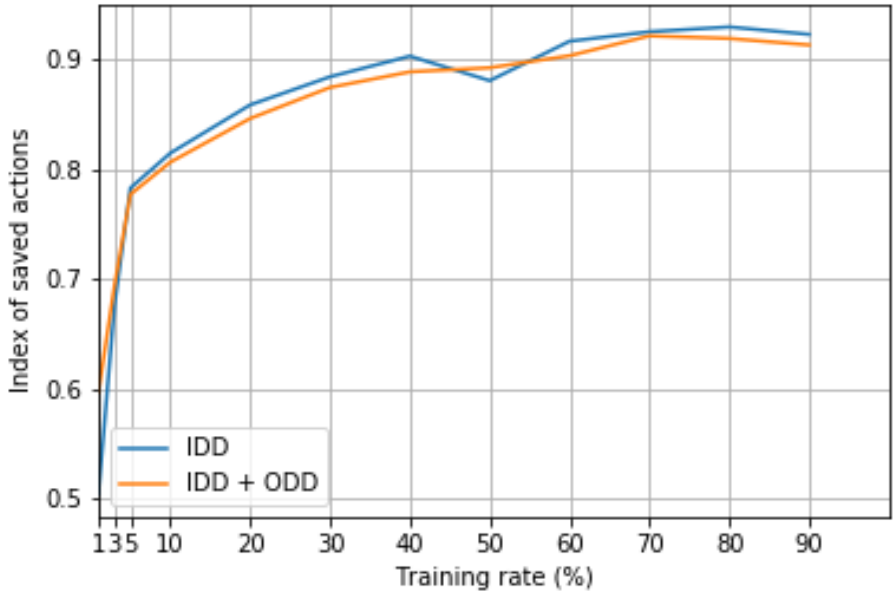


Figure C.13: The curves of the index of saved actions when training the model with only in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are associated with the training rates of CD-4 (see table C.13).

Training Rate (%)	IDD	IDD + ODD
1	0.14	0.27
3	0.52	0.55
5	0.60	0.59
10	0.66	0.67
20	0.76	0.73
30	0.76	0.75
40	0.77	0.76
50	0.78	0.77
60	0.81	0.77
70	0.80	0.79
80	0.80	0.78
90	0.81	0.79

Table C.14: The values of the index of saved actions when training the model with only in-domain data and in-domain data in conjunction with out-domain data for the different training rates. These values are associated with CD-5.

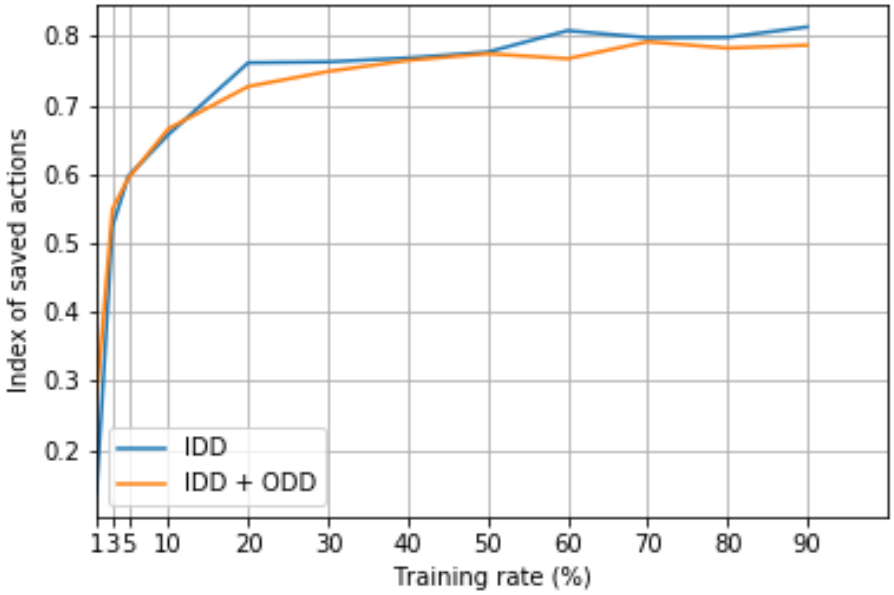


Figure C.14: The curves of the index of saved actions when training the model with only in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are associated with the training rates of CD-5 (see table C.14).

Training Rate (%)	IDD	IDD + ODD
1	-0.33	-0.29
3	-0.05	-0.03
5	0.08	0.06
10	0.19	0.18
20	0.33	0.29
30	0.33	0.34
40	0.39	0.34
50	0.37	0.36
60	0.40	0.37
70	0.39	0.37
80	0.41	0.36
90	0.42	0.40

Table C.15: The values of the index of saved actions when training the model with only in-domain data and in-domain data in conjunction with out-domain data for the different training rates. These values are associated with CD-6.

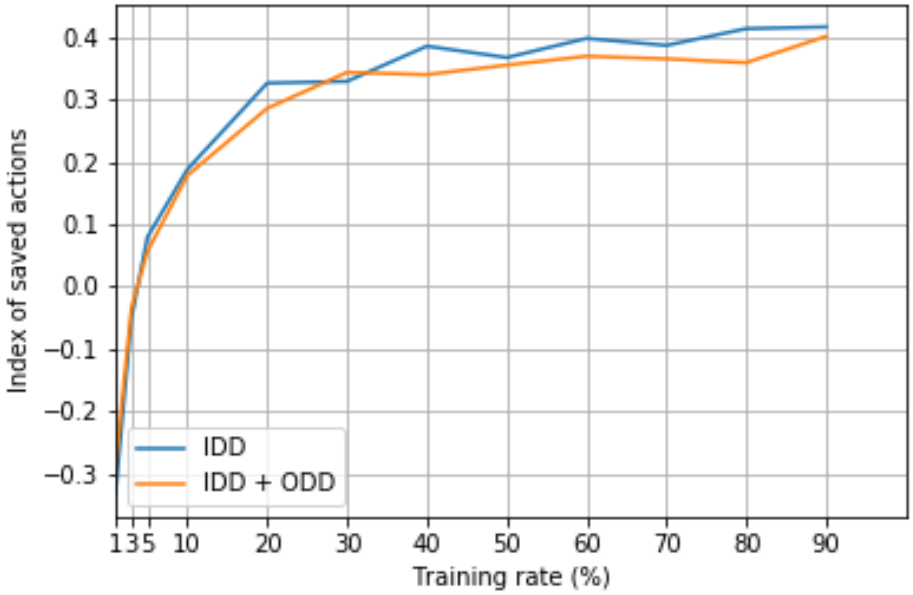


Figure C.15: The curves of the index of saved actions when training the model with only in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are associated with the training rates of CD-6 (see table C.15).

Training Rate (%)	IDD	IDD + ODD
1	-0.79	-0.8
3	-0.43	-0.36
5	-0.22	-0.27
10	-0.19	-0.10
20	-0.05	-0.05
30	-0.03	0.01
40	0.08	0.02
50	0.13	0.11
60	0.16	0.16
70	0.15	0.18
80	0.10	0.17
90	0.10	0.09

Table C.16: The values of the index of saved actions when training the model with only in-domain data and in-domain data in conjunction with out-domain data for the different training rates. These values are associated with CD-7.

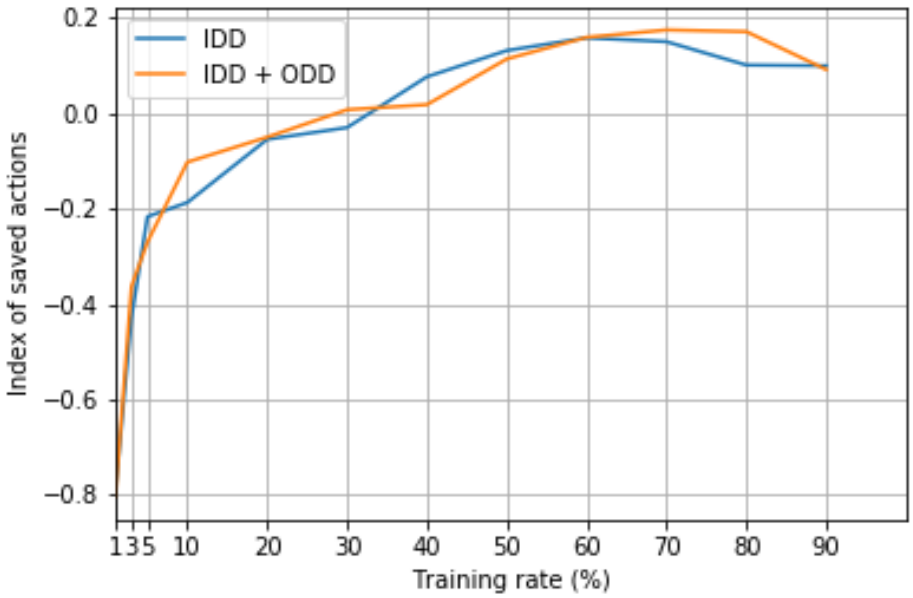


Figure C.16: The curves of the index of saved actions when training the model with only in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are associated with the training rates of CD-7 (see table C.16).

Training Rate (%)	IDD	IDD + ODD
1	-0.85	-0.82
3	-0.53	-0.61
5	-0.44	-0.42
10	-0.32	-0.41
20	-0.18	-0.17
30	-0.07	-0.22
40	0.01	-0.08
50	-0.07	-0.08
60	-0.03	-0.06
70	-0.01	0.07
80	0.33	-0.30
90	0.15	0.12

Table C.17: The values of the index of saved actions when training the model with only in-domain data and in-domain data in conjunction with out-domain data for the different training rates. These values are associated with CD-8.

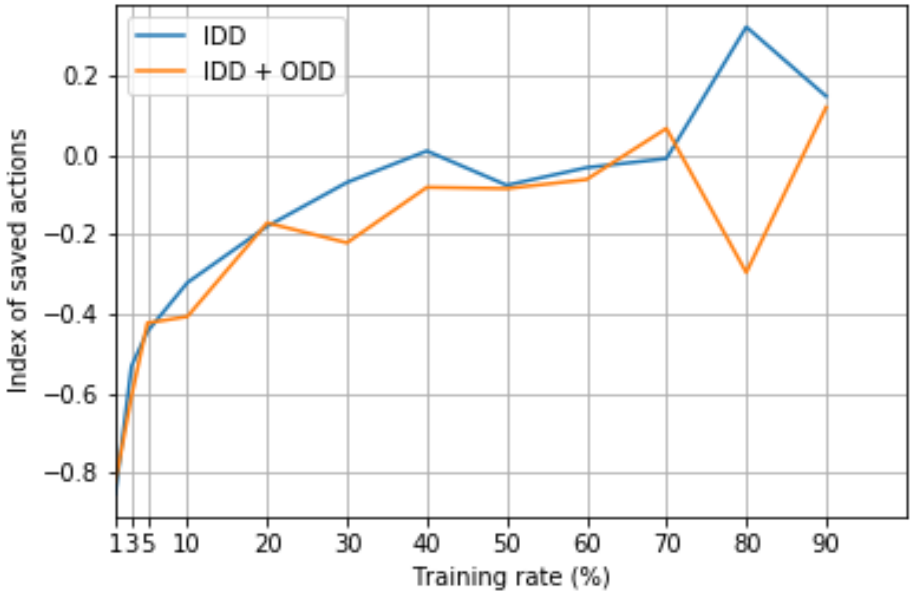


Figure C.17: The curves of the index of saved actions when training the model with only in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are associated with the training rates of CD-8 (see table C.17).

Training Rate (%)	IDD	IDD + ODD
1	-0.46	-0.4
3	-0.05	-0.06
5	0.06	0.05
10	0.21	0.18
20	0.28	0.23
30	0.36	0.29
40	0.13	0.29
50	0.32	0.3
60	0.37	0.33
70	0.4	0.36
80	0.45	0.36
90	0.44	0.28

Table C.18: The values of the index of saved actions when training the model with only in-domain data and in-domain data in conjunction with out-domain data for the different training rates. These values are associated with CD-9.

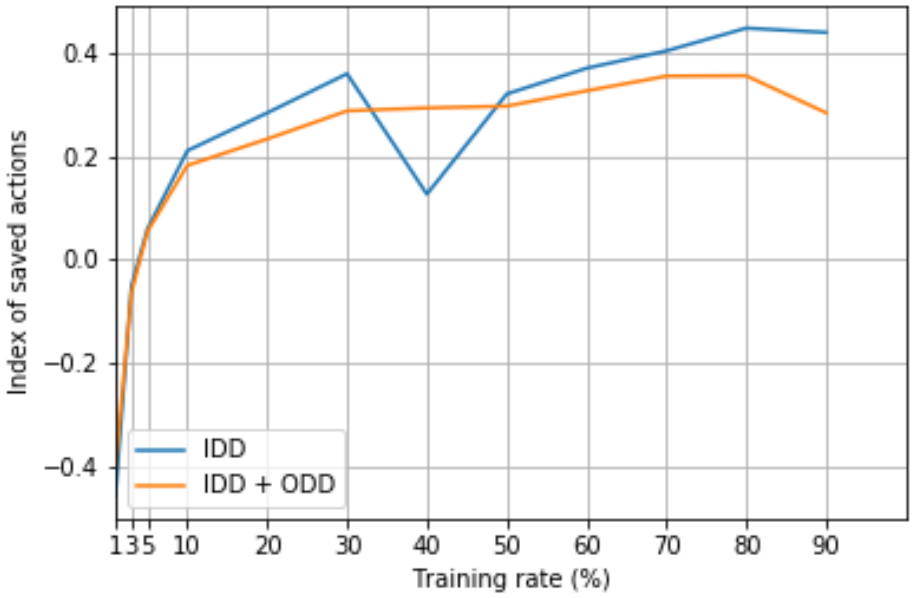


Figure C.18: The curves of the index of saved actions when training the model with only in-domain data (blue line) and in-domain data in conjunction with out-domain data (orange line). These values are associated with the training rates of CD-9 (see table C.18).

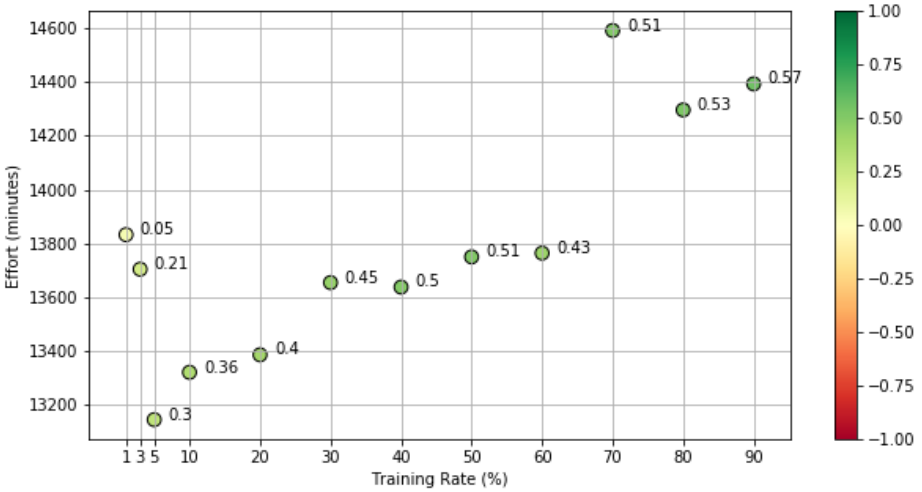


Figure C.19: A scatter plot corresponding to the relation between the total effort and the index of saved actions (mapped to the color) when the models are trained using only in-domain data. These values are associated with the training rates of the training rates of CD-1.

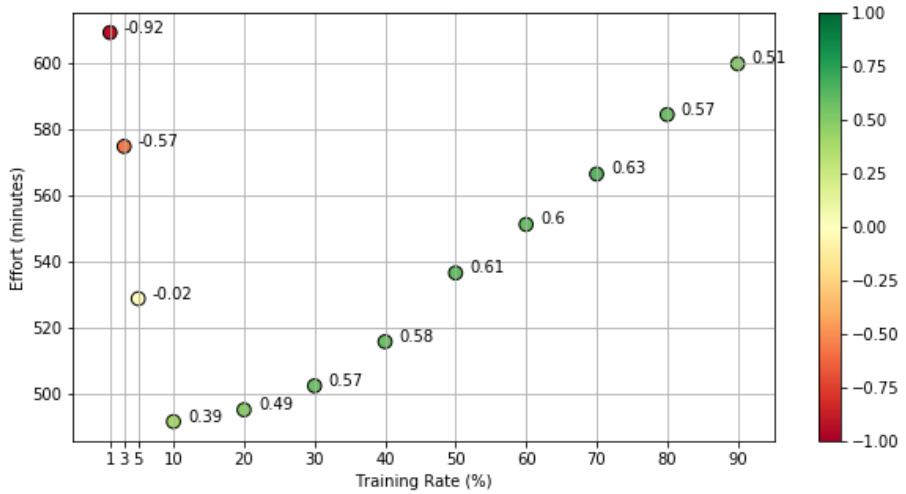


Figure C.20: A scatter plot corresponding to the relation between the total effort and the index of saved actions (mapped to the color) when the models are trained using only in-domain data. These values are associated with the training rates of the training rates of CD-2.

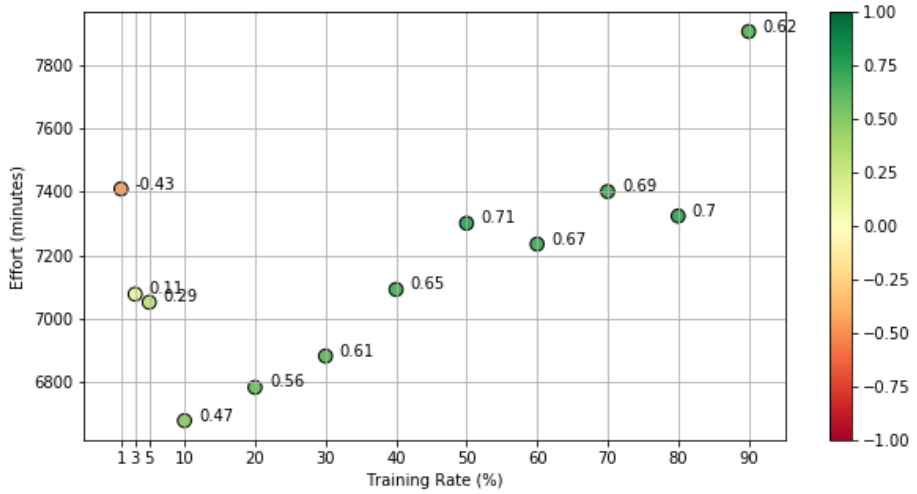


Figure C.21: A scatter plot corresponding to the relation between the total effort and the index of saved actions (mapped to the color) when the models are trained using only in-domain data. These values are associated with the training rates of the training rates of CD-3.

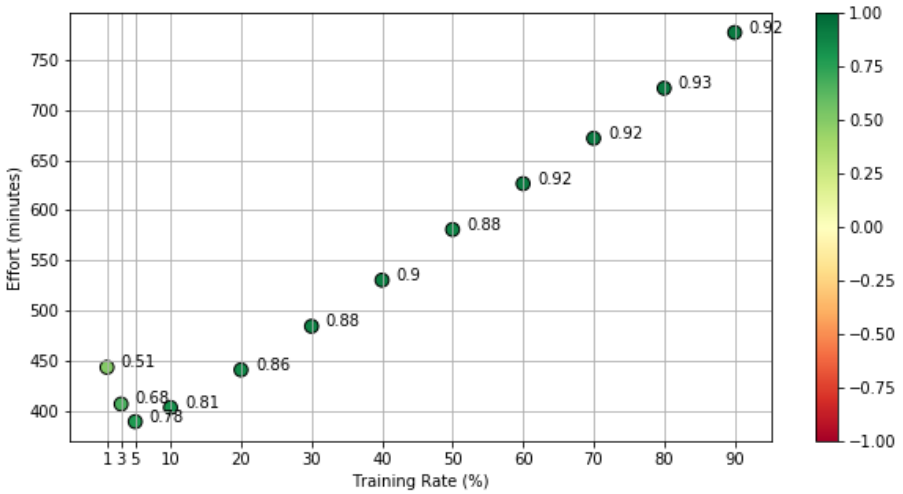


Figure C.22: A scatter plot corresponding to the relation between the total effort and the index of saved actions (mapped to the color) when the models are trained using only in-domain data. These values are associated with the training rates of the training rates of CD-4.

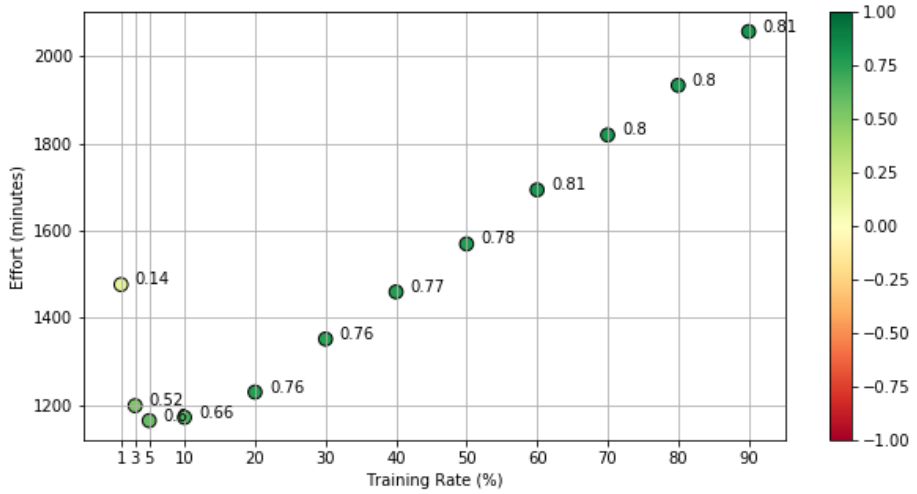


Figure C.23: A scatter plot corresponding to the relation between the total effort and the index of saved actions (mapped to the color) when the models are trained using only in-domain data. These values are associated with the training rates of the training rates of CD-5.

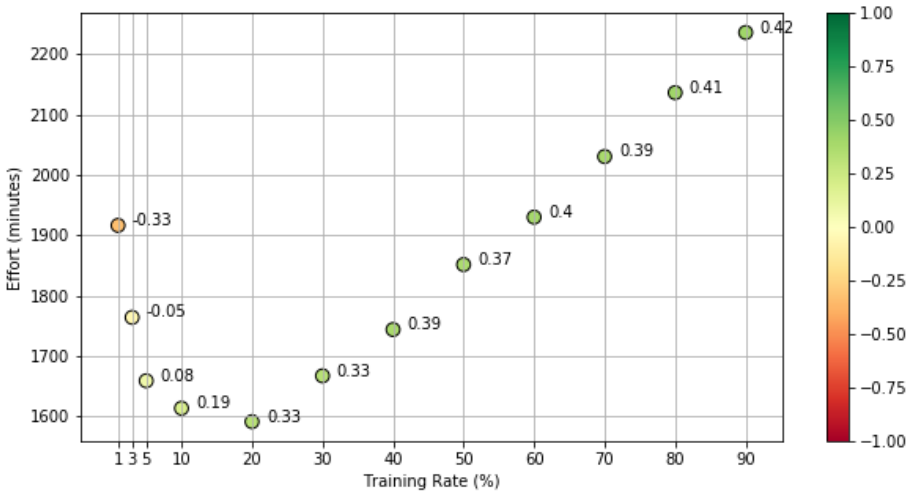


Figure C.24: A scatter plot corresponding to the relation between the total effort and the index of saved actions (mapped to the color) when the models are trained using only in-domain data. These values are associated with the training rates of the training rates of CD-6.

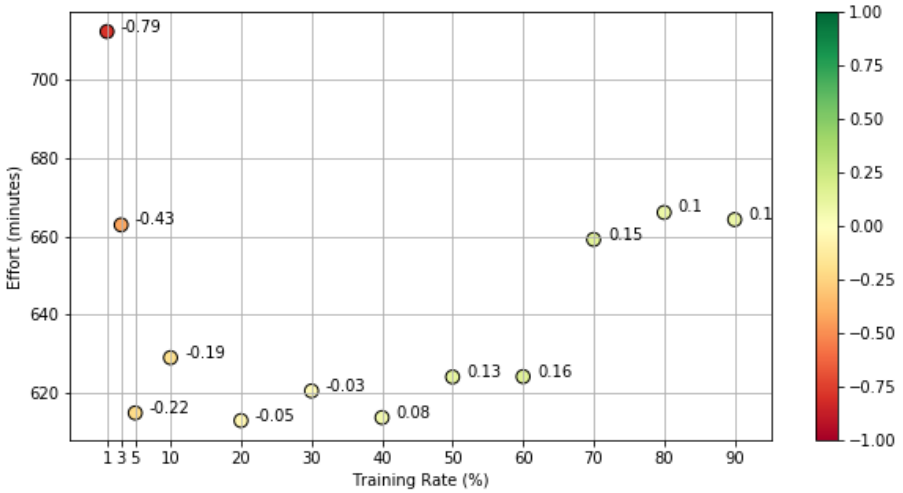


Figure C.25: A scatter plot corresponding to the relation between the total effort and the index of saved actions (mapped to the color) when the models are trained using only in-domain data. These values are associated with the training rates of the training rates of CD-7.

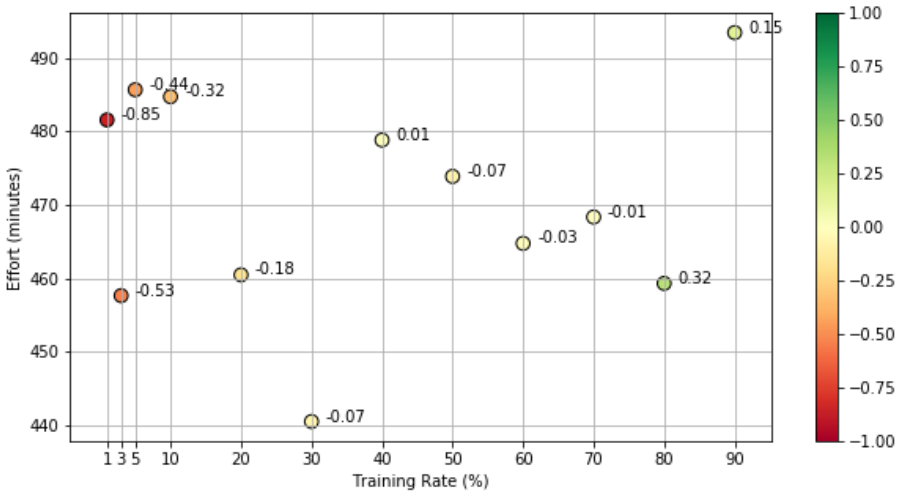


Figure C.26: A scatter plot corresponding to the relation between the total effort and the index of saved actions (mapped to the color) when the models are trained using only in-domain data. These values are associated with the training rates of the training rates of CD-8.

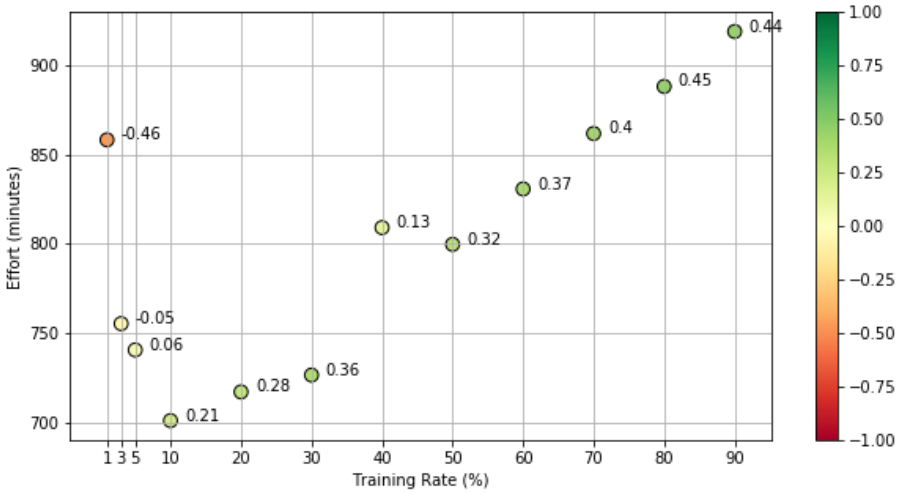


Figure C.27: A scatter plot corresponding to the relation between the total effort and the index of saved actions (mapped to the color) when the models are trained using only in-domain data. These values are associated with the training rates of the training rates of CD-9.

Appendix D

Simulated Crowdsourcing Scenario

# of HITs	Generation Effort	Validation Effort	Total Effort
100	16:26	204:56	221:22
200	32:04	194:32	226:37
300	47:57	176:12	224:10
400	68:21	162:53	231:14
500	82:09	144:38	226:47

Table D.1: The values of effort for the different amount of training data in terms of the absolute number of HITs for CD-1. These values are associated with training the models with in-domain data.

# of HITs	Generation Effort	Validation Effort	Total Effort
100	00:32	07:43	08:15
200	01:09	06:54	08:04
300	01:42	06:17	08:00
400	02:21	06:08	08:30
500	02:58	05:38	08:37

Table D.2: The values of effort for the different amount of training data in terms of the absolute number of HITs for CD-2. These values are associated with training the models with in-domain data.

# of HITs	Generation Effort	Validation Effort	Total Effort
100	32:38	84:14	116:52
200	62:24	55:44	118:08
300	92:45	32:27	125:12
400	123:50	07:31	131:22

Table D.3: The values of effort for the different amount of training data in terms of the absolute number of HITs for CD-3. These values are associated with training the models with in-domain data.

# of HITs	Generation Effort	Validation Effort	Total Effort
100	00:40	05:48	06:28
200	01:21	05:31	06:53
300	01:57	05:00	06:58
400	02:39	04:42	07:21
500	03:19	04:21	07:41

Table D.4: The values of effort for the different amount of training data in terms of the absolute number of HITs for CD-4. These values are associated with training the models with in-domain data.

# of HITs	Generation Effort	Validation Effort	Total Effort
100	01:13	19:12	20:26
200	02:23	17:02	19:25
300	03:25	16:38	20:04
400	04:37	14:36	19:13
500	05:50	14:00	19:50

Table D.5: The values of effort for the different amount of training data in terms of the absolute number of HITs for CD-5. These values are associated with training the models with in-domain data.

# of HITs	Generation Effort	Validation Effort	Total Effort
100	00:52	28:59	29:51
200	01:42	26:13	27:55
300	02:33	24:15	26:49
400	03:24	22:56	26:21
500	04:18	21:58	26:17

Table D.6: The values of effort for the different amount of training data in terms of the absolute number of HITs for CD-6. These values are associated with training the models with in-domain data.

# of HITs	Generation Effort	Validation Effort	Total Effort
100	03:21	06:50	10:12
200	06:09	04:03	10:13
300	09:46	01:20	11:06

Table D.7: The values of effort for the different amount of training data in terms of the absolute number of HITs for CD-7. These values are associated with training the models with in-domain data.

# of HITs	Generation Effort	Validation Effort	Total Effort
100	02:15	05:29	07:45
200	04:22	03:10	07:32
300	06:30	01:17	07:47

Table D.8: The values of effort for the different amount of training data in terms of the absolute number of HITs for CD-8. These values are associated with training the models with in-domain data.

# of HITs	Generation Effort	Validation Effort	Total Effort
100	00:47	11:01	11:49
200	01:38	10:06	11:45
300	02:27	09:12	11:40
400	03:21	08:33	11:54
500	04:03	07:46	11:50

Table D.9: The values of effort for the different amount of training data in terms of the absolute number of HITs for CD-9. These values are associated with training the models with in-domain data.

# of HITs	Absolute Effort Savings	Relative Effort Savings	ρ
100	17:34	7.35	0.30
200	12:19	5.16	0.33
300	14:46	6.18	0.41
400	07:42	3.22	0.45
500	12:09	5.09	0.41

Table D.10: The absolute effort savings, relative effort savings, and index of saved actions for the different amount of training rates. These values are associated with CD-1.

# of HITs	Absolute Effort Savings	Relative Effort Savings	ρ
100	02:05	20.14	0.24
200	02:16	22.06	0.47
300	02:20	22.70	0.54
400	01:50	17.83	0.46
500	01:43	16.71	0.50

Table D.11: The absolute effort savings, relative effort savings, and index of saved actions for the different amount of training rates. These values are associated with CD-2.

# of HITs	Absolute Effort Savings	Relative Effort Savings	ρ
100	11:07	8.69	0.53
200	09:51	7.71	0.55
300	02:47	2.18	0.56
400	-04:37	-2.64	0.59

Table D.12: The absolute effort savings, relative effort savings, and index of saved actions for the different amount of training rates. These values are associated with CD-3.

# of HITs	Absolute Effort Savings	Relative Effort Savings	ρ
100	07:14	52.74	0.78
200	06:49	49.78	0.79
300	06:44	49.17	0.84
400	06:21	46.33	0.85
500	06:01	43.92	0.84

Table D.13: The absolute effort savings, relative effort savings, and index of saved actions for the different amount of training rates. These values are associated with CD-4.

# of HITs	Absolute Effort Savings	Relative Effort Savings	ρ
100	15:42	43.48	0.49
200	16:43	46.24	0.62
300	16:04	44.49	0.62
400	16:55	46.81	0.72
500	16:18	45.1	0.72

Table D.14: The absolute effort savings, relative effort savings, and index of saved actions for the different amount of training rates. These values are associated with CD-5.

# of HITs	Absolute Effort Savings	Relative Effort Savings	ρ
100	09:05	23.32	-0.12
200	11:01	28.29	0.07
300	12:07	31.12	0.15
400	12:35	32.34	0.20
500	12:39	32.51	0.24

Table D.15: The absolute effort savings, relative effort savings, and index of saved actions for the different amount of training rates. These values are associated with CD-6.

# of HITs	Absolute Effort Savings	Relative Effort Savings	ρ
100	01:21	11.78	0.03
200	01:20	11.60	0.11
300	00:27	3.9	0.15

Table D.16: The absolute effort savings, relative effort savings, and index of saved actions for the different amount of training rates. These values are associated with CD-7.

# of HITs	Absolute Effort Savings	Relative Effort Savings	ρ
100	00:21	4.48	-0.09
200	00:34	7.01	0.05
300	00:19	3.92	0.22

Table D.17: The absolute effort savings, relative effort savings, and index of saved actions for the different amount of training rates. These values are associated with CD-8.

# of HITs	Absolute Effort Savings	Relative Effort Savings	ρ
100	04:01	25.43	0.11
200	04:05	25.84	0.23
300	04:10	26.37	0.28
400	03:56	24.83	0.30
500	04:00	25.29	0.33

Table D.18: The absolute effort savings, relative effort savings, and index of saved actions for the different amount of training rates. These values are associated with CD-9.